Evaluation of Robust Deep Learning Pipelines Targeting Low SWaP Edge Deployment

A Thesis Presented for the

Master of Science

Degree

The University of Tennessee, Knoxville

David Carter Cornett

December 2021

Copyright © 2021 by David Carter Cornett
All rights reserved.

This thesis is dedicated to the memory of my father, David Cornett Jr., who taught me to always think for myself and would remind me to "kick it in" when the last stretch seems like an insurmountable task,

and to the memory of Billie Lee Bolton whose encouraging conversations and stories of working in conjunction with the Manhattan Project motivated me to pursue my dreams of working in research.

Acknowledgments

Foremost, I would like to give special thanks to my family for all of their encouragement and support, in particular my wife Barbara, without whom I would have never had the courage to make it this far.

Particular thanks is needed for my advisor, Dr. Mongi Abidi, in providing me the guidance and feedback to navigate my graduate studies. I would also like to thank three other Imaging, Robotics and Intelligent Systems (IRIS) lab students: Steven Clukey for getting me started in this work, Samuel "Clay" Leach and Vijaysrinivas "VJ" Rajagopal for all the long hours we've spent together in the lab.

Abstract

The deep learning technique of convolutional neural networks (CNNs) has greatly advanced the state-of-the-art for computer vision tasks such as image classification and object detection. These solutions rely on large systems leveraging wattage-hungry GPUs to provide the computational power to achieve such performance. However, the size, weight and power (SWaP) requirements of these conventional GPU-based deep learning systems are not suitable when a solution requires deployment to so called "Edge" environments such as autonomous vehicles, unmanned aerial vehicles (UAVs) and smart security cameras.

The objective of this work is to benchmark FPGA-based alternatives to conventional GPU systems that have the potential to offer similar CNN inference performance while being delivered in a low SWaP platform suitable for Edge deployment. In this thesis we create equivalent pipelines for both GPU and FPGA which implement deep learning models for both image classification and object detection tasks. Beyond baseline benchmarking, we additionally quantify the impact on inference performance of two common real-world image degradation scenarios (simulated contrast reduced capture and salt-and-pepper sensor noise) and their associated correction methods (gamma correction and median kernel filtering) we selected as illustrative examples. The baseline system analysis, coupled with these additional robustness evaluations, provides a statistically significant benchmark comparison targeting a breadth of interest for the computer vision community.

We have conducted the following experiments to demonstrate the FPGA as an effective alternative to the GPU implementation when deployed to Edge environments: (1) we developed a hardware video processing architecture with an associated library of hardware processing functions to prototype a base FPGA ecosystem, (2) we established through benchmarking that two common CNN models (ResNet-50 and YOLO version 3) have a mere 1% drop in performance on FPGA versus GPU, (3) we show a quantitative baseline analysis for the image degradation/correction on the associated testing datasets, and (4) we proved that our FPGA-based computer vision system is an ideal platform for Edge deployment given its comparable robustness to input degradation when optimal correction is applied.

The significance of these findings is the demonstration of our FPGA-based solution as the superior candidate for Edge deployed vision systems evidenced by our experiments which illustrate its competitive inference performance to the conventional GPU solution and its equivalent robustness provided by correction methods to noise encountered during in-the-wild imaging while being delivered with far lower SWaP requirements.

Table of Contents

1 Introduction **1**
- 1.1 Motivation . . . 1
- 1.2 Pipeline . . . 4
- 1.3 Contributions . . . 4
- 1.4 Synopsis . . . 8

2 Literature Review **13**
- 2.1 Computer Vision with Edge Systems . . . 13
  - 2.1.1 Overview of Edge Systems . . . 14
  - 2.1.2 Options for Edge Implementation . . . 16
- 2.2 Review of Video Processing Systems in FPGAs . . . 19
  - 2.2.1 FPGA and Zynq Architecture Overview . . . 19
  - 2.2.2 Frameworks for Fast FPGA Prototyping . . . 20
  - 2.2.3 FPGA Video System Design . . . 25
- 2.3 Review of FPGA Implementations of Deep Learning Networks . . . 27
  - 2.3.1 Deep Learning Overview . . . 27
  - 2.3.2 Previous Work in FPGA-Based Deep Learning . . . 31
- 2.4 Impact of Image Quality on CNN Performance . . . 32
  - 2.4.1 Image Degradation and CNN Performance . . . 32
  - 2.4.2 Corrective Measures for Noise Reduction . . . 33
  - 2.4.3 Quantitative Image Quality Metrics . . . 34

3 Video Processing on FPGAs **37**
- 3.1 FPGA Video Pipeline Architectures Developed . . . 37
- 3.2 Library of Video Processing Cores Developed . . . 41

4 Benchmarking of CNNs in FPGA versus GPU **52**
- 4.1 ResNet-50 FPGA Architecture . . . 52
- 4.2 ResNet-50 FPGA versus GPU Benchmarking . . . 55
- 4.3 YOLO Version 3 FPGA Architecture . . . 58
- 4.4 YOLO Version 3 FPGA versus GPU Benchmarking . . . 59
- 4.5 Findings From Benchmarking FPGA-based and GPU-based CNNs . . . 61

5 Image Degradation and Correction **65**
5.1 Image Degradation Experimental Setup and Evaluation 66
5.2 Image Correction Experimental Setup and Evaluation 71
5.3 Findings From Image Degradation and Correction Experiments 91

6 Full Pipeline Experimental Benchmarking Results **96**
6.1 Pipeline 1: ResNet-50 Evaluation 96
6.1.1 Comparison of Contrast Reduction Degradation on GPU and FPGA 97
6.1.2 Comparison of Salt-and-Pepper Degradation on GPU and FPGA . . 107
6.1.3 Improvement of Accuracy with Gamma Correction Applied to Contrast Reduction Degradation on GPU and FPGA 117
6.1.4 Improvement of Accuracy with Median Filter Correction Applied to Salt-and-Pepper Degradation on GPU and FPGA 120
6.1.5 Experimental Results of ResNet-50 Benchmarking 124
6.2 Pipeline 2: YOLO Version 3 Evaluation 128
6.2.1 Comparison of Contrast Reduction Degradation on GPU and FPGA 131
6.2.2 Comparison of Salt-and-Pepper Degradation on GPU and FPGA . . 134
6.2.3 Improvement of Precision with Gamma Correction Applied to Contrast Reduction Degradation on GPU and FPGA 140
6.2.4 Improvement of Precision with Median Filter Correction Applied to Salt-and-Pepper Degradation on GPU and FPGA 150
6.2.5 Experimental Results of YOLO Version 3 Benchmarking 153
6.3 Findings From Pipeline Experimentation Results 157

7 Findings and Recommendations **161**
7.1 Summary 161
7.2 Contributions 164
7.3 Future Work 167

Bibliography **169**

Appendix **176**
A Library of Custom FPGA IP Cores 177

Vita **200**

List of Tables

2.1 **Comparison of Edge AI Hardware devices.** *For Edge deployment, the device must not only meet inference performance parameters but also physical parameters required by the system. Each device's parameters is ranked from poorest to optimal (worst, bad, good, best).* 17

2.2 **Comparison of FPGA development board resources.** *Although each of the summarized FPGA chips are from the Xilinx Zynq line, they differ greatly in both the amount of configurable resources and in their support for video processing.* 22

3.1 **Iterations of video processing systems.** *Over the course of the thesis work, four iterations of the video processing pipeline were developed which targeted three different FPGA development boards.* 40

3.2 **List of implemented video processing IP cores and associated FPGA boards.** *The majority of our custom IP cores have been implemented across all FPGA boards and custom video processing pipelines explored in this thesis.* 45

3.3 **Comparison of FPGA board resources utilization for Stream Passthrough IP core.** 46

3.4 **Comparison of FPGA board resources utilization for Pixel IP core.** 48

3.5 **Comparison of FPGA board resources utilization for Grayscale IP core.** 50

4.1 **ResNet-50 model sizes before and after quantization.** *The size of the original floating point ResNet-50 model is reduced by 80% via the quantization method leveraged by the DNNDK tool. This quantized model is used to build the hardware DPU (deep processing unit) for FPGA.* 54

4.2 **Comparison of GPU vs. FPGA performance on ResNet-50.** *The GPU model leverages 32-bit floating point weights whereas the FPGA model is based on 8-bit integer weights produced by a quantization method.* 56

4.3 **Comparison of GPU v.s FPGA considering SWaP (size, weight and power) characteristics in relation to accuracy performance with ResNet-50 model.** *Although an FPGA is a self-contained unit, the GPU requires other support hardware to run which is indicated by the * for additional computing ecosystem hardware. Also of note is that for this implementation the FPGA wattage** includes all I/O resources.* 60

4.4 **YOLO version 3 model sizes before and after quantization.** *The size of the original floating point YOLO version 3 model is reduced by 78.75% via the quantization method leveraged by the DNNDK tool. This quantized model is used to build the hardware DPU (deep processing unit) for FPGA.* 60

4.5 **Comparison of GPU vs. FPGA performance on YOLO version 3.** *The GPU model leverages 32-bit floating point weights whereas the FPGA model is based on 8-bit integer weights produced by a quantization method.* . . 62

4.6 **Comparison of GPU v.s FPGA considering SWaP (size, weight and power) characteristics in relation to accuracy performance with YOLO version 3 model.** *Although an FPGA is a self-contained unit, the GPU requires other support hardware to run which is indicated by the * for additional computing ecosystem hardware. Also of note is that for this implementation the FPGA wattage** includes all I/O resources.* 63

5.1 **Comparison of experimental findings for image degradation and correction on ImageNet-2012 and VOC2007 datasets.** *This table details examples from both the contrast reduction and contrast reduction gamma corrected experiments as well as the salt-and-pepper and salt-and-pepper median filter corrected experiments. For each of the two datasets we provide the average SSIM score on degradation with a specific parameter as well as the improved average SSIM score with the optimal correction parameter. Leveraging the optimal parameter for correction is key in the final chapter's experiments when we wish to maximize CNN inference accuracy. For both datasets contrast reduction beyond 5% and salt-and-pepper degradation beyond 0.1 probability were found to be difficult to correct such that any significant recovery in SSIM score could be achieved. This would indicate that after extreme image degradation that improvement for CNN inference accuracy is unlikely.* 95

6.1 **Comparison of contrast reduction degradation and gamma correction on ImageNet testing dataset of GPU (32-bit floating point model) vs. FPGA (8-bit integer quantized model) performance on ResNet-50.** *This table outlines seven experiments to provide an overview of both the GPU and FPGA pipeline performances. For each of these experiments we also display the associated average SSIM score for the quality of the degraded or corrected images being evaluated in the ImageNet dataset. As demonstrated in Chapter 4, the GPU and FPGA implementations have near identical performance at the base experiment where no contrast reduction degradation is introduced (Base experiment in the table). For example, with contrast reduction degradation of 20% applied the GPU accuracy drops from 74.44% to 61.48% (a 12.96% decrease) and the associated SSIM scores drops from 1 to 0.2563 (a decrease of 0.7437). For the same contrast degradation with the FPGA, accuracy drops from 73.34% to 53.42% (a 19.92% decrease) with the same SSIM score to reflect testing dataset quality. However, after optimal gamma correction is applied the GPU accuracy returns to 70.08% (an 8.6% increase) and the FPGA returns to 66.25% (a 12.83% increase) and the SSIM score for the corrected dataset becomes 0.8901 (an increase of 0.6338). As theorized, the SSIM score can be used as a predictor for inference performance. These experiments demonstrate that the FPGA model can achieve comparable accuracy to the GPU model when optimal gamma correction is applied to contrast reduction degradation. Although the FPGA model provides slightly lower accuracy for this type of degradation it comes with the advantage of superior SWaP characteristics better suited for Edge applications.* 129

6.2 **Comparison of salt-and-pepper degradation and median filter correction on ImageNet testing dataset of GPU (32-bit floating point model) vs. FPGA (8-bit integer quantized model) performance on ResNet-50.** *This table outlines seven experiments to provide an overview of both the GPU and FPGA pipeline performances. For each of these experiments we also display the associated average SSIM score for the quality of the degraded or corrected images being evaluated in the ImageNet dataset. As demonstrated in Chapter 4, the GPU and FPGA implementations have near identical performance at the base experiment where no salt-and-pepper degradation is introduced (Base experiment in the table). For example, with salt-and-pepper degradation of 0.05 the GPU accuracy drops from 74.44% to 39.06% (a 35.38% decrease) and the associated SSIM scores drops from 1 to 0.2545 (a decrease of 0.7455). For the same salt-and-pepper degradation with FPGA, accuracy drops from 73.34% to 37.8% (a 35.53% decrease) with the same SSIM score to reflect testing dataset quality. However, after optimal median filter correction is applied the GPU accuracy returns to 72.74% (a 33.68% increase) and the FPGA returns to 71.86% (a 34.06% increase) and the SSIM score for the corrected dataset becomes 0.8381 (an increase of 0.5836). As theorized, the SSIM score can be used as a predictor for inference performance. These experiments demonstrate that the FPGA model can achieve near identical accuracy to the GPU model when optimal median filter correction is applied to salt-and-pepper degradation. Not only does the FPGA model provides near identical robustness for this type of degradation it comes with the advantage of superior SWaP characteristics better suited for Edge applications.* 130

6.3 **Comparison of contrast reduction degradation and gamma correction on VOC2007 testing dataset of GPU (32-bit floating point model) vs. FPGA (8-bit integer quantized model) performance on YOLO version 3.** *This table outlines seven experiments to provide an overview of both the GPU and FPGA pipeline performances. For each of these experiments we also display the associated average SSIM score for the quality of the degraded or corrected images being evaluated in the ImageNet dataset. As demonstrated in Chapter 4, the GPU and FPGA implementations have near identical performance at the base experiment where no contrast reduction degradation is introduced (Base experiment in the table). For example, with contrast reduction degradation of 13.3% applied the GPU mAP drops from 78.46% to 70.94% (a 7.52% decrease) and the associated SSIM scores drops from 1 to 0.1512 (a decrease of 0.8488). For the same contrast degradation with the FPGA, mAP drops from 77.44% to 65.34% (a 12.1% decrease) with the same SSIM score to reflect testing dataset quality. However, after optimal gamma correction is applied the GPU mAP returns to 75.90% (an 5.04% increase) and the FPGA returns to 74.92% (a 9.58% increase) and the SSIM score for the corrected dataset becomes 0.6996 (an increase of 0.5484). As theorized, the SSIM score can be used as a predictor for inference performance. These experiments demonstrate that the FPGA model can achieve comparable mAP to the GPU model when optimal gamma correction is applied to contrast reduction degradation. Although the FPGA model provides slightly lower mAP for this type of degradation it comes with the advantage of superior SWaP characteristics better suited for Edge applications.* 158

6.4 **Comparison of salt-and-pepper degradation and median filter correction on VOC2007 testing dataset of GPU (32-bit floating point model) vs. FPGA (8-bit integer quantized model) performance on YOLO version 3.** *This table outlines seven experiments to provide an overview of both the GPU and FPGA pipeline performances. For each of these experiments we also display the associated average SSIM score for the quality of the degraded or corrected images being evaluated in the ImageNet dataset. As demonstrated in Chapter 4, the GPU and FPGA implementations have near identical performance at the base experiment where no salt-and-pepper degradation is introduced (Base experiment in the table). For example, with salt-and-pepper degradation of 0.05 the GPU mAP drops from 78.46% to 40.69% (a 37.77% decrease) and the associated SSIM scores drops from 1 to 0.2601 (a decrease of 0.7399). For the same salt-and-pepper degradation with FPGA, mAP drops from 77.44% to 42.37% (a 35.07% decrease) with the same SSIM score to reflect testing dataset quality. However, after optimal median filter correction is applied the GPU mAP returns to 74.52% (a 33.83% increase) and the FPGA returns to 74.13% (a 31.76% increase) and the SSIM score for the corrected dataset becomes 0.8075 (an increase of 0.5474). As theorized, the SSIM score can be used as a predictor for inference performance. These experiments demonstrate that the FPGA model can achieve near identical mAP to the GPU model when optimal median filter correction is applied to salt-and-pepper degradation. Not only does the FPGA model provides near identical robustness for this type of degradation it comes with the advantage of superior SWaP characteristics better suited for Edge applications.* 159

1 **Comparison of FPGA board resources utilization for Sobel IP core.** 179
2 **Comparison of FPGA board resources utilization for Threshold IP core.** 180
3 **Comparison of FPGA board resources utilization for Sobel-Threshold IP core.** 183
4 **Comparison of FPGA board resources utilization for Dilation IP core.** 185
5 **Comparison of FPGA board resources utilization for Erosion IP core.** 188
6 **Comparison of FPGA board resources utilization for Inversion IP core.** 189
7 **Comparison of FPGA board resources utilization for Gaussian Blur IP core.** 193
8 **Comparison of FPGA board resources utilization for Sharpen IP core.** 194
9 **Comparison of FPGA board resources utilization for Contrast Reduction (pixel operation) IP core.** 196
10 **Comparison of FPGA board resources utilization for Convolution IP core.** 199

List of Figures

1.1 **Examples of Edge applications.** *In (a) an autonomous vehicle's Edge sensors are tested under simulated heavy rain conditions at the Singapore-MIT testing center. The image in (b) demonstrates how cameras used for surveillance are often deployed in areas where illumination conditions are uncontrolled and highly variable. The UAV in (c) is deployed during a snow storm. Such conditions provide both mechanical challenges as well as add degradations to the frame quality of the video payload system.* 3

1.2 **Pipeline for FPGA versus GPU benchmarking of CNN models with optional image degradation and correction.** *From a high level the pipeline can be viewed as a GPU processing path (upper portion) and an FPGA processing path (lower portion) having common input (dataset image) and output (Edge benchmark analysis) endpoints. The sections of the pipeline corresponding to specific thesis contributions are marked with different color boxes (see figure contribution legend). The yellow box (contribution 1) represents the integration and extension of the following minor contributions into our major contribution of an Edge benchmarking framework evaluating our FPGA-based and GPU-based systems to prove which is optimal for Edge AI deployment. The red box (contribution 2) represents our FPGA video processing system. The green box (contribution 3) represents our FPGA CNN models which are used in benchmarking against GPU models. The blue box (contribution 4) represents our statistical analysis of the quality responses from our evaluated degradations and corrections on our selected datasets. The pipeline is organized such that different experimental setups can be evaluated. This experimental configuration is achieved by first selecting a CNN and associated dataset, then determining whether an unaltered input is to be benchmarked (original dataset image) or by selecting to apply one of the two degradation noises and/or applying an associated correction function to the degradation. Once the experiment parameters are determined, the comparison begins with the selection of a testing image from the chosen dataset. Next, depending on the experimental setup, we choose whether to degrade the image with one of the possible degradation noises. Once again depending on the parameters of the experiment, an associated correction function is optionally applied to the image. At this point the now processed test image is evaluated by the equivalent models in the GPU and FPGA paths as well as to be compared to the unaltered input image downstream to calculate a SSIM score. Depending on the model, the appropriate method is selected to evaluate the individual inference predictions related to ground truth. Lastly, the performance of the GPU and FPGA predictions, the related input SSIM scores, and the SWaP characteristics of the two platforms are compiled to provide Edge suitability benchmarking comparison of the experimental setup.* 5

1.3 **ResNet-50 example testing image with contrast reduction degradation (20.0%) and gamma correction (1.5).** *The original image before degradation appears in the top left. The top right image shows the result after contrast reduction is applied with a parameter of 20.0%. Both the GPU and FPGA failed to properly classify the image when contrast reduction degradation at 20.0% was applied. However, as the bottom two images demonstrate, both the GPU and FPGA models provide the correct inference class label (lighthouse) when a gamma correction of 1.5 is applied to the contrast reduced image.* 11

1.4 **YOLO version 3 example testing image with salt-and-pepper degradation (0.05) and median filter correction (3x3).** *The original image before degradation appears in the top left. The top right image shows the result after salt-and-pepper is applied with a parameter of 0.05. Both the GPU and FPGA failed to properly detect the objects in the image when salt-and-pepper degradation at 0.05 was applied. However, as the bottom two images demonstrate, both the GPU and FPGA models provide the correct inference class label (2 instance of class bicycle) when a median filter correction of 3x3 is applied to the salt-and-pepper degraded image.* 12

2.1 **Deployment options for deep learning applications.** *Traditionally, deep learning applications are used on-site on individual workstations or in high-performance computing (HPC) centers. Due to increases in high-speed internet communication, Cloud-based solutions allow projects with limited budgets to affordably lease remotely located equipment. Advances in hardware which allow deep learning on smaller physical devices have created so called Edge AI platforms to service systems deployed into the real world such as autonomous cars, drones, medical devices, robotics and surveillance. The focus of our research in this thesis lies within this domain of Edge AI devices.* . . 15

2.2 **Spectrum of Edge AI platforms.** *Platforms to the left are more flexible and are easier to program. Platforms to the right achieve higher performance efficiency in terms of size, weight, power and inference speed. In choosing the platform used for Edge deployment one should consider all of the parameters of the project ranging from size, cost, power consumption and inference requirements.* 17

2.3 **The thesis work lies at the intersection of three research areas.** *The goal being to leverage the capabilities within these branches of research for the design and evaluation of robust, low SWaP computer vision applications.* . . 18

2.4 **Physical layout of FPGA fabric [1].** *(a) The FPGA fabric is composed of an array of logic blocks interconnected with high speed buses and I/O blocks. (b) A logic block typically contains a look-up-table and flip-flop combination.* 21

2.5 **High-level representation of Zynq architecture [2].** *The ARM CPU is located in the Processing System (PS) area while the FPGA fabric is housed in the Programmable Logic (PL) region. The communication between the two is achieved via tightly-coupled, high speed AXI connections.* 21

2.6 **Detail of Zynq architecture resource interconnections [2].** *The PS has connection to memory interfaces and fixed communication peripherals such as UART and the SD card. The PL system hardware accelerators can either communicate directly with several peripherals to enhance performance or communicate through the PS to work with PS specific peripherals.* 22

2.7 **The series of Xilinx FPGA development boards used in this thesis work.** *Clockwise from top-left: (1) Zybo Zynq-7000 Development Board [3], (2) Z1 Development Board [4], and (3) ZCU104 Ultrascale+ Development Board [5].* 23

2.8 **Typical FPGA development life cycle.** *An FPGA design starts with HDL code, then is synthesized into component netlist, any design constraints are added to the modeling, the physical layout of the netlist components are determined in floor planning, the place and route step then optimizes interconnections between gate level components while maintaining timing constraints, and finally a bitstream is generated which provides instructions for how the fabric is to be configured to implement the designed circuit.* . . . 26
2.9 **Xilinx AXI4-Stream table of slave interface signals [6].** *There are five signals used in the slave interface of the Xilinx AXI4 IP.* . . . 26
2.10 **Xilinx AXI4-Stream table of master interface signals [6].** *There are five signals used in the master interface of the Xilinx AXI4 IP.* . . . 26
2.11 **Xilinx AXI4-Stream handshaking [6].** *The AXI4-Stream interface uses a ready/valid handshaking protocol to flag the mark of a new frame being transmitted.* . . . 26
2.12 **Areas of artificial intelligence [7].** *Deep learning is a specific sub-branch of machine learning techniques.* . . . 28
2.13 **Recent milestones in object detection deep learning architectures [8].** *Traditional methods led in performance until the introduction of AlexNet in 2012. Current deep learning methods can be broadly categorized as either one-stage or two-stage detectors.* . . . 28
2.14 **Standard CNN architecture representing the standard layers [9].** *The feature learning portion of the network is shown on the left and the classification layers on the right.* . . . 36

3.1 **Custom video processing pipeline targeting Z1 FPGA board.** *The 9 principal IP cores associated with the video processing pipeline are outlined in red. These IP cores are detailed in other figures and text in this section. Of note is block 4 which is where video processing cores from our custom library are implemented.* . . . 39
3.2 **DVI to RGB Decoder and Video In to AXI4-Stream IP cores.** *The DVI to RGB Decoder IP core (left) converts the HDMI TMDS signal into a 24-bit RGB signal to be consumed downstream in the video processing pipeline. The Video In to AXI4-Stream IP core (right) converts the RGB signal in AXI4-Stream format which is leveraged by the internal IP cores within the video processing pipeline.* . . . 40
3.3 **Custom HLS Video Processing IP core.** *This IP core converts the RGB signal in AXI4-Stream format which is leveraged by the internal IP cores within the video processing pipeline.* . . . 42
3.4 **VDMA and Video Timing Controller IP cores.** *The VDMA IP core (left) is used for managing video frames in memory while the Video Timing Controller IP core (right) ensures proper syncing of all video frames throughout the entire pipeline.* . . . 42

3.5 **AXI4-Stream to Video Output and RGB to DVI Encoder IP cores.** *The AXI4-Stream to Video Output IP core (left) is used for converting the AXI4-Stream data back to 24-bit RGB and the RGB to DVI Encoder IP core (right) encodes the RGB signal, plus syncing signals, to be ready for transmission via TMDS.* . . . 43
3.6 **Stream Passthrough operation IP core flow chart.** *The HDMI source signal is captured via the AXI interconnect and stored as an input frame within the Zynq PL. This frame is then placed back onto the AXI interconnect and then to HDMI sink.* . . . 45
3.7 **Stream Passthrough IP core block diagram.** . . . 45
3.8 **Stream Passthrough IP timing/latency.** . . . 46
3.9 **Stream Passthrough IP hardware utilization.** . . . 46
3.10 **Stream Passthrough - grayscale image results (image source [10]).** . . . 46
3.11 **Stream Passthrough - color image results (image source [10]).** . . . 47
3.12 **Pixel Access Passthrough operation IP core flow chart.** *The HDMI source signal is captured via the AXI interconnect and stored as an input frame within the Zynq PL. This frame is then copied pixel-by-pixel into a second frame and then placed back onto the AXI interconnect and then to HDMI sink.* 47
3.13 **Pixel Access Passthrough IP core block diagram.** . . . 47
3.14 **Pixel Access Passthrough IP timing/latency.** . . . 47
3.15 **Pixel Access Passthrough IP hardware utilization.** . . . 47
3.16 **Pixel Access Passthrough - grayscale image results (image source [10]).** . . . 48
3.17 **Pixel Access Passthrough - color image results (image source [10]).** 48
3.18 **Grayscale operation IP core flow chart.** *The HDMI source signal is captured via the AXI interconnect and stored as an input frame within the Zynq PL. HLS code is then used to convert the R,G and B channels into grayscale equivalent. This value is then duplicated back to a 3-channel grayscale image and placed into the output frame before sending it out of the PL via AXI interconnect and then to HDMI sink.* . . . 50
3.19 **Grayscale IP core block diagram.** . . . 50
3.20 **Grayscale IP timing/latency.** . . . 50
3.21 **Grayscale IP hardware utilization.** . . . 50
3.22 **Grayscale - color image results.** . . . 51

4.1 **A standard residual layer in the ResNet architecture scheme [11].** . . . 54
4.2 **Layer definitions for various versions of the ResNet model [11].** *Of particular importance for this thesis work is the layer configuration for the ResNet-50 variant.* . . . 54
4.3 **Work flow for model quantization via the Xilinx DNNDK tool [12].** 56
4.4 **ResNet-50 example testing image where both GPU and FPGA models make the correct inference (lighthouse).** . . . 56
4.5 **ResNet-50 example testing image where the GPU model provides correct inference (snowplow) but the FPGA model is incorrect (forklift).** . . . 57

4.6 **ResNet-50 example testing image where both the GPU model and FPGA provides incorrect inference for the ground truth class of cellphone.** . . . 57
4.7 **CNN architecture design of YOLO version 1 [13].** . . . 60
4.8 **YOLO version 3 example testing image where both GPU and FPGA models make the correct inference (2 instances of class cat).** . . . 62
4.9 **YOLO version 3 example testing image where the GPU model provides correct inference (3 instances of potted plant class) but the FPGA model is incorrect (4 instances of potted plant class).** . . . 62
4.10 **YOLO version 3 example testing image where both the GPU and FPGA models provide incorrect inference for the ground truth class of one instance of class cat.** . . . 63

5.1 **Process of contrast reduction degradation.** *The original image in the top left is first split into its three color channels (red, green, blue). Each of the channels has the dynamic range reduced according to the desired percentage (in this example 50% contrast reduction was selected). Each degraded channel is then combined to create the resulting degraded color image as seen in the bottom left image.* . . . 67
5.2 **Demonstration of SSIM on contrast reduction degradation.** *The images from top left to bottom right are degraded by contrast reduction to provide SSIM scores of 1.0, 0.9, 0.8, ..., 0.1 to provide a visual demonstration of quality degradation.* . . . 68
5.3 **Process of salt-and-pepper degradation.** *The original image appears in the top left. The desired probability for salt-and-pepper is assigned and pixels are set to either white or black accordingly (in this example a noise probability of 0.01 was selected). The randomized noise is then combined with the original image to produce the salt-and-pepper degradation.* . . . 69
5.4 **Demonstration of SSIM on salt-and-pepper degradation.** *The images from top left to bottom right are degraded by salt-and-pepper to provide SSIM scores of 1.0, 0.9, 0.8, ..., 0.1 to provide a visual demonstration of quality degradation.* . . . 70
5.5 **ImageNet - contrast reduction.** *As the degradation parameter increases the SSIM score drops accordingly. With contrast reduction of 50% the average SSIM score for the entire data set quickly drops to 0.72. Between 50% and 10% contrast reduction the rate at which the SSIM score is impacted begins to level off. Beyond a parameter of 2.5% the value of the SSIM score flattens and is not reported here. The response of this experiment demonstrates that the SSIM score can be used to predict quality which may be a predictor for CNN inference accuracy.* . . . 72
5.6 **ImageNet - contrast reduction sample images.** . . . 73

5.7 **ImageNet - salt & pepper.** *As the degradation parameter increases the SSIM score drops accordingly. As compared to the contrast reduction noise, the impact upon the SSIM score is more gradual for salt-and-pepper degradation. At a probability of 0.01875 the average SSIM score for the entire dataset drops to 0.5. From 0.0187 to 0.1 the rate of SSIM reduction slows and beyond a parameter of 0.1 the value of the SSIM score flattens and is not reported here. The response of this experiment demonstrates that the SSIM score can be used to predict quality which may be a predictor for CNN inference accuracy.* . . . 74
5.8 **ImageNet - salt-and-pepper sample images.** 75
5.9 **VOC2007 - contrast reduction.** *As the degradation parameter increases the SSIM score drops accordingly. It can be observed that the VOC2007 dataset's SSIM response to contrast reduction is near identical to that of ImageNet. At contrast reduction of 50% the average SSIM score for the entire dataset is 0.71%. As with the ImageNet contrast reduction experiment, beyond a parameter of 2.5% the value of the SSIM score flattens and is thus not reported here. Given that both datasets respond to contrast reduction in a near identical manner, the evidence to support a potential correspondence between SSIM score for image quality and CNN inference accuracy is increased.* . . . 76
5.10 **VOC2007 - contrast reduction sample images.** 77
5.11 **VOC2007 - salt & pepper.** *As the degradation parameter increases the SSIM score drops accordingly. It can be observed that the VOC2007 dataset's SSIM response to salt-and-pepper degradation is near identical to that of ImageNet. At a probability of 0.01875 the average SSIM score for the entire dataset drops to 0.51. As with the ImageNet salt-and-pepper experiment, beyond a parameter of 0.1 the value of the SSIM score flattens and is thus not reported here. This finding demonstrates that both datasets have near identical SSIM score response to both selected degradation types. This correspondence provides evidence for leveraging the SSIM image quality score to predict CNN inference performance.* 78
5.12 **VOC2007 - salt-and-pepper sample images.** 79
5.13 **Comparison - contrast reduction.** *The ImageNet and VOC2007 datasets have almost identical responses to the degradation. Therefore, the SSIM score for the contrast reduction degradation can be used as a predictor for CNN inference accuracy regardless of dataset.* 80
5.14 **Comparison - salt & pepper.** *This degradation type was the most degrading for both datasets. The ImageNet and VOC2007 datasets have almost identical responses. Therefore, the SSIM score for the salt-and-pepper degradation can be used as a predictor for CNN inference accuracy regardless of dataset.* . . . 80

5.15 **Process of gamma correction applied to contrast reduction degradation.** *The original image in the top left is first split into its three color channels (red, green, blue). Each of the channels has the dynamic range reduced according to the desired percentage (in this example 50% contrast reduction was selected). Each degraded channel is then combined to create the resulting degraded color image as seen in the top right image. To correct the degradation, this image is then split into its three color channel images. Each color channel then is gamma corrected with the selected parameter (in this example an optimal gamma value of 1.2 is used). The corrected channels are then combined to produce the final corrected image as seen in the bottom left. The degraded image's SSIM score was 0.6881 but has a significant raise to 0.9661 after optimal gamma correction is applied.* 82

5.16 **Process of median filter correction applied to salt-and-pepper degradation.** *The original image appears in the top left. The desired probability for salt-and-pepper is assigned and pixels are set to either white or black accordingly (in this example a noise probability of 0.01 was selected). The randomized noise is then combined with the original image to produce the salt-and-pepper degradation. Next, a median filter kernel size is selected to be used for correction (in this example we use a kernel size of 3x3). Finally, the degraded image has the selected median filter applied which produces the final corrected image as seen in the bottom left. The degraded image's SSIM score was 0.6557 but has a significant raise to 0.8388 after optimal median filter correction is applied.* 82

5.17 **ImageNet - gamma correction applied to contrast reduction.** *Each curve represents a single contrast reduction degradation for a specific parameter whose SSIM score is then altered by applying gamma correction. When the gamma parameter is 1.0 this reflects no correction to the degradation. Note that for each curve there is a peak in SSIM score corresponding to an optimal gamma parameter value. For example, the red arrow points to the average SSIM score of 0.3338 for ImageNet when a contrast reduction of 25% is applied. However, the blue arrow points to when an optimal gamma correction of 1.4 is applied providing a corrected average SSIM score of 0.9191 - an increase of 0.5853.* 83

5.18 **ImageNet - gamma correction applied to contrast reduction sample images.** 84

5.19 **ImageNet - median filter correction applied to salt-and-pepper.** *Each bar represents a single salt-and-pepper degradation for a specific parameter whose SSIM score is then altered by applying median filter correction. A median filter kernel size of "None" means no correction has been applied and appears here for reference purposes. Note that although our experiments used various kernel sizes a kernel size of 3x3 always provides the optimal correction. For example, the red arrow points to the average SSIM score of 0.2545 for ImageNet when a salt-and-pepper degradation with probability of 0.05 is applied. However, the blue arrow points to a 3x3 optimal median filter correction which provides a SSIM score of 0.8381 - an increase of 0.5836.* . . 86

5.20 **ImageNet - 3x3 median filter correction applied to salt-and-pepper.** *For each probability of noise across the horizontal axis, we plot the SSIM score with no correction "None" as well as with 3x3 median filter kernel correction to illustrate the very high potential for correction from the salt-and-pepper degradation. For example, the red arrow points to the average SSIM score of 0.2545 for ImageNet when a salt-and-pepper degradation with probability of 0.05 is applied. However, the blue arrow points to a 3x3 optimal median filter correction which provides a SSIM score of 0.8381 - an increase of 0.5836.* . . . 87
5.21 **ImageNet - median filter correction applied to salt-and-pepper sample images.** . . . 88
5.22 **VOC2007 - gamma correction applied to contrast reduction.** *Each curve represents a single contrast reduction degradation for a specific parameter whose SSIM score is then altered by applying gamma correction. When the gamma parameter is 1.0 this reflects no correction to the degradation. Note that for each curve there is a peak in SSIM score corresponding to an optimal gamma parameter value. For example, the red arrow points to the average SSIM score of 0.3266 for VOC2007 when a contrast reduction of 25% is applied. However, the blue arrow points to when an optimal gamma correction of 1.4 is applied providing a corrected average SSIM score of 0.9115 - an increase of 0.5849.* . . . 89
5.23 **VOC2007 - gamma correction applied to contrast reduction sample images.** . . . 90
5.24 **VOC2007 - median filter correction applied to salt-and-pepper.** *Each bar represents a single salt-and-pepper degradation for a specific parameter whose SSIM score is then altered by applying median filter correction. A median filter kernel size of "None" means no correction has been applied and appears here for reference purposes. Note that although our experiments used various kernel sizes a kernel size of 3x3 always provides the optimal correction. For example, the red arrow points to the average SSIM score of 0.2601 for VOC2007 when a salt-and-pepper degradation with probability of 0.05 is applied. However, the blue arrow points to a 3x3 optimal median filter correction which provides a SSIM score of 0.8075 - an increase of 0.5474.* . . . 92
5.25 **VOC2007 - 3x3 median filter correction applied to salt-and-pepper.** *For each probability of noise across the horizontal axis, we plot the SSIM score with no correction "None" as well as with 3x3 median filter kernel correction to illustrate the very high potential for correction from the salt-and-pepper degradation. For example, the red arrow points to the average SSIM score of 0.2601 for VOC2007 when a salt-and-pepper degradation with probability of 0.05 is applied. However, the blue arrow points to a 3x3 optimal median filter correction which provides a SSIM score of 0.8075 - an increase of 0.5474.* . . . 93
5.26 **VOC2007 - median filter correction applied to salt-and-pepper sample images.** . . . 94

6.1 **Results of contrast reduction degradation on ResNet-50 GPU implementation accuracy for the ImageNet testing set (5,000 images).** *With no contrast reduction (a contrast parameter of 100.0%) the top-1 accuracy is observed to be 74.44% and the top-5 accuracy achieves 91.85% which reflects baseline performance for reference. Both the top-1 and top-5 accuracy decrease minimally until contrast reduction is set to 40.0% (accuracy becoming 71.0% and 89.54%, respectively). Between the contrast reduction parameters of 40.0% and 3.3% the top-1 and top-5 accuracy decreases quickly and at similar rates. After a contrast reduction of 2.5% the decrease in accuracy plateaus and is not reported here. Images and video frames degraded beyond 2.5% are shown to be visually improvable with optimal correction but not to a degree such that desirable CNN accuracy can be achieved.* 98

6.2 **Results of contrast reduction degradation on ResNet-50 FPGA implementation accuracy for the ImageNet testing set (5,000 images).** *With no contrast reduction (a contrast parameter of 100.0%) the top-1 accuracy is observed to be 73.34% and the top-5 accuracy achieves 91.31% which reflects baseline performance for reference. As with the GPU model, both the top-1 and top-5 accuracy decrease minimally until contrast reduction is set to 40.0% (accuracy becoming 67.06% and 87.36%, respectively). Between the contrast reduction parameters of 40.0% and 4.4% the top-1 and top-5 accuracy decreases quickly and at similar rates. However, the rate of this decrease in accuracy is sharper for the FPGA model as compared to the GPU model. After a contrast reduction of 2.5% the decrease in accuracy plateaus and is not reported here. Images and video frames degraded beyond 2.5% are shown to be visually improvable with optimal correction but not to a degree such that desirable CNN accuracy can be achieved.* 99

6.3 **Comparison of the results of contrast reduction degradation on top-1 accuracy for ResNet-50 on GPU versus FPGA implementation for the ImageNet testing set (5,000 images).** *With no contrast reduction (a contrast parameter of 100.0%) the top-1 accuracy for the GPU model (blue line) is observed to be 74.44% and the top-1 accuracy for the FPGA model (orange line) achieves 73.34% (a difference of 1.1%) which reflects baseline performance for reference. As previously noted, the decrease in accuracy for both models is minimal up to a contrast parameter of 40.0%. Between the parameters of 40.0% and 3.3% we see a sharper rate of decrease in inference accuracy for both hardware implementations, however the FPGA model is shown to be more impacted by the contrast reduction degradation. This higher rate of accuracy degradation for the FPGA implementation is due to the less precision of the 8-bit integer weights it leverages as compared to the 32-bit floating point weights of the GPU model. However, this increase in weight precision and thus accuracy comes at the cost of much greater SWaP requirements for deployment. For both hardware models, beyond a contrast reduction parameter of 2.5% the accuracy performance plateaus and is not reported here.* 101

6.4 **Comparison of the results of contrast reduction degradation on top-5 accuracy for ResNet-50 on GPU versus FPGA implementation for the ImageNet testing set (5,000 images).** *With no contrast reduction (a contrast parameter of 100.0%) the top-5 accuracy for the GPU model (blue line) is observed to be 91.85% and the top-5 accuracy for the FPGA model (orange line) achieves 91.31% (a difference of 0.54%) which reflects baseline performance for reference. As previously noted, the decrease in accuracy for both models is minimal up to a contrast parameter of 40.0%. Between the parameters of 40.0% and 3.3% we see a sharper rate of decrease in inference accuracy for both hardware implementations, however the FPGA model is shown to be more impacted by the contrast reduction degradation. This higher rate of accuracy degradation for the FPGA implementation is due to the less precision of the 8-bit integer weights it leverages as compared to the 32-bit floating point weights of the GPU model. However, this increase in weight precision and thus accuracy comes at the cost of much greater SWaP requirements for deployment. For both hardware models, beyond a contrast reduction parameter of 2.5% the accuracy performance plateaus and is not reported here.* 102

6.5 **Side-by-side comparison of the results of contrast reduction degradation on both top-1 and top-5 accuracy for ResNet-50 on GPU versus FPGA implementation.** *The set of blue lines (solid for GPU and dashed for FPGA) plot the top-5 accuracies when contrast reduction is applied. The set of red lines (solid for GPU and dashed for FPGA) plot the top-1 accuracies when contrast reduction is applied. With no contrast reduction (a contrast parameter of 100.0%) the top-1 and top-5 accuracies for both the FPGA and GPU models have an approximate 1% difference and are shown here for a baseline reference. For both cases, the FPGA model accuracy is shown to be more impacted by contrast reduction between the 40.0% and 3.3% parameters as compared to the GPU model. However, this degradation can be corrected with optimally applied methods.* 103

6.6 **ResNet-50 example testing image with contrast reduction degradation at 20%.** *The original image from the ImageNet testing set appears in the top. Both GPU and FPGA models make the correct inference (lens cap). For this particular instance the degradation level was not severe enough to cause a mislabeling for either hardware model.* 104

6.7 **ResNet-50 example testing image with contrast reduction degradation at 20%.** *The original image from the ImageNet testing set appears in the top. The GPU model provides correct inference (bee) but the FPGA model is incorrect (tarantula). In this instance the higher precision of the GPU model weights appear to be more robust to this degree of contrast reduction.* 105

6.8 **ResNet-50 example testing image with contrast reduction degradation at 20%.** *The original image from the ImageNet testing set appears in the top. Both the GPU and FPGA models provide incorrect inference for the class lighthouse. Both models provide the correct level before degradation, however, at 20% contrast reduction neither model can successfully classify the image.* . . . 106

6.9 **Results of salt-and-pepper degradation on ResNet-50 GPU implementation accuracy for the ImageNet testing set (5,000 images).** *With no salt-and-pepper degradation (a parameter of 0) the top-1 accuracy is observed to be 74.44% and the top-5 accuracy achieves 91.85% which reflects baseline performance for reference. The impact of salt-and-pepper degradation upon accuracy is significant at even small parameters. With a probability of noise parameter of 0.005 the top-1 accuracy goes from 74.44% to 62.84% (a drop of 11.6%) and the top-5 accuracy goes from 91.85% to 83.58% (a drop of 8.27%). For our experiments we cut off the probability of noise parameter to 0.1, where the GPU top-1 accuracy achieves 19.9% and the top-5 accuracy achieves 40.48%.* . . . 108

6.10 **Results of salt-and-pepper degradation on ResNet-50 FPGA implementation accuracy for the ImageNet testing set (5,000 images).** *With no salt-and-pepper degradation (a parameter of 0) the top-1 accuracy is observed to be 73.34% and the top-5 accuracy achieves 91.31% which reflects baseline performance for reference. The impact of salt-and-pepper degradation upon accuracy is significant at even small parameters. With a probability of noise parameter of 0.005 the top-1 accuracy goes from 73.34% to 62.7% (a drop of 10.64%) and the top-5 accuracy goes from 91.31% to 83.5% (a drop of 7.81%). For our experiments we cut off the probability of noise parameter to 0.1, where the FPGA top-1 accuracy achieves 19.14% and the top-5 accuracy achieves 39.42%.* . . . 109

6.11 **Comparison of the results of salt-and-pepper degradation on top-1 accuracy for ResNet-50 on GPU versus FPGA implementation for the ImageNet testing set (5,000 images).** *With no salt-and-pepper degradation (a probability of noise parameter of 0) the top-1 accuracy for the GPU model (blue line) is observed to be 74.44% and the top-1 accuracy for the FPGA model (orange line) achieves 73.34% (a difference of 1.1%) which reflects baseline performance for reference. With a probability of noise parameter of 0.005 the top-1 accuracy for GPU 74.44% to 62.84% (a drop of 11.6%) and the top-1 accuracy for FPGA goes from 73.34% to 62.7% (a drop of 10.64%) showing that the impact of salt-and-pepper degradation upon accuracy is significant at even small parameters. Unlike with the contrast reduction degradation, the FPGA implementation appears to have a similar degree of robustness to salt-and-pepper degradation as compared to the GPU implementation. The rate of accuracy decrease is almost identical between the two implementations with the FPGA accuracy maintaining an approximate 1% drop at all salt-and-pepper parameters.* . . . 111

6.12 **Comparison of the results of salt-and-pepper degradation on top-5 accuracy for ResNet-50 on GPU versus FPGA implementation for the ImageNet testing set (5,000 images).** *With no salt-and-pepper degradation (a probability of noise parameter of 0) the top-5 accuracy for the GPU model (blue line) is observed to be 91.85% and the top-5 accuracy for the FPGA model (orange line) achieves 91.31% (a difference of 0.54%) which reflects baseline performance for reference. With a probability of noise parameter of 0.005 the top-5 accuracy for GPU 91.85% to 83.58% (a drop of 8.27%) and the top-5 accuracy for FPGA goes from 91.31% to 83.5% (a drop of 7.81%) showing that the impact of salt-and-pepper degradation upon accuracy is significant at even small parameters. Unlike with the contrast reduction degradation, the FPGA implementation appears to have a similar degree of robustness to salt-and-pepper degradation as compared to the GPU implementation. The rate of accuracy decrease is almost identical between the two implementations with the FPGA accuracy maintaining an approximate 1% drop at all salt-and-pepper parameters.* 112

6.13 **Side-by-side comparison of the results of salt-and-pepper degradation on both top-1 and top-5 accuracy for ResNet-50 on GPU versus FPGA implementation.** *The set of blue lines (solid for GPU and dashed for FPGA) plot the top-5 accuracies when salt-and-pepper degradation is applied. The set of red lines (solid for GPU and dashed for FPGA) plot the top-1 accuracies when salt-and-pepper degradation is applied. With no salt-and-pepper degradation (a probability of noise parameter of 0) the top-1 and top-5 accuracies for both the FPGA and GPU models have an approximate 1% difference and are shown here for a baseline reference. The rate of accuracy decrease is almost identical between the two implementations with the FPGA accuracy maintaining an approximate 1% drop at all salt-and-pepper parameters.* 113

6.14 **ResNet-50 example testing image with salt-and-pepper degradation at 0.05.** *The original image from the ImageNet testing set appears in the top. Both GPU and FPGA models make the correct inference (bee). For this particular instance the degradation level was not severe enough to cause a mislabeling for either hardware model.* 114

6.15 **ResNet-50 example testing image with salt-and-pepper degradation at 0.05.** *The original image from the ImageNet testing set appears in the top. The GPU model provides correct inference (tent) but the FPGA model is incorrect (pinwheel). In this instance the higher precision of the GPU model weights appear to be more robust to this degree of salt-and-pepper degradation.* 115

6.16 **ResNet-50 example testing image with salt-and-pepper degradation at 0.05.** *The original image from the ImageNet testing set appears in the top. Both the GPU model and FPGA provides incorrect inference for the class snowplow. Both models provide the correct level before degradation, however, at salt-and-pepper degradation with a noise parameter of 0.05 neither model can successfully classify the image.* 116

6.17 **The results of gamma correction at various parameters when applied to contrast reduction based degradation for the GPU implementation of ResNet-50 for the ImageNet testing set (5,000 images).** *Each line in the graph represents the response of a particular contrast reduction factor across the range of gamma correction parameters of 1 to 2. For example, with a contrast reduction factor of 20.0% (the top grey line) and a gamma correction factor of 1 (no correction) the top-1 accuracy is 61.48% (see red arrow). The optimal gamma factor for this contrast reduction parameter is observed at 1.5 returning the top-1 accuracy to 70.09% (see blue arrow) - an increase of 8.61%. Therefore, we observe that contrast reduced images and videos can have their quality significantly improved with optimal correction methods which also increases ResNet-50 inference accuracy. Given that images and videos with reduced contrast are common for in-the-wild imaging, this significant increase in accuracy is highly advantageous for Edge systems.* 118

6.18 **The results of gamma correction at various parameters when applied to contrast reduction based degradation for the FPGA implementation of ResNet-50. for the ImageNet testing set (5,000 images)** *Each line in the graph represents the response of a particular contrast reduction factor across the range of gamma correction parameters of 1 to 2. For example, with a contrast reduction parameter of 20.0% (the top blue line) and a gamma correction factor of 1 (no correction) the top-1 accuracy is 53.42% (see red arrow). The optimal gamma factor for this contrast reduction parameter is observed at 1.5 returning the top-1 accuracy to 66.25% (see blue arrow) - an increase of 12.83%. As with the GPU model, we observe that contrast reduced images and videos can have their quality significantly improved with optimal correction methods which also increases the FPGA ResNet-50 model inference accuracy. Given that images and videos with reduced contrast are common for in-the-wild imaging, this significant increase in accuracy is highly advantageous for Edge systems.* 119

6.19 **Side-by-side comparison of the results of gamma correction on contrast reduction based degradation for ResNet-50 on GPU versus FPGA implementations.** *Three illustrative contrast reduction parameters were selected to compare implementation performance: 20.0%, 6.7%, and 3.3% (solid lines for GPU and dashed lines for FPGA). For example, when the contrast reduction parameter is 20.0% and no correction is applied (gamma parameter is 1) the GPU top-1 accuracy (blue line) is 61.48% whereas the FPGA top-1 accuracy (orange line) is 53.42% (see red arrow). With an optimal gamma correction of 1.5 the GPU implementation at contrast reduction parameter of 20.0% raises to 70.08% (see blue arrow) - an 8.6% increase. Similarly, with an optimal gamma correction of 1.5 the FPGA implementation at a contrast reduction parameter of 20.0% raises to 66.25% (see blue arrow) - a 12.83% increase. For this example, after optimal correction the FPGA accuracy returns to within 3.83% of the GPU accuracy. Therefore, even though the contrast reduction degradation impacts the FPGA model more severely, the gamma correction method improves the FPGA model at a higher rate than the GPU model. After optimal correction, the FPGA model shows similar robustness to contrast reduction as compared to the GPU model. This similar robustness coupled with the FPGA's advantage of far superior SWaP characteristics make it the superior implementation for deployment in Edge applications.* 121

6.20 **ResNet-50 example testing image with contrast reduction degradation (20.0%) and gamma correction (1.5).** *The original image from the ImageNet testing set appears in the top left. The top right image displays this image after contrast reduction is applied with a parameter of 20.0%. Both the GPU and FPGA failed to properly classify this image at contrast reduction degradation at 20.0% but both provide the correct inference class label (lighthouse) when an optimal gamma correction of 1.5 is applied (GPU result in bottom left and FPGA result in bottom right).* 122

6.21 **The results of median filter correction at various kernel sizes when applied to salt-and-pepper based degradation for the GPU implementation of ResNet-50 for the ImageNet testing set (5,000 images).** *Each line in the graph represents the response of a particular salt-and-pepper degradation probability of noise parameter when corrected by several median filters of various kernel sizes. The kernel size of "None" is used to indicate no correction is applied. The top line reflects the case where no salt-and-pepper degradation is introduced (probability of noise parameter of 0.0) and is used to illustrate that the median filtering correction can have a negative impact upon image quality by causing a blurring effect if the kernel size is set too large. For example, the bottom blue line represents the most severe salt-and-pepper degradation with the probability parameter of 0.1. With no correction it has a top-1 accuracy of 19.9% (see red arrow). However, with an optimal correction kernel size of 5x5 the top-1 accuracy is increased to 63.74% (see blue arrow), which represents a significant increase of 43.84%. Of additional note is that simply applying a median filter kernel of size 3x3 can greatly increase accuracy regardless of the salt-and-pepper degradation parameter. Therefore, we observe that salt-and-pepper degraded images and videos can have their quality significantly improved when optimal correction is applied which in turn increases ResNet-50 inference accuracy. Given that images and videos with salt-and-pepper noise are common for in-the-wild imaging, this significant increase in accuracy is highly advantageous for Edge systems.* 123

6.22 **The results of median filter correction at various kernel sizes when applied to salt-and-pepper based degradation for the FPGA implementation of ResNet-50 for the ImageNet testing set (5,000 images).** *Each line in the graph represents the response of a particular salt-and-pepper degradation probability of noise parameter when corrected by several median filters of various kernel sizes. The kernel size of "None" is used to indicate no correction is applied. The top line reflects the case where no salt-and-pepper degradation is introduced (probability of noise parameter of 0.0) and is used to illustrate that the median filtering correction can have a negative impact upon image quality by causing a blurring effect if the kernel size is set too large. For example, the bottom blue line represents the most severe salt-and-pepper degradation with the probability parameter of 0.1. With no correction it has a top-1 accuracy of 19.14% (see red arrow). However, with an optimal correction kernel size of 5x5 the top-1 accuracy is increased to 63.29% (see blue arrow), which represents a significant increase of 44.15%. Of additional note is that simply applying a median filter kernel of size 3x3 can greatly increase accuracy regardless of the salt-and-pepper degradation parameter. As with the GPU model, we observe that salt-and-pepper degraded images and videos can have their quality significantly improved when optimal correction is applied which in turn increases the FPGA ResNet-50 inference accuracy. Given that images and videos with salt-and-pepper noise are common for in-the-wild imaging, this significant increase in accuracy is highly advantageous for Edge systems.* 125

6.23 **Side-by-side comparison of the results of median filter correction on salt-and-pepper based degradation for ResNet-50 on GPU versus FPGA implementations.** *Three illustrative salt-and-pepper probability of noise parameters were selected to compare implementation performance: 0.005, 0.05, and 0.1 (solid lines for GPU and dashed lines for FPGA). For example, when the probability of noise parameter is 0.05 and no correction is applied (no median filter applied, i.e. "None") the GPU top-1 accuracy is 39.06% whereas the FPGA top-1 accuracy of 37.8% (see red arrow). With an optimal median filter size of 3x3 the GPU implementation at salt-and-pepper probability of 0.05 raises to 72.74% (see blue arrow) - a 33.68% increase. Similarly, with an optimal median filter size of 3x3 the FPGA implementation at salt-and-pepper probability of 0.05 raises to 71.86% (see blue arrow) - a 34.06% increase. For this example, after optimal correction the FPGA returns to within 0.88% of the GPU accuracy. Unlike with the gamma corrected contrast reduced experiments, the FPGA has near identical robustness as the GPU to salt-and-pepper degradation when optimal median filter correction is applied. This near identical robustness coupled with the FPGA's advantage of far superior SWaP characteristics make it the superior implementation for deployment in Edge applications.* 126

6.24 **ResNet-50 example testing image with salt-and-pepper degradation (0.05) and median filter correction (3x3).** *The original image from the ImageNet testing set appears in the top left. The top right image displays this image after salt-and-pepper degradation is applied with a parameter of 0.05. Both the GPU and FPGA failed at salt-and-pepper degradation at 0.05 but both provide the correct inference class label (snowplow) when an optimal median filter correction of 3x3 is applied (GPU result in bottom left and FPGA result in bottom right.* 127

6.25 **Results of contrast reduction degradation on YOLO version 3 GPU implementation average precision as evaluated on the VOC2007 testing set (4,952 images).** *With no contrast reduction (a contrast parameter of 100.0%) the mAP for all classes (bold green line) is observed to be 78.46%. For no contrast reduction the best performing class ("car" - bold blue line) has an average precision of 86.45% and the worst performing class ("potted plant" - bold red line) has an average precision of 51.68%. After a contrast reduction of 2.5% the decrease in mAP plateaus and is not reported here. Images and video frames degraded beyond 2.5% are shown to be visually improvable with optimal correction but not to a degree such that desirable CNN mAP can be achieved.* 132

6.26 **Results of contrast reduction degradation on YOLO version 3 FPGA implementation average precision as evaluated on the VOC2007 testing set (4,952 images).** *With no contrast reduction (a contrast parameter of 100.0%) the mAP for all classes (bold green line) is observed to be 77.44%. For no contrast reduction the best performing class ("car" - bold blue line) has an average precision of 85.47% and the worst performing class ("potted plant" - bold red line) has an average precision of 50.46%. After a contrast reduction of 2.5% the decrease in mAP plateaus and is not reported here. Images and video frames degraded beyond 2.5% are shown to be visually improvable with optimal correction but not to a degree such that desirable CNN mAP can be achieved.* 133

6.27 **Comparison of the results of contrast reduction degradation on average precision for YOLO version 3 on GPU versus FPGA implementation for the VOC2007 testing set (4,952 images).** *The GPU implementation has a mAP of 78.46% as compared to the FPGA implementation of 77.44% (a difference of 1.02%) when no contrast reduction degradation is present (contrast parameter of 100.0%). The FPGA implementation average precision decreases at a quicker rate than the GPU implementation indicating a greater impact of degradation upon model inference. The most likely reason for this difference in degradation impact is the nature of the two model weights. Although the 32-bit floating point weights in the GPU model require higher SWaP characteristics to calculate they are more robust to the contrast reduction degradation than the 8-bit integer quantized weights of the FPGA model.* 135

6.28 **YOLO version 3 example testing image with contrast reduction degradation at 20.0%.** *The original image from the VOC2007 testing set appears in the top. Both GPU and FPGA models make the correct inference (two instances of class potted plant). For this particular instance the degradation level was not severe enough to cause a mislabeling for either hardware model.* 136
6.29 **YOLO version 3 example testing image with contrast reduction degradation at 20.0%.** *The original image from the VOC2007 testing set appears in the top. The GPU model provides correct inference (single instance of potted plant) but the FPGA model is incorrect (no detections). In this instance the higher precision of the GPU model weights appear to be more robust to this degree of contrast reduction.* 137
6.30 **YOLO version 3 example testing image with contrast reduction degradation at 20.0%.** *The original image from the VOC2007 testing set appears in the top. Both the GPU model and FPGA provides incorrect inference (ground truth is two instances of class horse). Both models provide the correct level before degradation, however, at 20% contrast reduction neither model can successfully classify the image.* 138
6.31 **Results of salt-and-pepper degradation on YOLO version 3 GPU implementation average precision as evaluated on the VOC2007 testing set (4,952 images).** *With no noise (a parameter of 0) the mAP for all classes (bold green line) is observed to be 78.46%. For no salt-and-pepper noise the best performing class ("bicycle" - bold blue line) has an average precision of 86.47% and the worst performing class ("potted plant" - bold red line) has an average precision of 51.68%. For our experiments we cut off the probability of noise parameter to 0.1.* 139
6.32 **Results of salt-and-pepper degradation on YOLO version 3 FPGA implementation average precision as evaluated on the VOC2007 testing set (4,952 images).** *With no noise (a parameter of 0) the mAP for all classes (bold green line) is observed to be 77.44%. For no salt-and-pepper noise the best performing class ("bicycle" - bold blue line) has an average precision of 86.92% and the worst performing class ("potted plant" - bold red line) has an average precision of 50.46%. As expected, the greater the salt-and-pepper parameter the lower the calculated average precision. For our experiments we cut off the probability of noise parameter to 0.1.* 141

6.33 **Comparison of the results of salt-and-pepper degradation on mean average precision (mAP) for YOLO version 3 on GPU versus FPGA implementation for the VOC2007 testing set (4,952 images).** *GPU results appear in solid lines and FPGA results in dashed lines. We report the mAP for all classes (green lines), the best class ("bicycle" - blue lines), and the worst class ("potted plant" - red lines) in this figure. The GPU implementation has a mAP of 78.46% as compared to the FPGA implementation of 77.44% (a difference of 1.02%) when no salt-and-pepper degradation is present (probability of noise is 0). As was the case for the ResNet-50 architecture, the FPGA implementation appears to have a similar degree of robustness to this degradation type as compared to the GPU implementation. The most likely explanation for this is that since salt-and-pepper degradation is a discreet operation instead of continuous (as is contrast reduction) the integer based weights of the FPGA model react nearly identically to the floating point based weights of the GPU model.* 142

6.34 **YOLO version 3 example testing image with salt-and-pepper degradation at 0.05.** *The original image from the VOC2007 testing set appears in the top. Both GPU and FPGA models make the correct inference (two instances of class bicycle). For this particular instance the degradation level was not severe enough to cause a mislabeling for either hardware model.* . . . 143

6.35 **YOLO version 3 example testing image with salt-and-pepper degradation at 0.05.** *The original image from the VOC2007 testing set appears in the top. The GPU model provides correct inference (single instance of class bicycle) but the FPGA model is incorrect (no instance of class bicycle). In this instance the higher precision of the GPU model weights appear to be more robust to this degree of salt-and-pepper degradation.* 144

6.36 **YOLO version 3 example testing image with salt-and-pepper degradation at 0.05.** *The original image from the VOC2007 testing set appears in the top. Both the GPU model and FPGA provide incorrect inference for the ground truth of 9 instances of class aeroplane. Both models provide the correct level before degradation, however, at salt-and-pepper degradation with a noise parameter of 0.05 neither model can successfully classify the image.* . 145

6.37 **The results of gamma correction at various parameters when applied to contrast reduction based degradation for the GPU implementation of YOLO version 3 for the VOC2007 testing set (4,952 images).** *Each line in the graph represents the response of a particular contrast reduction factor across the range of gamma correction parameters of 1 to 2. For example, with a contrast reduction parameter of 13.3% (top orange line) and a gamma correction factor of 1 (no correction) the mAP is 70.94% (see red arrow). The optimal gamma factor for this contrast reduction parameter is observed at 1.5 returning the mAP to 75.90% (see blue arrow) - an increase of 4.96%. The other corrections in this graph show similar response with some achieving significant improvements in mAP. Therefore, we observe that contrast reduced images and videos can have their quality significantly improved with optimal correction methods which also increases YOLO version 3 mAP. Given that images and videos with reduced contrast are common for in-the-wild imaging, this significant increase in mAP is highly advantageous for Edge systems.* 147

6.38 **The results of gamma correction at various parameters when applied to contrast reduction based degradation for the FPGA implementation of YOLO version 3 for the VOC2007 testing set (4,952 images).** *Each line in the graph represents the response of a particular contrast reduction factor across the range of gamma correction parameters of 1 to 2. For example, with a contrast reduction parameter of 13.3% (top orange line) and a gamma correction factor of 1 (no correction) the mAP is 65.34% (see red arrow). The optimal gamma factor for this contrast reduction parameter is observed at 1.5 returning the mAP to 74.92% (see blue arrow) - an increase of 9.58%. The other corrections in this graph show similar response with some achieving significant improvements in mAP. As with the GPU model, we observe that contrast reduced images and videos can have their quality significantly improved with optimal correction methods which also increases the FPGA YOLO version 3 model mAP. Given that images and videos with reduced contrast are common for in-the-wild imaging, this significant increase in accuracy is highly advantageous for Edge systems.* . . . 148

6.39 **Side-by-side comparison of the results of gamma correction on contrast reduction based degradation for YOLO version 3 on GPU versus FPGA implementations.** *Three illustrative low contrast parameters were selected to compare implementation performance: 13.3%, 4.0%, and 2.5% (solid lines for GPU and dashed lines for FPGA). For example, when the contrast reduction parameter is 13.3% and no correction is applied (gamma parameter is 1) the GPU mAP is 70.94% whereas the FPGA mAP is 65.34% (see red arrow). With an optimal gamma correction of 1.5 the GPU implementation at contrast reduction parameter of 13.3% raises to 75.90% (see blue arrow) - a 5.04% increase. Similarly, with an optimal gamma correction of 1.5 the FPGA implementation at a contrast reduction parameter of 13.3% raises to 74.92% (see blue arrow) - an increase of 9.58%. For this example, after optimal correction the FPGA mAP returns to within 0.98% of the GPU mAP. Therefore, even though the contrast reduction degradation impacts the FPGA model more severely, the gamma correction method improves the FPGA model at a higher rate than the GPU model. After optimal correction, the FPGA model shows similar robustness to contrast reduction as compared to the GPU model. This similar robustness coupled with the FPGA's advantage of far superior SWaP characteristics make it the superior implementation for deployment in Edge applications.* 149

6.40 **YOLO version 3 example testing image with contrast reduction degradation (20.0%) and gamma correction (1.5).** *The original image from the ImageNet testing set appears in the top left. The top right image displays this image after contrast reduction is applied with a parameter of 20.0%. Both the GPU and FPGA failed to properly classify this image at contrast reduction degradation at 20.0% but both provide the correct inference class label (two instance of class potted plant) when an optimal gamma correction of 1.5 is applied (GPU result in bottom left and FPGA result in bottom right).* 151

6.41 **The results of median filter correction at various kernel sizes when applied to salt-and-pepper based degradation for the GPU implementation of YOLO version 3 for the VOC2007 testing set (4,952 images).** *Each line in the graph represents the response of a particular salt-and-pepper degradation probability of noise parameter when corrected by several median filters of various kernel sizes. The kernel size of "None" is used to indicate no correction is applied. The top line reflects the case where no salt-and-pepper degradation is introduced (probability of noise parameter of 0.0) and is used to illustrate that the median filtering correction can have a negative impact upon image quality by causing a blurring effect if the kernel size is set too large. For example, the bottom blue line represents the most severe salt-and-pepper degradation with the probability parameter of 0.1. With no correction it has a mAP of 29.9% (see red arrow). However, with an optimal correction kernel size of 5x5 the mAP is increased to 66.79% (see blue arrow), which represents a significant increase of 36.89%. Of additional note is that simply applying a median filter kernel of size 3x3 can greatly increase accuracy regardless of the salt-and-pepper degradation parameter. Therefore, we observe that salt-and-pepper degraded images and videos can have their quality significantly improved when optimal correction is applied which in turn increases YOLO version 3 mAP. Given that images and videos with salt-and-pepper noise are common for in-the-wild imaging, this significant increase in accuracy is highly advantageous for Edge systems.* 152

6.42 **The results of median filter correction at various kernel sizes when applied to salt-and-pepper based degradation for the FPGA implementation of YOLO version 3 for the VOC2007 testing set (4,952 images).** *Each line in the graph represents the response of a particular salt-and-pepper degradation probability of noise parameter when corrected by several median filters of various kernel sizes. The kernel size of "None" is used to indicate no correction is applied. The top line reflects the case where no salt-and-pepper degradation is introduced (probability of noise parameter of 0.0) and is used to illustrate that the median filtering correction can have a negative impact upon image quality by causing a blurring effect if the kernel size is set too large. For example, the bottom blue line represents the most severe salt-and-pepper degradation with the probability parameter of 0.1. With no correction it has a mAP of 27.60% (see red arrow). However, with an optimal correction kernel size of 5x5 the mAP is increased to 67.56% (see blue arrow), which represents a significant increase of 39.96%. Of additional note is that simply applying a median filter kernel of size 3x3 can greatly increase accuracy regardless of the salt-and-pepper degradation parameter. As with the GPU model, we observe that salt-and-pepper degraded images and videos can have their quality significantly improved when optimal correction is applied which in turn increases the FPGA YOLO version 3 mAP. Given that images and videos with salt-and-pepper noise are common for in-the-wild imaging, this significant increase in accuracy is highly advantageous for Edge systems.* 154

6.43 **Side-by-side comparison of the results of median filter correction on salt-and-pepper based degradation for YOLO version 3 on GPU versus FPGA implementations.** *Three illustrative salt-and-pepper probability of noise parameters were selected to compare implementation performance: 0.005, 0.05, and 0.1 (solid lines for GPU and dashed lines for FPGA). For example, when the probability of noise parameter is 0.05 and no correction is applied (no median filter applied, i.e. "None") the GPU mAP is 40.69% whereas the FPGA mAP of 42.37% (see red arrow). With an optimal median filter size of 3x3 the GPU implementation at salt-and-pepper probability of 0.05 raises to 74.52% (see blue arrow) - a 33.83% increase. Similarly, with an optimal median filter size of 3x3 the FPGA implementation at salt-and-pepper probability of 0.05 raises to 74.13% (see blue arrow) - a 31.76% increase. For this example, after optimal correction the FPGA returns to within 0.39% of the GPU mAP. Unlike with the gamma corrected contrast reduced experiments, the FPGA has near identical robustness as the GPU to salt-and-pepper degradation when optimal median filter correction is applied. This near identical robustness coupled with the FPGA's advantage of far superior SWaP characteristics make it the superior implementation for deployment in Edge applications.* 155

6.44 **YOLO version 3 example testing image with salt-and-pepper degradation (0.05) and median filter correction (3x3).** *he original image from the ImageNet testing set appears in the top left. The top right image displays this image after salt-and-pepper degradation is applied with a parameter of 0.05. Both the GPU and FPGA failed at salt-and-pepper degradation at 0.05 but both provide the correct inference class label (two instances of class bicycle) when an optimal median filter correction of 3x3 is applied (GPU result in bottom left and FPGA result in bottom right.)* 156

1 **Horizontal and vertical Sobel kernels.** 178

2 **Sobel operation IP core flow chart.** *The HDMI source signal is captured via the AXI interconnect and stored as an input frame within the Zynq PL. HLS code is then used to calculate the horizontal and vertical Sobel filters, combine them and cast the result into an output frame before sending it out of the PL via AXI interconnect and then to HDMI sink.* 178

3 **Sobel IP core block diagram.** 178

4 **Sobel IP timing/latency.** 178

5 **Sobel IP hardware utilization.** 178

6 **Sobel Edge Detector - grayscale image results (image source [10]).** 179

7 **Sobel Edge Detector - color image results (image source [10]).** . . . 179

8 **Threshold operation IP core flow chart.** *The HDMI source signal is captured via the AXI interconnect and stored as an input frame within the Zynq PL. HLS code is then used to convert the R,G and B channels into grayscale equivalent. Each pixel is then evaluated, if the value is below 127 it is now mapped to 0 and if it is greater than or equal to 127 then it is mapped to 255. This value is then duplicated back to a 3-channel grayscale image and placed into the output frame before sending it out of the PL via AXI interconnect and then to HDMI sink.* . . . 180
9 **Threshold IP core block diagram.** . . . 180
10 **Threshold IP timing/latency.** . . . 180
11 **Threshold IP hardware utilization.** . . . 180
12 **Threshold (threshold parameter = 127) - color image results.** . . . 182
13 **Sobel-Threshold operation IP core flow chart.** *The HDMI source signal is captured via the AXI interconnect and stored as an input frame within the Zynq PL. HLS code is then used to calculate the Sobel derivative of the frame. Additional HLS code is then used to convert the R,G and B channels into grayscale equivalent. Each pixel is then evaluated, if the value is below 127 it is now mapped to 0 and if it is greater than or equal to 127 then it is mapped to 255. This value is then duplicated back to a 3-channel grayscale image and placed into the output frame before sending it out of the PL via AXI interconnect and then to HDMI sink.* . . . 182
14 **Sobel-Threshold IP core block diagram.** . . . 182
15 **Sobel-Threshold IP timing/latency.** . . . 183
16 **Sobel-Threshold IP hardware utilization.** . . . 183
17 **Sobel-Threshold - grayscale image results (image source [10]).** . . . 183
18 **Sobel-Threshold - color image results (image source [10]).** . . . 184
19 **Dilation structuring element.** . . . 184
20 **Dilation operation IP core flow chart.** *The HDMI source signal is captured via the AXI interconnect and stored as an input frame within the Zynq PL. HLS code is then used to apply the structuring element to the pixels in the frame performing the dilation morphological operation and placed into the output frame before sending it out of the PL via AXI interconnect and then to HDMI sink.* . . . 184
21 **Dilation IP core block diagram.** . . . 184
22 **Dilation IP timing/latency.** . . . 185
23 **Dilation IP hardware utilization.** . . . 185
24 **Dilation - grayscale image results.** . . . 185
25 **Dilation - color image results (image source [10]).** . . . 187
26 **Erosion structuring element.** . . . 187
27 **Erosion operation IP core flow chart.** *The HDMI source signal is captured via the AXI interconnect and stored as an input frame within the Zynq PL. HLS code is then used to apply the structuring element to the pixels in the frame performing the erosion morphological operation and placed into the output frame before sending it out of the PL via AXI interconnect and then to HDMI sink.* . . . 187

28 **Erosion IP core block diagram.** . . . 187
29 **Erosion IP timing/latency.** . . . 188
30 **Erosion IP hardware utilization.** . . . 188
31 **Erosion - grayscale image results.** . . . 188
32 **Erosion - color image results (image source [10]).** . . . 189
33 **Inversion operation IP core flow chart.** *The HDMI source signal is captured via the AXI interconnect and stored as an input frame within the Zynq PL. HLS code is then used to invert the pixel values in the frame and then placed into the output frame before sending it out of the PL via AXI interconnect and then to HDMI sink.* . . . 189
34 **Inversion IP core block diagram.** . . . 189
35 **Inversion IP timing/latency.** . . . 190
36 **Inversion IP hardware utilization.** . . . 190
37 **Inversion - grayscale image results (image source [10]).** . . . 190
38 **Inversion - color image results (image source [10]).** . . . 190
39 **Gaussian blur kernel.** . . . 192
40 **Gaussian Blur operation IP core flow chart.** *The HDMI source signal is captured via the AXI interconnect and stored as an input frame within the Zynq PL. HLS code is then used to blur the pixel values in the frame via the 5x5 Gaussian kernel and then placed into the output frame before sending it out of the PL via AXI interconnect and then to HDMI sink.* . . . 192
41 **Gaussian Blur IP core block diagram.** . . . 192
42 **Gaussian Blur IP timing/latency.** . . . 192
43 **Gaussian Blur IP hardware utilization.** . . . 192
44 **Gaussian Blur - grayscale image results (image source [10]).** . . . 193
45 **Gaussian Blur - color image results (image source [10]).** . . . 193
46 **Sharpen kernel.** . . . 194
47 **Sharpen operation IP core flow chart.** *The HDMI source signal is captured via the AXI interconnect and stored as an input frame within the Zynq PL. HLS code is then used to sharpen the pixel values in the frame via a 3x3 kernel and then placed into the output frame before sending it out of the PL via AXI interconnect and then to HDMI sink.* . . . 194
48 **Sharpen IP core block diagram.** . . . 194
49 **Sharpen IP timing/latency.** . . . 194
50 **Sharpen IP hardware utilization.** . . . 194
51 **Sharpen - Grayscale image results (image source [10]).** . . . 195
52 **Sharpen - color image results (image source [10]).** . . . 195
53 **Contrast Reduction (contrast reduced image created by pixel operation of division) IP core flow chart.** *The HDMI source signal is captured via the AXI interconnect and stored as an input frame within the Zynq PL. HLS code is then used to reduce the image contrast by dividing each R, G and B value of the pixel by three and then placed into the output frame before sending it out of the PL via AXI interconnect and then to HDMI sink.* 195
54 **Contrast Reduction (pixel operation) IP core block diagram.** . . . 196
55 **Contrast Reduction (pixel operation) IP timing/latency.** . . . 196

56 **Contrast Reduction (pixel operation) IP hardware utilization.** . . . 196
57 **Contrast Reduction (pixel operation) - color image results.** 196
58 **Edge detection kernel.** 198
59 **Convolution operation IP core flow chart.** *The HDMI source signal is captured via the AXI interconnect and stored as an input frame within the Zynq PL. HLS code is then used to detect edges in the frame via a 3x3 kernel and then placed into the output frame before sending it out of the PL via AXI interconnect and then to HDMI sink.* 198
60 **Convolution IP core block diagram.** 198
61 **Convolution IP timing/latency.** 198
62 **Convolution IP hardware utilization.** 198
63 **Convolution - grayscale image results (image source [10]).** 199
64 **Convolution - color image results (image source [10]).** 199

Nomenclature

ASIC Application Specific Integrated Circuit

AXI Advanced eXtensible Interface

CNN Convolutional Neural Network

DPU Deep-learning Processing Unit

FPGA Field Programmable Gate Array

GPU Graphics Processing Unit

HDL Hardware Description Language

HDMI High-Definition Multimedia Interface

HLS High Level Synthesis

HPC High-Performance Computing

IP Intellectual Property

IRIS Imaging, Robotics, and Intelligent Systems

mAP mean Average Precision

PSNR Peak Signal to Noise Ratio

ROC Receiver-Operator Curve

SSIM Structural Similarity Index Measure

SWaP Size, Weight, and Power

TMDS Transmission Minimized Differential Signaling

UAV Unmanned Aerial Vehicle

Chapter 1

Introduction

This research explores the feasibility of using field programmable gate arrays (FPGA) for the deployment of computer vision systems to Edge environments. We detail the design and development of an FPGA-based system that leverages custom hardware cores for CNN inference as well as signal degradation and correction. Next we provide benchmarking via various experiments to analyze our FPGA system against an equivalent system implemented in a conventional GPU environment proving that the FPGA system is better suited for Edge deployment.

We developed an FPGA-based system that can implement various CNN deep learning models along with a library of custom hardware cores that can produce various noise degradations and corrections. We benchmarked the performance of this FPGA system via the use of two deep learning models, ResNet-50 and YOLO version 3, compared to equivalent models in a conventional GPU pipeline proving the FPGA as the more ideal solution for Edge deployment. We established both the baseline inference of these models between the two implementations as well as the impact of noise degradation and correction on the two hardware systems. This benchmarking of robustness of the two systems to in-the-wild conditions coupled with physical deployment requirements such as size, weight and power (SWaP) prove the potential of the FPGA as an appropriate platform for vision systems deployed to the Edge.

1.1 Motivation

The overall goal of this research is to develop an FPGA-based computer vision system, benchmark it against a GPU equivalent and then provide analysis and comparison regarding the robustness to image degradation across both systems. The motivation being to determine whether or not the FPGA-based system provides a level of inference performance and noise robustness similar to the GPU equivalent system while simultaneously delivering it in a much smaller SWaP footprint. Performing these development and analysis steps will allow us to determine if an FPGA-based deep learning system would provide an ideal platform for Edge deployment.

Edge systems such as autonomous vehicles, UAVs and smart cameras would benefit greatly from a low SWaP platform that includes robustness to “in-the-wild” imaging. For

instance, in (a) of Figure 1.1 an autonomous vehicle is evaluated during weather conditions which drivers often find themselves in [14]. Such extreme weather can confound the Edge AI systems making autonomous navigation dangerous to both the driver and pedestrians. In (b) of Figure 1.1 a security camera is shown in a high-risk setting for criminal activity [15]. Uncontrolled illumination in such areas is common and creates and additional layer of complexity to the security application. Finally, in (c) of Figure 1.1 an UAV is deployed for a rescue mission during a snow storm [16]. The drone has limits to payload weight capability as well as battery life but must also operate in non-ideal imaging scenarios. In all three of these Edge examples the processing unit responsible for vision computation not only requires low physical and power attributes but must also be robust to imaging degradations common in operational environments.

Motivation for using FPGAs for deep learning computer vision tasks was originally derived from previous work in the Imaging, Robotics and Intelligent Systems (IRIS) Laboratory by former graduate student, Steven Clukey. Specifically, this work explored the implementation of low SWaP video processing with FPGA systems [17]. After an initial FPGA prototype was validated, this work also lead to a production-grade ASIC (application specific integrated circuit) board which could be integrated into larger Edge environments for real-time, low SWaP video processing. This work targeted a smaller sized FPGA board (Xilinx Zybo) and used a mixture of tools such as Xilinx HLS (high level synthesis) [18] as well as the Chisel hardware prototyping language [19] for its development. Using this work as inspiration, we first extended the system enabling it to process HDMI input and output streaming video as well as file processing. This design was then targeted to be benchmarked on two newer Xilinx boards: Z1 and ZCU104 UltraScale+. The value of this hardware video processing architecture is that Edge systems are required to collect the data to be processed in the field effectively combining both the sensing and computation functions into one system.

Motivation for the next part of the thesis, implementation and benchmarking of deep learning models in FPGA, came from the proposed future work extensions in the aforementioned IRIS lab work. In particular, we hypothesized on whether the FPGA physical architecture could be efficiently used to implement those deep learning architectures that directly work with visual data (i.e. convolutional neural networks). When the work in this thesis was originally conceived there were no viable frameworks to support direct CNN implementation into FPGA fabric. Therefore, we began by exploring the implementation of layers individually as hardware IP cores. However, during the course of the thesis work a series of deep learning development frameworks became available from the FPGA vendor, Xilinx, which we ultimately leveraged to build the FPGA models used in our experiments. With these tools we decided to explore two application spaces for FPGA versus GPU evaluation: image classification and object detection. Our decision here was to determine if FPGA performance would be comparable to GPU performance and whether this performance would be consistent across problem space. Establishment of similar inference performance of FPGA-based CNNs as compared to GPU-based CNNs would demonstrate that low SWaP solutions for Edge computer vision tasks could be realized in FPGA hardware.

Given that we endeavor to prove the FPGA as a superior option of Edge deployment we were also motivated to demonstrate its capability to deal with common optical noises in such Edge environments. To measure the impact of image quality on inference performance we



(a) *Autonomous vehicle.*



(b) *Security camera.*



(c) *Unmanned aerial vehicle.*

Figure 1.1: Examples of Edge applications. *In (a) an autonomous vehicle's Edge sensors are tested under simulated heavy rain conditions at the Singapore-MIT testing center. The image in (b) demonstrates how cameras used for surveillance are often deployed in areas where illumination conditions are uncontrolled and highly variable. The UAV in (c) is deployed during a snow storm. Such conditions provide both mechanical challenges as well as add degradations to the frame quality of the video payload system.*

performed baseline experiments upon the two selected datasets to be evaluated with our two CNN architectures. The motivation for these experiments was to determine if a particular dataset could be biased towards a particular noise as compared to another dataset - thus skewing inference results. Inspiration for this analysis of robustness to degradation from deep learning models came from the work of another IRIS lab student, Matthew Seals, where in his masters thesis he evaluates object detectors with various noise types and other image manipulations (i.e. resolution) [20]. Therefore, we selected an image quality metric to determine the quantitative impact of our simulated noises. We additionally extended the work in [20] to include image classification networks as well as to investigate whether corrective processing could raise inference performance after degradation.

Lastly, to provide a true side-by-side analysis of the FPGA versus GPU as deep learning Edge systems we decided to benchmark an end-to-end pipeline of each system when configured with experimental degradation and correction parameters. Our motivation being to provide performance metrics that not only evaluate inference performance and associated robustness in the two hardware implementations but also compare SWaP characteristics to draw conclusions for desirability for Edge deployment. This final section of the thesis thus strives to compare GPU versus FPGA models when utilized in real world environments when non-ideal imaging for model probes is to be expected. Such an FPGA-based Edge system would offer an ideal platform for various Edge vision system tasks.

1.2 Pipeline

The pipeline which lays out the experimental evaluations benchmarking our FPGA and GPU systems appears in Figure 1.2.

1.3 Contributions

There are four contributions in this work, in particular a major contribution that leverages and extends three minor contributions: (1) we prove that FPGA-based CNNs have comparable robustness to image degradations as their GPU counterparts when such degradations are mitigated with optimally selected correction methods attesting the potential of the FPGA as an appropriate platform for vision systems deployed to the Edge (major contribution); (2) we developed an FPGA system for video processing, along with an accompanying library of custom processing hardware cores, to serve as the base FPGA design of the thesis pipeline and is extendable for our benchmarking experiments (minor contribution); (3) we established that FPGA-based ResNet-50 and YOLO version 3 models both incur only an approximate 1% drop in performance as compared to GPU-based models, proving similar performance across computer vision problem space, while being delivered in hardware with significant SWaP improvements (minor contribution); and, (4) we show a baseline statistical analysis of the impact of two degradation methods, along with corresponding correction methods, on the evaluated datasets via a quantitative image quality metric which illustrates that neither contain an inherent bias towards our evaluated methods (minor contribution).

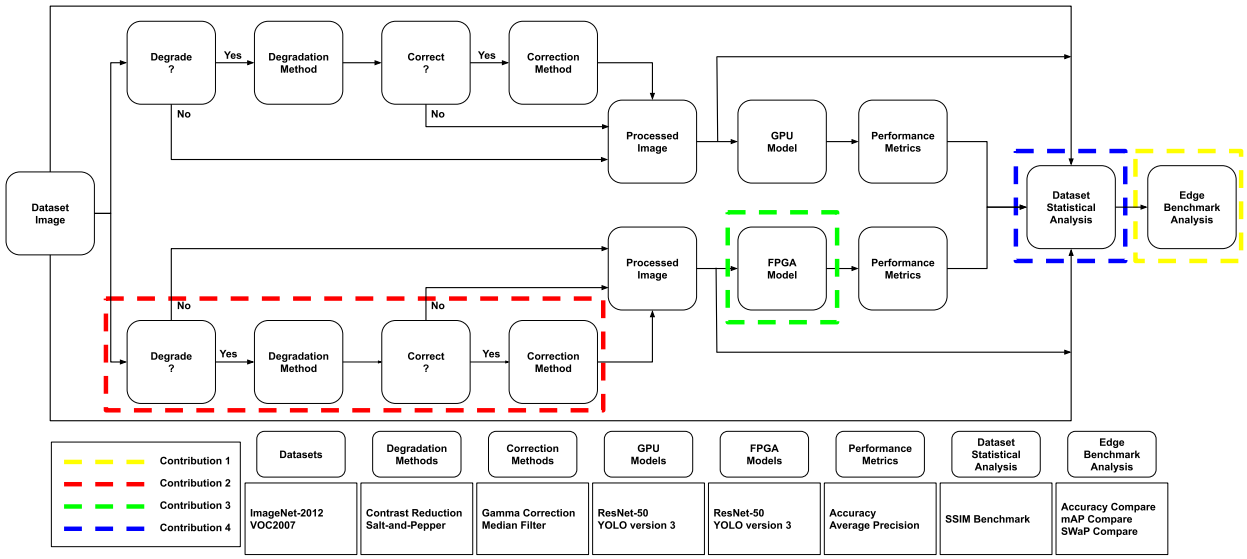


Figure 1.2: Pipeline for FPGA versus GPU benchmarking of CNN models with optional image degradation and correction. *From a high level the pipeline can be viewed as a GPU processing path (upper portion) and an FPGA processing path (lower portion) having common input (dataset image) and output (Edge benchmark analysis) endpoints. The sections of the pipeline corresponding to specific thesis contributions are marked with different color boxes (see figure contribution legend). The yellow box (contribution 1) represents the integration and extension of the following minor contributions into our major contribution of an Edge benchmarking framework evaluating our FPGA-based and GPU-based systems to prove which is optimal for Edge AI deployment. The red box (contribution 2) represents our FPGA video processing system. The green box (contribution 3) represents our FPGA CNN models which are used in benchmarking against GPU models. The blue box (contribution 4) represents our statistical analysis of the quality responses from our evaluated degradations and corrections on our selected datasets. The pipeline is organized such that different experimental setups can be evaluated. This experimental configuration is achieved by first selecting a CNN and associated dataset, then determining whether an unaltered input is to be benchmarked (original dataset image) or by selecting to apply one of the two degradation noises and/or applying an associated correction function to the degradation. Once the experiment parameters are determined, the comparison begins with the selection of a testing image from the chosen dataset. Next, depending on the experimental setup, we choose whether to degrade the image with one of the possible degradation noises. Once again depending on the parameters of the experiment, an associated correction function is optionally applied to the image. At this point the now processed test image is evaluated by the equivalent models in the GPU and FPGA paths as well as to be compared to the unaltered input image downstream to calculate a SSIM score. Depending on the model, the appropriate method is selected to evaluate the individual inference predictions related to ground truth. Lastly, the performance of the GPU and FPGA predictions, the related input SSIM scores, and the SWaP characteristics of the two platforms are compiled to provide Edge suitability benchmarking comparison of the experimental setup.*

Contribution 1 (major)

In our principal contribution we prove that FPGA-based CNNs have comparable robustness to image degradations as their GPU counterparts when such degradations are mitigated with optimally selected correction methods attesting the potential of the FPGA as an appropriate platform for vision systems deployed to the Edge. This contribution integrates and extends our three minor contributions and is marked by the yellow box in 1.2. Both models were evaluated with our two degradation methods (contrast reduction and salt-and-pepper noise) as well as evaluated when the associated correction method (gamma correction or median filter) is applied. For both the ResNet-50 and YOLO version 3 models it was observed that as the contrast reduction degradation parameter was increased inference performance dropped off at a quicker rate for the FPGA implementation demonstrating that the quantized weights of the hardware model were less robust to this sort of noise. With contrast reduction degradation (noise parameter = 20.0%), the GPU ResNet-50 model top-1 performance dropped by 12.96% whereas the FPGA ResNet-50 model dropped by 19.92%. Likewise, with contrast reduction degradation (noise parameter = 13.3%) the GPU YOLO version 3 model mean average precision dropped by 7.52% whereas the FPGA YOLO version 3 model mean average precision dropped by 12.1%. However, both the FPGA and GPU models had a near identical drop in the rate of inference performance for the same parameters when the salt-and-pepper degradation type was introduced. For instance, with a salt-and-pepper degradation (noise probability = 0.05) the GPU ResNet-50 top-1 accuracy dropped by 35.38% and the FPGA ResNet-50 top-1 accuracy dropped by a near identical 35.54%. With a salt-and-pepper degradation (noise probability = 0.05) the GPU YOLO version 3 mean average precision fell by 37.77% and the FPGA YOLO version 3 mean average precision fell by 35.07%. Therefore, for this degradation, the FPGA and GPU demonstrated similar robustness. When gamma correction was applied to contrast reduction degradation both of the FPGA and GPU models recovered significantly. For instance, with optimal gamma correction applied to contrast reduction degradation (noise parameter = 20.0%), the GPU ResNet-50 model top-1 performance increased by 8.6% to achieve 70.08% and the FPGA ResNet-50 model top-1 performance increased by 12.83% to achieve 66.25%. Similarly, with optimal gamma correction applied to contrast reduction degradation (noise parameter = 13.3%), the GPU YOLO version 3 model mean average precision increases by 5.04% to achieve 75.90% and the FPGA YOLO version 3 model mean average precision increases by 9.58% to achieve 74.92%. As with the salt-and-pepper degradation, the median filter correction sees near identical response from both GPU and FPGA models. With an optimal median filter correction applied to salt-and-pepper degradation (noise probability = 0.05) the GPU ResNet-50 model top-1 accuracy increases 33.68% to achieve 72.74% performance while the FPGA ResNet-50 model top-1 accuracy increases 34.06% to achieve 71.86%. Likewise, with an optimal median filter correction applied to salt-and-pepper degradation (noise probability = 0.05) the GPU YOLO version 3 model mean average precision increases 33.83% to achieve 74.52% performance while the FPGA YOLO version 3 model mean average precision increases 31.76% to achieve 74.13%. Thus, when appropriate correction methods are applied, the FPGA implementation provides similar robustness to noise as well as inference performance to its conventional GPU counterpart while maintaining a much smaller SWaP profile. We consider this demonstration of the superiority of the FPGA-based

solution over the GPU-based solution for Edge vision system deployment to be a major contribution to the thesis work.

Contribution 2 (minor)

The first minor contribution of this thesis was the development of an FPGA-based video processing pipeline (see red box in 1.2). We achieved this using various Xilinx design tools creating first a video processing architecture which could receive a HDMI input source, process the signal, and then output this new signal to a HDMI output sink. In addition to this base design was the development of a library of video processing IP cores that could be placed within the FPGA data stream to alter the video signal as desired. This base FPGA system and library of video processing cores were implemented in several Xilinx FPGA boards whose performance achieved 30 frames-per-second at a resolution of 1920x1080. This base FPGA video processing system was designed to be extendable for the experiments in our other contributions. We consider this FPGA base system to be a minor contribution to the thesis work.

Contribution 3 (minor)

In our next minor contribution we established that FPGA-based models (ResNet-50 and YOLO version 3) both incur only an approximate 1% drop in performance as compared to GPU-based models, providing similar performance across both image classification and object detection problems, while benefiting from the significant SWaP characteristics of the FPGA. After developing the base FPGA video processing system we extended this architecture to include deep learning computer vision IP cores in the hardware processing pipeline (see green box in 1.2). The CNN hardware IP cores themselves were created leveraging Xilinx's DNNDK tool. This tool takes a pre-trained model from several popular deep learning frameworks and quantizes the model weights from 32-bit floating point to 8-bit integer representation for FPGA implementation. For our side-by-side analysis of FPGA versus GPU performance we decided to target two computer vision tasks: (1) image classification via the ResNet-50 CNN architecture; and, (2) object detection via the YOLO version 3 CNN architecture. The ResNet-50 model was trained on the ImageNet-2012 training dataset (approximately 1.2 million images) and tested on a subset of the ImageNet-2012 testing dataset (5,000 images) in which the images are assigned to 1,000 classes. The YOLO version 3 model was trained on the VOC2007 training dataset consisting of 9,963 images with 24,640 annotated objects and tested on the VOC2007 testing dataset (4,952 images) in which images can fall under 20 classes. Our FPGA models were evaluated using a Xilinx ZCU104 UltraScale+ evaluation board and the original floating point models on a Nvidia RTX 2070 Super. Model performance was measured via accuracy for the image classification task with ResNet-50 and the FPGA model was shown to have an 1.1% drop in the top-1 accuracy and a 0.54% drop in top-5 accuracy as compared to the GPU solution. The FPGA had a slight boost in frames-per-second performance, 144fps for FPGA versus 140fps for GPU. Although the FPGA solution is approximately half the size and weight of the GPU solution, of particular note is the performance-to-power metrics. When consuming maximum system power, the FPGA-based system achieves 2.4 fps/W whereas the GPU based-solution

achieves 0.65 fps/W. Performance for the object detection task with the YOLO version 3 model was measured with mean average precision and the FPGA had a 1.02% drop in mAP as compared to the GPU solution. In contrast to the ResNet-50 model, the GPU model had a slight boost in frames-per-second performance, 29fps for the GPU versus 27fps for the FPGA. When consuming maximum system power, the FPGA-based system achieves 0.45 fps/W whereas the GPU based-solution achieves 0.13 fps/W. We consider this side-by-side benchmarking of the two CNNs on our FPGA system against a conventional GPU system which establishes that FPGA models can achieve near identical performance to conventional GPU models to be a minor contribution to the thesis work.

Contribution 4 (minor)

In this final minor contribution we show a baseline statistical analysis of the impact of two degradation methods, along with corresponding correction methods, on the ImageNet-2012 and VOC2007 datasets via a quantitative image quality metric which illustrates that neither contain an inherent bias towards our evaluated methods. After establishing that the FPGA system provides near identical inference performance compared to the GPU counterpart we next wanted to explore its robustness to corrupted signals. Signals in unconstrained scenarios where Edge systems are deployed are often degraded due to environmental conditions, electro-mechanical interference, or other factors. Therefore, determining if the FPGA model would provide similar robustness to noise allows for one measure of explainability when an incorrect inference is made. Our experiments evaluated the impact of two degradation methods (contrast reduced images and salt-and-pepper noise) as well as their correction methods (gamma correction and median filter, respectively). Before determining the impact on inference with our FPGA and GPU models, we designed experiments to establish whether the datasets used in our evaluations were biased to any of our selected degradation methods. This contribution is shown as an extension to our pipeline in the blue box in 1.2. The structural similarity (SSIM) score, an image quality analysis technique, was used to provide a quantitative analysis of the impact of the degradations and corrections explored in this thesis. As predicted, these experiments shown that we could mimic real-world noises and correct the degradation, as well as provide a quantitative score to each image to provide statistical significance in our final experiments. Additionally, we demonstrated that neither evaluated dataset is biased towards any selected degradation or correction method. We consider this baseline statistical analysis of our selected processing functions to our selected testing datasets to be a minor contribution to the thesis work.

1.4 Synopsis

In Chapter 2 we present the current literature and previous work related to this thesis. The scope of this literature review includes the relevant areas of Edge-based systems, hardware video processing, FPGA-based deep learning systems and the impacts of image degradation upon CNN inference. The intent of this literature review is to discuss previous research in these areas that served as both inspiration as well as what could be leveraged in our experimental designs.

Chapter 3 lays out the development of our FPGA video processing architecture and its performance metrics. This architecture served as a foundation system that was further extended to accommodate the experimental needs of our FPGA-based Edge system. Our system demonstrates that FPGAs are suitable for the needs of real-time video processing by achieving 30 frames-per-second processing at a resolution of 1920x1080.

In Chapter 4 we benchmark two deep learning models in FPGA and compare their performance against GPU implementations of the same models. We selected models from two computer vision spaces (ResNet-50 for image classification and YOLO version 3 for object detection) to demonstrate that the FPGA is viable across different tasks. Our FPGA-based system leverages 8-bit integer quantized versions of the 32-bit floating point models used in the GPU-based system. Our findings establish that our FPGA-based system has near identical inference performance (a minimal drop of approximately 1% for both models) as the GPU-based system while also realizing a reduction in all SWaP characteristics. In particular, the FPGA-based solution achieved 2.4fps/W (ResNet-50) and 0.45fps/W (YOLO version 3) as compared to 0.65fps/W (ResNet-50) and 0.13fps/W (YOLO version 3) for the equivalent models on the GPU-based solution.

Chapter 5 details the particular noises we selected to degrade our two datasets with as well as the associated noise reduction methods we leverage for correction. Specifically, we consider the noises of contrast reduction with parameters of 100% (no noise) to 2.5% of original dynamic range and salt-and-pepper noise using probability of noise from 0% (no noise) to 0.1%. To reduce the impact of contrast reduction we selected gamma correction with gamma parameters of 1.0 to 2.0. To reduce the impact of salt-and-pepper noise we selected median filtering with kernel sizes of 3x3 to 11x11. Additionally, we establish the use of a baseline image quality metric (SSIM) to provide a quantitative score of overall quality once an image is either degraded with noise or had such noise mitigated with a correction measure. The SSIM score also provides insight into finding the ranges of noise parameters in which degraded images are no longer able to be corrected such that any relevant quality can be achieved. The intent of this experiment is to show experimentally that neither of our selected datasets had an inherent bias to any of our selected noises and associated corrections. Our results both demonstrate that the ImageNet-2012 and VOC2007 datasets respond near identically to any combination of degradation and/or correction as well as that the SSIM score can be used to predict the impact on model performance when benchmarking for robustness. The experimental results of this section provide further evidence to support the findings from our major contribution.

Chapter 6 provides the analysis of both the FPGA and GPU models when image degradation and optional correction is present in the inference pipeline proving our FPGA solution as the superior candidate for Edge deployment. This chapter details the major contribution of this thesis which leverages the findings from the three minor contributions laid out in the previous chapters. Figure 1.3 provides an example of an instance were noise confounds both the FPGA and GPU ResNet-50 models but a proper inference is made after noise reduction. The top left image in Figure 1.3 shows an unaltered image from the ImageNet-2012 dataset (both the FPGA and GPU models label correctly) and the top right image shows the original with contrast reduction set to capture only 20% of the original dynamic range. Neither the FPGA-based or GPU-based model correctly identifies this degraded image. However, both models are able to infer the image correctly (lighthouse)

as shown in the bottom row of images in Figure 1.3 when a gamma correction with parameter 1.5 is applied. For the entire dataset when contrast reduction of 20% is applied the GPU top-1 accuracy drops to 61.48% and the FPGA top-1 accuracy drops to 53.42% with associated SSIM score of 0.2563. However, when optimal gamma correction is applied the GPU top-1 accuracy improves 8.6% and the FPGA top-1 accuracy improves 12.83%. This trend of larger drops with degradation but greater improvements with noise reduction is consistent with FPGA-based models when handling the contrast reduction degradation. Similarly, Figure 1.4 provides an example of an instance were noise confounds both the FPGA and GPU YOLO version 3 models but a proper inference is made after noise reduction. The top left image in Figure 1.4 shows an unaltered image from the VOC2007 dataset (both the FPGA and GPU models label correctly) and the top right image show the original with salt-and-pepper noise with a probability of 0.05. Neither the FPGA-based or GPU-based model correctly identifies this degraded image. However, both models are able to infer the image correctly (two instances of class bicycle) as shown in the bottom row of images in Figure 1.4 when a median filter correction with kernel size of 3x3 is applied. For the entire dataset when salt-and-pepper with probability of 0.05 is applied the GPU mean average precision drops to 68.47% and the FPGA mean average precision drops to 69.34% with associated SSIM score of 0.8141. When optimal median filter correction is applied the the GPU mean average precision improves 7.49% and the FPGA mean average precision improves 6.12%. For the salt-and-pepper noise both the FPGA-based and GPU-based models have near identical robustness. With this major contribution we prove that the FPGA-based solution can provide similar robustness to degradations as the GPU-based solution but with significant improvements in SWaP characteristics making it more ideal for Edge deployment.

Finally, Chapter 7 concludes the thesis with a review of the findings from our work. We detail the importance of each of the contributions and discuss how these findings establish the FPGA-based system as the more ideal candidate for Edge deployment. We conclude this chapter with proposed potential extensions for future expansion of our work.



(a) *Original.*



(b) *Contrast at 20.0%.*



(c) *GPU Inference - Gamma Corrected.*



(d) *FPGA Inference - Gamma corrected.*

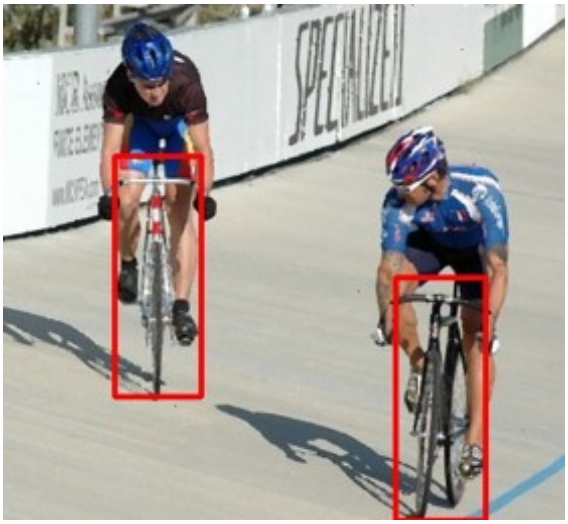
Figure 1.3: ResNet-50 example testing image with contrast reduction degradation (20.0%) and gamma correction (1.5). *The original image before degradation appears in the top left. The top right image shows the result after contrast reduction is applied with a parameter of 20.0%. Both the GPU and FPGA failed to properly classify the image when contrast reduction degradation at 20.0% was applied. However, as the bottom two images demonstrate, both the GPU and FPGA models provide the correct inference class label (lighthouse) when a gamma correction of 1.5 is applied to the contrast reduced image.*



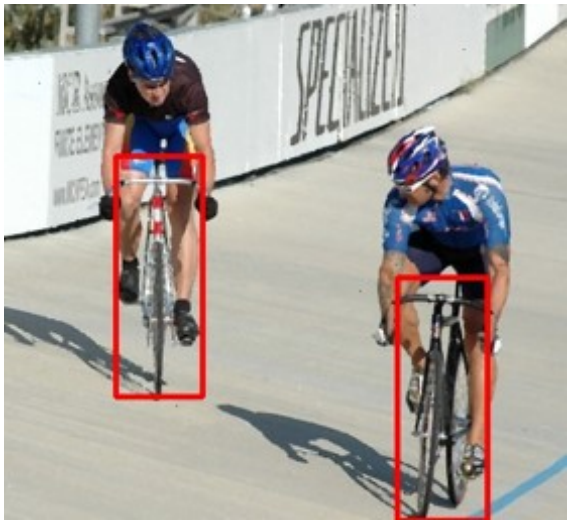
(a) *Original.*



(b) *Salt-and-Pepper at 0.05.*



(c) *GPU Inference - Median Filter Corrected.*



(d) *FPGA Inference - Median Filter corrected.*

Figure 1.4: YOLO version 3 example testing image with salt-and-pepper degradation (0.05) and median filter correction (3x3). *The original image before degradation appears in the top left. The top right image shows the result after salt-and-pepper is applied with a parameter of 0.05. Both the GPU and FPGA failed to properly detect the objects in the image when salt-and-pepper degradation at 0.05 was applied. However, as the bottom two images demonstrate, both the GPU and FPGA models provide the correct inference class label (2 instance of class bicycle) when a median filter correction of 3x3 is applied to the salt-and-pepper degraded image.*

Chapter 2

Literature Review

In this section we discuss the background of four key areas related to the work in this thesis. The ordering of these topics is based upon the timeline of which the thesis work was developed. Additionally, each of these four topics are reflected in the thesis pipeline we describe in Figure 1.2. We first explore the concept of Edge systems providing example applications and relevant work. Next we explore the research related to leveraging FPGA systems for high performance video processing. The next section of this chapter focuses on related work in the implementation of deep learning networks into FPGA fabric. Finally, we review research on the impact of image degradation upon CNN inference performance.

2.1 Computer Vision with Edge Systems

The field of computer vision began as part of the broader subject of artificial intelligence in the 1960s. Computer vision heavily leverages the tools of image processing in its goal to automate the human visual decision making process for various applications. Some of its underlying tasks include 3D scene reconstruction, motion estimation and object detection. Image processing tasks such as edge detection and quality enhancement are therefore fundamental components in the performance of such applications.

Recently, high performance computer vision applications have entered the mainstream via both industrial and consumer products. In particular, object detection has been a heavily explored fundamental problem in computer vision for the past several decades. Object detection comes in various varieties such as image classification, general object detection with localization, and face detection. For example, numerous historical approaches such as the Viola-Jones detector and the Histogram of Gradients (HOG) detector have shown promise but still suffer from under-performing accuracies and are notoriously slow in their calculations. Since Alexnet's [21] victory of the ImageNet Large Scale Visual Recognition Challenge in 2012, the task of object detection has experienced a renaissance from leveraging the computational prowess of convolutional neural network based deep learning methods. After Alexnet, numerous CNN architectures have been developed which achieve even greater performance, some of which even challenge human performance at object detection and image classification. Some of the more recent of these networks are Faster R-CNN, YOLO version 3, and InsightFace/ArcFace which all provide state-of-the-art performance.

The inherent computational complexity of deep CNNs can be leveraged to make object detection more generalizable than previous traditional methods. Such deep architectures were explored prior to AlexNet's victory but were too computationally expensive to implement even in the best CPUs of the time. Therefore, the success of deep learning models for vision tasks is not only a product of the advancements in the architectures themselves but also due to the progress of the underlying computational hardware that the architectures rely on [22]. Ultimately it was the implementation of the CNN within the graphics processing unit (GPU) that provided the fundamental hardware to make these object detection architectures viable. Since 2012, GPUs are the de-facto hardware considered during both the training and inference phases of deep learning.

2.1.1 Overview of Edge Systems

Although GPU implemented CNNs provide the desired detection performance the system architecture does not lend itself to applications beyond an office work station or cloud computing center. This limitation comes about due to the required host system, the weight of both the host and GPU itself as well as the associated high power demands. Additionally, more powerful cloud-based solutions cannot necessarily meet the strict timing requirements needed at the Edge [23] and [24]. Therefore, such solutions do not lend themselves to low SWaP (size, weight and power) constrained environments. Provided that studies such as [25] predict that at least 50 billion Internet-of-Things (IoT) devices will be deployed worldwide by 2020, the need for a more efficient Edge computing platform is apparent [26], [27], and [28].

As seen in figure 2.1, deep learning applications can be deployed in various environments. Traditional GPU deployments occur either in the on-site environment with standard desktop computers or via Cloud computing options where the work-horse GPUs are located remotely and accessed via internet connections. However, neither of these environments lend themselves to the aforementioned low SWaP implementations which are designed to be deployed "in the wild" as extensions to drones, autonomous cars, etc. Due to the rapid rise in technology that can greatly benefit from low SWaP deep learning, the so-called Edge environment has arisen.

Unlike other deep learning deployment options, the physical specifications are of utmost importance for Edge systems. As mentioned, SWaP characteristics are typically of principal importance but additional constraints include ruggedization of equipment, physical security features, and typically a requirement for the resource to always be available. This last feature can be envisioned for autonomous cars where a pedestrian avoidance system is critical to be continuously functional whereas a cloud-based movie recommendation service does not pose a safety concern if it is down. Edge systems also provide an additional layer of security in that probe inputs are consumed and processed locally instead of communicated up to a centralized server. Such a feature can be highly desirable in deployed systems for human biometrics.

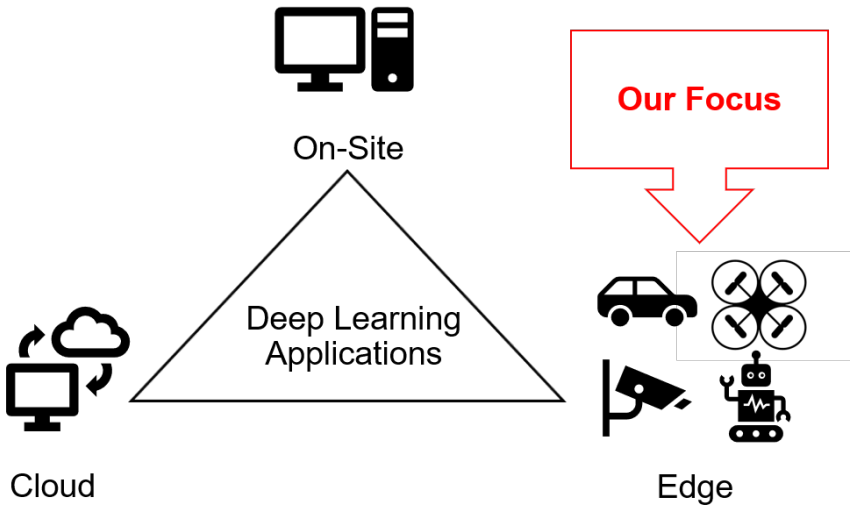


Figure 2.1: Deployment options for deep learning applications. *Traditionally, deep learning applications are used on-site on individual workstations or in high-performance computing (HPC) centers. Due to increases in high-speed internet communication, Cloud-based solutions allow projects with limited budgets to affordably lease remotely located equipment. Advances in hardware which allow deep learning on smaller physical devices have created so called Edge AI platforms to service systems deployed into the real world such as autonomous cars, drones, medical devices, robotics and surveillance. The focus of our research in this thesis lies within this domain of Edge AI devices.*

2.1.2 Options for Edge Implementation

Figure 2.2 provides a quick summary of the primary hardware options currently available for Edge deployment. As imagined, the selected hardware comes with its associated benefits and weaknesses. For development purposes, the conventional hardware of CPUs and GPUs are very flexible and are considered much easier to design than FPGAs and ASICs. On the other hand, the performance oriented options of FPGAs and ASICs allow for high inference speeds and energy efficiency. Table 2.1 further explores the pros and cons of these four options along several parameters. CPU-based systems are easy to program and debug and can leverage standard deep learning libraries. CPUs are the most ubiquitous Edge systems given their usage in modern smart phones. However, CPUs have the slowest inference speeds making them non-ideal for real-time constrained systems. Additionally, CPUs require an OS to manage general purpose tasks further reducing real-time capabilities as well as providing additional exposure to security breaches. GPU-based systems provide the easiest development platform given that all modern deep learning frameworks target GPUs. GPU systems, like CPU systems, are easy to develop and debug but provide far superior inference speeds as compared to CPU systems. Like CPU-based systems, GPU systems are reliant upon a managing OS because they require a CPU-based ecosystem in which to reside. This ecosystem further increases their SWaP footprint which is significant given they have the highest power and cooling requirements of any other option. FPGA-based solutions provide very high inference speeds while maintaining a small SWaP footprint. Unlike CPUs and GPUs, they don't require a managing OS or larger ecosystem in which to operate. However, FPGAs have very few options for deep learning development, particularly those that leverage standard programming frameworks. Finally, ASIC-based systems provide the best SWaP characteristics along with higher inference speeds as compared to FPGAs. Like FPGAs, the development life-cycle does not leverage conventional deep learning libraries and requires additional hardware knowledge. Unlike FPGAs, which are programmable, ASICs are not easily updated to modify model weights or incorporate new SOTA architectures.

As represented in figure 2.3, the work in this thesis requires the integration of several research domains to properly address Edge systems deployed into unconstrained environments: real-time hardware based video systems, deep learning models deployed in hardware, and inference robustness through noise correction. Edge applications such as autonomous cars, unmanned drones, and field-deployable surveillance equipment all have physical constraints that the traditional deep learning environment cannot satisfy. Therefore, a need to provide solutions for such use cases must be explored. Based upon previous work from the Imaging, Robotics and Intelligent Systems (IRIS) lab's work [17], as well as its ability to leverage high inference speeds similar to ASICs while still allowing for model customization, we propose leveraging the low SWaP platforms of FPGAs as a solution for such constrained environments where both high accuracy and low physical demands are required. Additionally, like GPUs, FPGAs leverage highly parallelizable architecture making them ideal for image and video processing. Such a solution can bring real-time computer vision technology to a market where size, weight and power consumption is highly limited.

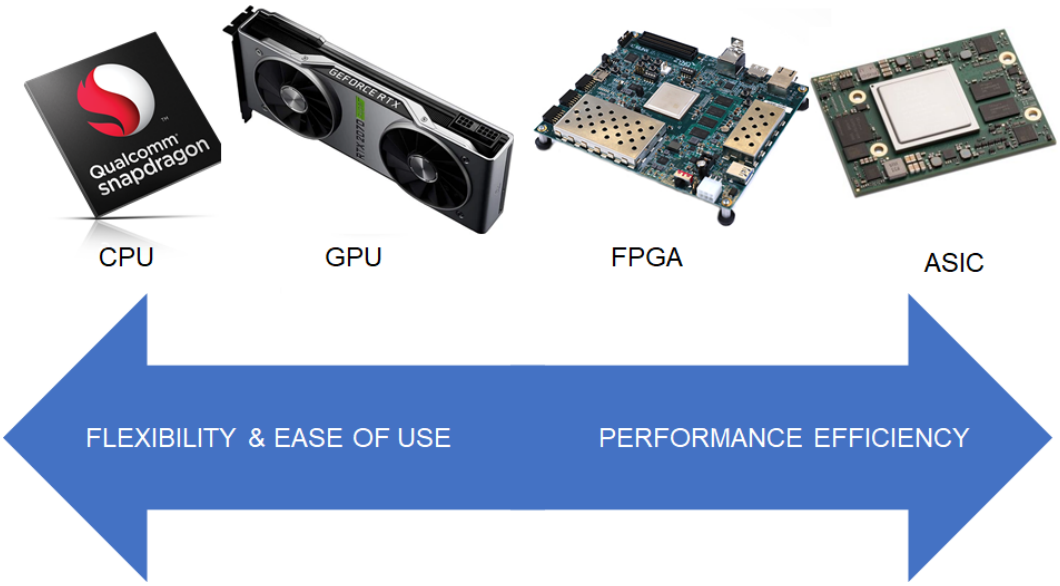


Figure 2.2: Spectrum of Edge AI platforms. *Platforms to the left are more flexible and are easier to program. Platforms to the right achieve higher performance efficiency in terms of size, weight, power and inference speed. In choosing the platform used for Edge deployment one should consider all of the parameters of the project ranging from size, cost, power consumption and inference requirements.*

Table 2.1: Comparison of Edge AI Hardware devices. *For Edge deployment, the device must not only meet inference performance parameters but also physical parameters required by the system. Each device's parameters is ranked from poorest to optimal (worst, bad, good, best).*

PARAMETER	CPU	GPU	FPGA	ASIC
Programming Ease	BEST	GOOD	BAD	WORST
Available Libraries	GOOD	BEST	GOOD	WORST
Inference Speed	WORST	GOOD	GOOD	BEST
Physical Dimensions	BAD	WORST	GOOD	BEST
Weight	BAD	WORST	GOOD	BEST
Power Efficiency	BAD	WORST	GOOD	BEST
Ease of Model Updates	GOOD	BEST	GOOD	N/A

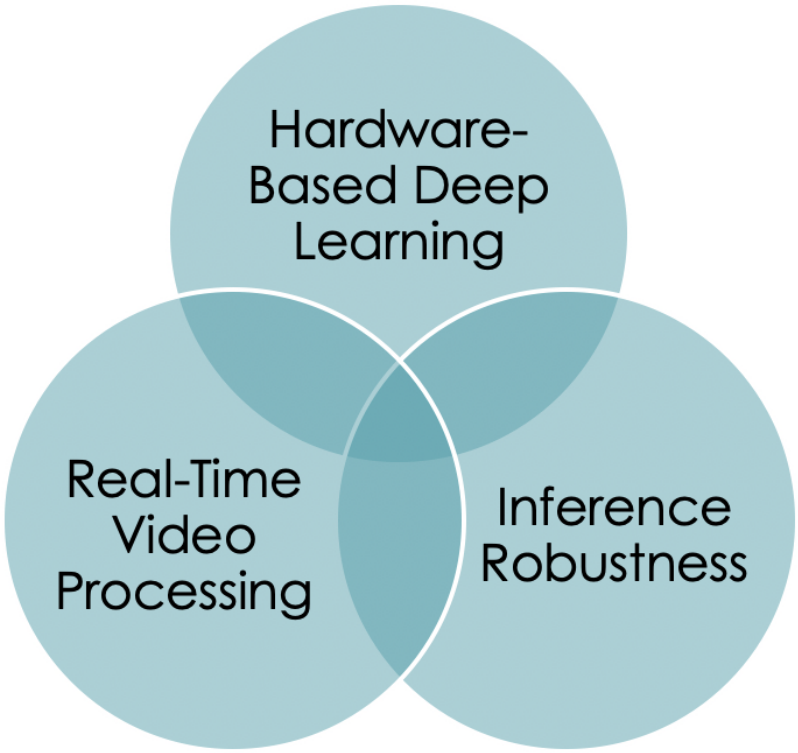


Figure 2.3: The thesis work lies at the intersection of three research areas. *The goal being to leverage the capabilities within these branches of research for the design and evaluation of robust, low SWaP computer vision applications.*

2.2 Review of Video Processing Systems in FPGAs

This section covers several technical areas related to FPGA design. First, we explore FPGA architecture in general and the Zynq line of fabric in particular. Next, frameworks for fast FPGA prototyping are explored. Finally, the general hardware design needed for streaming video hardware systems is described in detail. The content in this section therefore provides a basis for the use of FPGA-based Edge systems for deep learning applications in the subsequent sections.

2.2.1 FPGA and Zynq Architecture Overview

As laid out in the previous section, the explosion of top performing deep learning networks is not only due to advancements in design, but also due to advancements in the processing power of the hardware this technology leverages. Market trends have lead to great advances in the performance of CPUs and GPUs which are traditionally used in both training and inference stages of deep learning systems. In particular, market demands for high performance computation for several domains has driven the fast growth in GPU performance. However, as laid out in the previous section, GPUs do come with limitations that can offset their performance gains. In particular, they require large power supplies, and are significant in size and weight parameters - all of which stand at odds with low SWaP principals. Although large cloud options are available to mitigate some of these issues, these environments cannot meet the constraints of real-time performance which can be limited by the required high-speed connection to the system. This same potential bottleneck can also be completely unavailable if a system needs to be deployed into a region where such communications are limited. Reasons such as these are why we explore the feasibility of the FPGA as a hardware option for deep learning tasks in this thesis.

FPGAs are integrated circuit boards which allow for re-programmable hardware solutions. Figure 2.4 lays out the fundamental components of the FPGA fabric [1]. The fabric of an FPGA is composed of an array of logic blocks which are interconnected by a series of high speed communication buses. These buses are used to send timing, data and control signals to manage both the sequencing and data between the logic blocks as well as to physical I/O connections to other sensors or systems. The logic blocks themselves typically contain a look-up-table and flip-flop (or some other memory element). This array of hardware components in the FPGA fabric can be configured to perform highly complex combinational functions to implement sophisticated circuit designs. Configuration of the fabric is achieved by hardware description languages (HDLs) such as VHDL and Verilog, which are often described as languages that "transform software to circuits." FPGAs are often used to prototype physical designs before implementation into ASICs for production but are also very useful when a physical system needs to be updated in the field after final deployment.

The Zynq architecture from Xilinx offers an efficient approach to FPGA development by tightly coupling the flexibility of a CPU to the high throughput capability of an FPGA fabric. This interconnection between the CPU or processing system (PS) and the FPGA fabric or programmable logic (PL) is serviced by high speed AXI interfaces [2] as shown in figure 2.5. The tight coupling of the PS and PL allows the designer to have freedom in choosing whether to place all computation on the PL or offload some non-critical functionality to the

PS. A more detailed reference for the Zynq architecture can be observed in figure 2.6 [2]. The PS portion of the Zynq is powered by an ARM Dual Cortex A9 processor with CPU frequency and cache availability and is offered at different scales for project needs. The PL is comprised of configurable logic blocks, digital signal processing blocks, and dedicated block RAM, available in different sizes depending on the model selected from the Zynq line of options.

There are numerous Xilinx Zynq-based chips that provide a wide range of functionality and performance. Several of these Zynq chips are also available as components in development boards which are marketed by various vendors such as Digilent and Avnet. Over the course of this thesis work three of these development boards were used extensively and can be seen in Figure 2.7 [3], [4], and [5]. Each of these development boards possess a selection of various GPIO, communications, and other functionality, but (besides the FPGA fabric size itself) the most significant for the work in this thesis is the video in/out capabilities. The Zybo board has one HDMI port that can be used for input or output and a single dedicated VGA output port. The newer Z1 removed the VGA port and provides both a dedicated HDMI in and HDMI out port. Therefore, the Z1 provided the minimum baseline resolution of this work at 1080p. Alternatively, the Z1 fabric is limited in its speed and cannot achieve 60 fps at this resolution. However, the ZCU104 board provides both dedicated HDMI in and out ports but also a dedicated Displayport output and a USB 3.0 input that can be used for video. The ZCU104's Ultrascale Zynq fabric can also support 4K resolutions up to 60 fps - the principal reason it was chosen as the final implementation board for this thesis work.

The Zybo and Z1 boards both use chips from the Zynq 7000 series line (XC7Z010 and XC7Z020, respectively) whereas the ZCU104 chip is a Zynq Ultrascale+ (XCZU7EV). A comparison of the features of each of these chips can be seen in table 2.2. Of note should be the amount of logic cells and size of block RAM offered by the chip (which influences the maximum size of configurable hardware that can be programmed) and the number of DSP slices (which influences the speed of operation). As mentioned earlier, all of these chips are based on Xilinx's Zynq architecture and therefore contain several high speed AXI lines for communication between the ARM processor and FPGA fabric on the chip.

2.2.2 Frameworks for Fast FPGA Prototyping

The traditional development process for FPGA implementation appears in Figure 2.8. The circuit is first represented by a HDL which describes the structure and behavior of the design. Next, (after a recommended design verification step), the design is synthesized which produces a netlist from the HDL giving a listing of what functionality will be eventually configured in the FPGA. Next, any constraints placed upon the system (i.e. timing, etc.) are modeled along with the netlist. This information is then used during the floor planning stage to determine the optimal placement of the modules in the netlist onto the physical fabric. During placement and routing a final optimization is performed to ensure that the wire interconnections between the logic blocks conforms to the constraints. Finally, a bitstream is generated which is the final file used to configure the FPGA fabric to behave based upon the original HDL circuit design. Not shown in Figure 2.8 is the highly important iterative step of system verification that happens after the HDL code is written. Here, a test bench is composed to exhaustively test all of the circuit behavior through a simulation process.

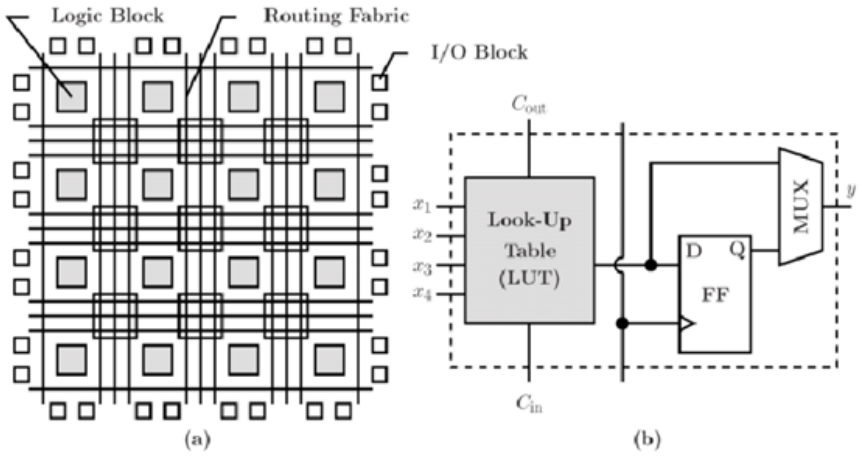


Figure 2.4: Physical layout of FPGA fabric [1]. *(a) The FPGA fabric is composed of an array of logic blocks interconnected with high speed buses and I/O blocks. (b) A logic block typically contains a look-up-table and flip-flop combination.*

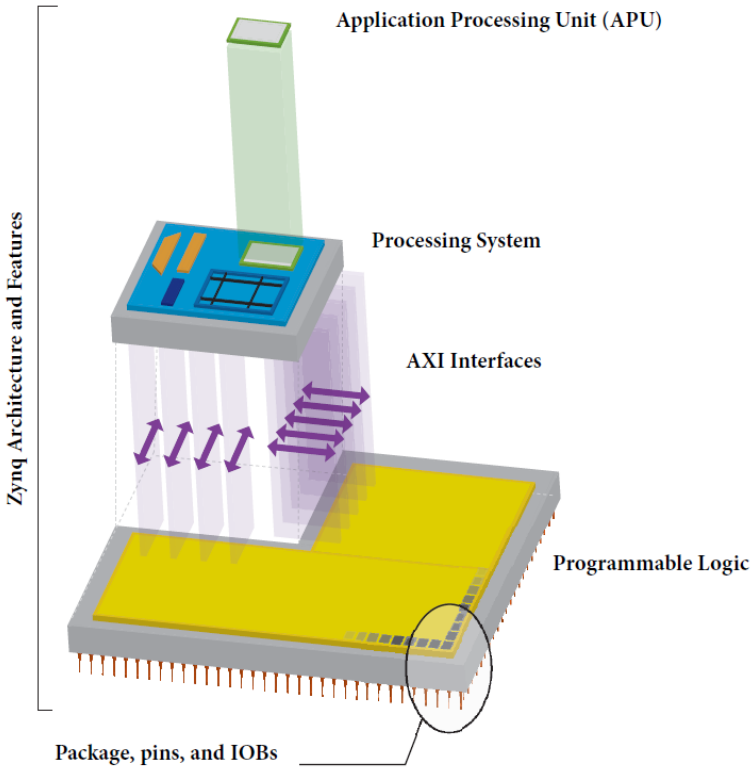


Figure 2.5: High-level representation of Zynq architecture [2]. *The ARM CPU is located in the Processing System (PS) area while the FPGA fabric is housed in the Programmable Logic (PL) region. The communication between the two is achieved via tightly-coupled, high speed AXI connections.*

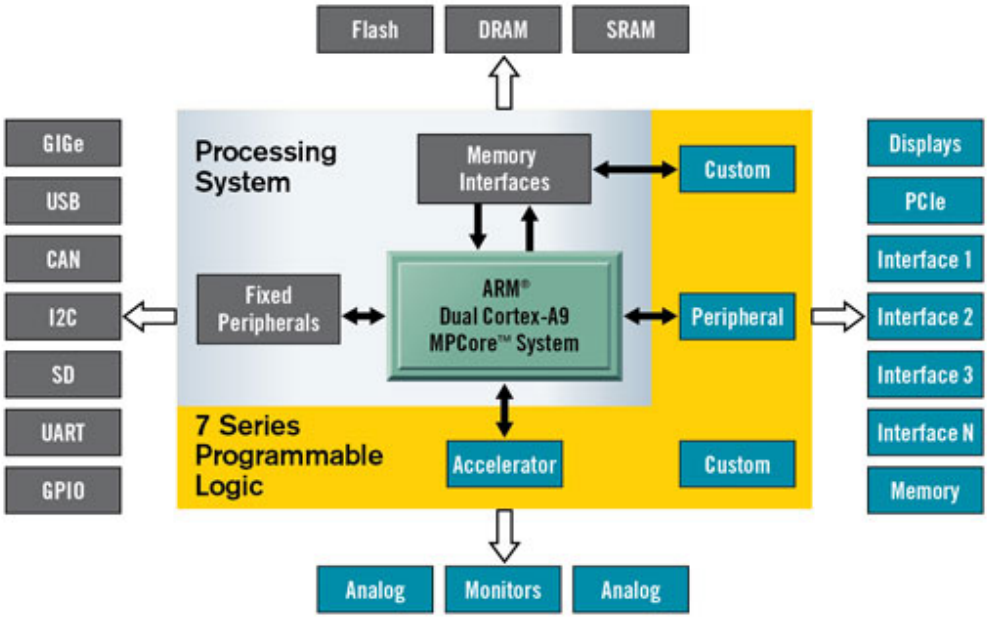


Figure 2.6: Detail of Zynq architecture resource interconnections [2]. *The PS has connection to memory interfaces and fixed communication peripherals such as UART and the SD card. The PL system hardware accelerators can either communicate directly with several peripherals to enhance performance or communicate through the PS to work with PS specific peripherals.*

Table 2.2: Comparison of FPGA development board resources. *Although each of the summarized FPGA chips are from the Xilinx Zynq line, they differ greatly in both the amount of configurable resources and in their support for video processing.*

BOARD	LUTs	Flip-Flops	BRAM	Processor	Video Support
Zybo	26,400	35,200	240 KB	650 MHz Dual-Core ARM A9	Single HDMI Port Single VGA Port 480p @ 30fps
Z1	53,200	106,400	630 KB	650 MHz Dual-Core ARM A9	Dual HDMI Ports 1080p @ 30fps
ZCU104 Ultrascale+	504,000	4,032,000	38 MB	Quad-Core ARM A53 + Dual-Core ARM A5	Dual HDMI Ports 1080p @ 30/60fps 4K @ 30fps

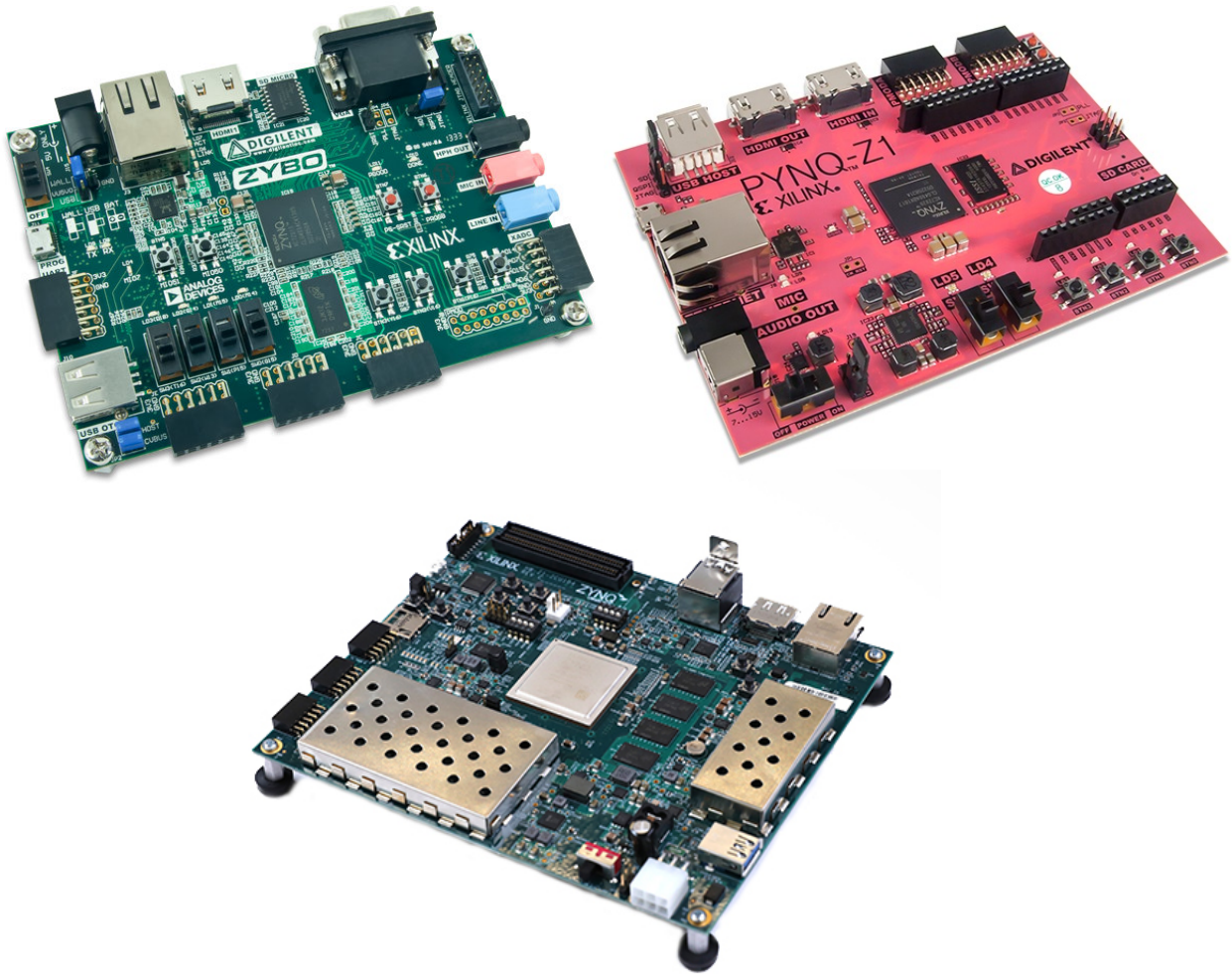


Figure 2.7: The series of Xilinx FPGA development boards used in this thesis work. *Clockwise from top-left: (1) Zybo Zynq-7000 Development Board [3], (2) Z1 Development Board [4], and (3) ZCU104 Ultrascale+ Development Board [5].*

After the system is fully verified it would then be ready to complete the remainder of the development life cycle.

As mentioned previously, FPGA system designs are coded in a hardware description language (HDL) such as Verilog or VHDL. However, such design requires the engineer to account for every wire and signal in the system which can become quite cumbersome for larger designs. The need to make FPGA prototyping more efficient has produced several HLS implementation languages [29].

Regardless if the HLS language originated in academia or in industry, the common goal is to convert a high level programming language to a HDL. Leveraging the same principals as object oriented programming from higher level programming paradigms, FPGA frameworks collectively known as high level synthesis (HLS) tools exist to increase FPGA design efficiency and reusability. Of the currently available HLS implementations, the vast majority accept either the C or C++ programming language as their input and typically allow either Verilog or VHDL as an output which is then synthesized for FPGA consumption. The benefit of HLS is that it affords faster prototyping of the circuit design as well as more efficient testing and debugging. For this thesis work we explored two different HLS tools: Chisel [19] and Vivado HLS [18].

The Chisel language (an academic HLS) had originally been used in the IRIS lab in previous work [17], and takes Scala as an input language. Initial attempts were made for this thesis work to prototype image processing IPs with Chisel but this was eventually abandoned for the more fully developed industrial HLS tool provided by Xilinx in their Vivado tool chain.

The Vivado HLS tool allows for prototyping in C/C++ and can handle C-based test benches to verify proper function of the design before advancing in the process. The process flow of IP core creation within Vivado HLS is: C Simulation, C Synthesis, C/RTL Cosimulation, and Export RTL. The C Simulation step involves the user generated C/C++ code to be ran through a hardware transformation simulation to ensure proper functionality. During C Simulation a test bench is written which allows for various testing techniques to achieve this end. During C Synthesis, code that has been shown to function properly is converted to its RTL (register transfer level) equivalent. This equivalent RTL code is then simulated concurrently with the original C/C++ code during the C/RTL Cosimulation step. This ensures that both the original and RTL versions produce the same results. A report at this stage also provides diagnostic information such as estimated run times. Finally, once the generated RTL code is shown to function properly the Export RTL stage exports the IP core into a package that can be used in the Vivado Design suite to be implemented by any other design.

Beyond its streamlined design for HLS prototyping, the Vivado HLS tool had two other advantages. First, being officially supported by Xilinx, the IP cores produced by this tool worked without need for additional modifications into a larger Vivado design project. Secondly, Vivado HLS had additional libraries to support specific use cases - in particular the HLS video library defined interfaces on how to directly convert streaming video input/output into the AXI interfaces required by the Zynq architecture.

2.2.3 FPGA Video System Design

With the widespread use of smart phones, the number of embedded devices with integrated imaging hardware has grown dramatically. Systems such as smart security cameras and drones also demand power-efficient, high-speed video transmission to achieve their goals. Therefore, the capability to process video at the Edge to provide input to deep learning models is of utmost interest. Surveys such as [30] have provided benchmarking on the energy efficiency of CPUs, GPUs and FPGAs for Edge devices and have shown that FPGAs are 1.2-22.3 times more energy efficient. This comes as no surprise given that FPGAs are particularly well suited for signal and video processing given their optimal architectures for processing parallel data.

Given the prevalence of imaging sensors and displays that utilize the HDMI interface for video we decided to target this video interface for our FPGA system design in this thesis. The HDMI interface allows for the streaming of uncompressed video data along with compressed or uncompressed audio data. The standard HDMI type A receptacle contains 19 pins which are mapped to power, ground, clock, and hot plug detection as well as for the transmission of the video and audio data via the transition-minimized differential signaling (TMDS) protocol. This protocol is designed for the high-speed transmission of serial data. To decode the TMDS signals from the physical HDMI source/sink for use in our FPGA video system we relied heavily on the Xilinx HDMI Subsystem guide [31].

To decode the TDMS signals, and for other video functionality, Xilinx has several resources to make FPGA prototyping more efficient. In particular, the AXI4-Stream interface and associated library of Xilinx cores can be leveraged to develop a video subsystem that is both modular and easily extensible to fit the goals of our thesis work. The AXI4-Stream transmits video data and is driven by both slave and master interfaces as seen in Figures 2.9 and 2.10 which leverages the AXI4 handshaking protocol in Figure 2.11 to send video frames [6].

In addition to the library of Xilinx AXI4-Stream related IPs, Xilinx also has a library of video functions which work with their HLS framework to expedite video system prototyping [32]. This library can be invoked within the HLS tool to decode an incoming video stream into the AXI4 protocol as well as encode the stream back to a desired protocol for downstream transmission. As laid out in Chapter 3, this library was used extensively in the development of our custom video processing IP cores.

On a final note, the Xilinx PYNQ project provides another resource that can be leveraged to prototype FPGA video systems. PYNQ is an open source FPGA project created by Xilinx that leverages the Python programming language in an attempt to bring FPGA design to an audience more accustomed to software design [33]. Papers such as [34] and [35] explore the use of Xilinx's PYNQ development environment to increase the prototyping speeds of FPGA-based video processing systems.



Figure 2.8: Typical FPGA development life cycle. *An FPGA design starts with HDL code, then is synthesized into component netlist, any design constraints are added to the modeling, the physical layout of the netlist components are determined in floor planning, the place and route step then optimizes interconnections between gate level components while maintaining timing constraints, and finally a bitstream is generated which provides instructions for how the fabric is to be configured to implement the designed circuit.*

Function	Width	Direction	AXI4-Stream Signal Name	Video Specific Name
Video Data	Any number of bytes	In	s_axis_video_tdata	DATA
Valid	1	In	s_axis_video_tvalid	VALID
Ready	1	Out	s_axis_video_tready	READY
Start Of Frame	1	In	s_axis_video_tuser	SOF
End Of Line	1	In	s_axis_video_tlast	EOL

Figure 2.9: Xilinx AXI4-Stream table of slave interface signals [6]. *There are five signals used in the slave interface of the Xilinx AXI4 IP.*

Function	Width	Direction	AXI4-Stream Signal Name	Video Specific Name
Video Data	Any number of bytes	Out	m_axis_video_tdata	DATA
Valid	1	Out	m_axis_video_tvalid	VALID
Ready	1	In	m_axis_video_tready	READY
Start Of Frame	1	Out	m_axis_video_tuser	SOF
End Of Line	1	Out	m_axis_video_tlast	EOL

Figure 2.10: Xilinx AXI4-Stream table of master interface signals [6]. *There are five signals used in the master interface of the Xilinx AXI4 IP.*

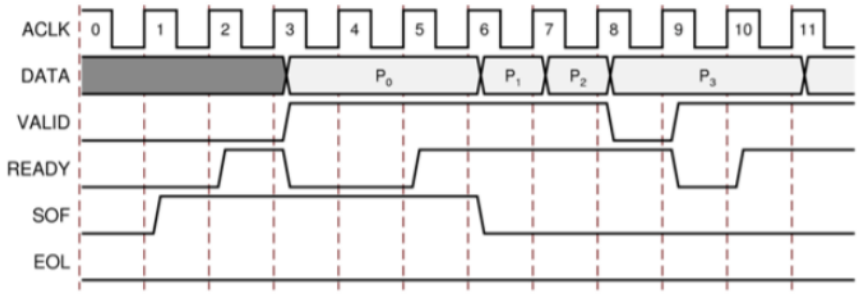


Figure 2.11: Xilinx AXI4-Stream handshaking [6]. *The AXI4-Stream interface uses a ready/valid handshaking protocol to flag the mark of a new frame being transmitted.*

2.3 Review of FPGA Implementations of Deep Learning Networks

This section provides an overview of previous works concerned with the implementation of deep learning models into FPGAs. First we provide a quick overview of modern deep learning and its conventional implementation in GPU environments. Then we outline a survey of recent implementations of deep learning systems in FPGA hardware.

2.3.1 Deep Learning Overview

Deep learning is a sub-field of machine learning which is itself a research area under the broader umbrella of artificial intelligence [7] as illustrated in figure 2.12. Deep learning methods utilize neural network architectures, in combination with very large training example databases, to create models that can accurately discriminate the features of their inputs to detect, classify, and segment a problem space. The first neural network based method was proposed in 1958 with the design of the perceptron [36] by Frank Rosenblatt. The perception is essentially a linear classifier based upon a binary interpretation of a biological neuron. Ultimately, the perceptron was abandoned by the research community due to its inability to converge to a solution for the XOR function and neural network based methods were mostly forgotten for decades. During this time period, several other techniques in artificial intelligence gained and lost prominence in the research community during which two AI "winters" were experienced. These winters reference time periods where the hyped promise of AI fell short and thus the majority of government and commercial funding was cut from its researchers.

Several years later, the neural network made a surprise comeback by being the top performer in the 2012 ImageNet large scale image recognition challenge. AlexNet [21] achieved the lowest top 5 error rate by 10.8 points from the next competitor. This performance marked a new boom in AI research with deep learning methods shown as a highly viable solution to many challenging vision tasks. In subsequent years, other deep learning neural networks would continually push the envelope on performance [8] as seen in figure 2.13.

The methods and architectures of deep learning are varied and tailored to solve specific problems. Popular variants include the Recurrent Neural Network (RNN), Generative Adversarial Networks (GAN), and the Convolutional Neural Network (CNN). Recurrent Neural Networks (RNN) are particularly suited for handling sequential data since their structure considers data in the past in relation to the present and future. Generative Adversarial Networks (GAN) are neural networks made up from two networks in competition with each other: a generator and a discriminator. The generator produces a synthetic instance of a class and the discriminator actively tries to distinguish synthetic examples from genuine. Through training the generator learns features to accurately fool the discriminator producing realistic instances of a class. Lastly, Convolutional Neural Networks (CNN) have been shown to be highly effective at vision tasks. CNNs are the de-facto choice for tasks such as object detection, classification and segmentation.

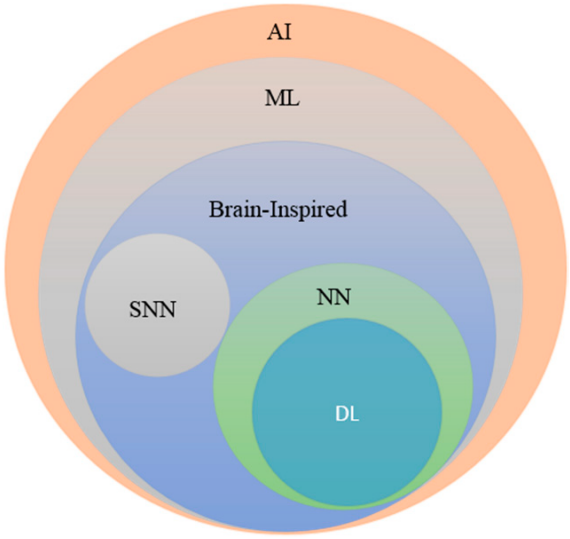


Figure 2.12: Areas of artificial intelligence [7]. *Deep learning is a specific sub-branch of machine learning techniques.*

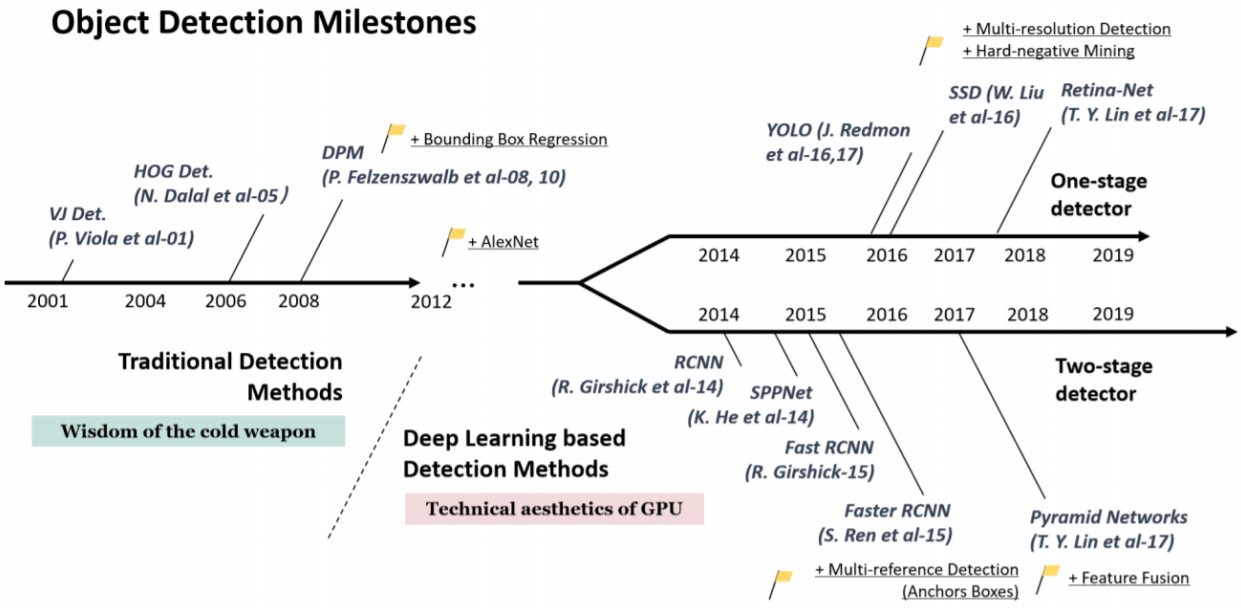


Figure 2.13: Recent milestones in object detection deep learning architectures [8]. *Traditional methods led in performance until the introduction of AlexNet in 2012. Current deep learning methods can be broadly categorized as either one-stage or two-stage detectors.*

Since the ImageNet 2012 challenge, CNNs have been instrumental in modern research for several computer vision problems. As seen in figure 2.14 the CNN is composed of several layer types (including the raw input layer) some of which are standard in all of the CNN variants (convolution, activation, pooling, and full connected layers) [9]. The convolution layer uses the mathematical operation of convolution to detect features in the previous layer. Convolution adds the values of a m x m region of an image which are weighted by a learnable m x m matrix (called a kernel). The kernel is centered on one pixel and this pixel's value is replaced by the result of the convolution. Convolution can be used to find edges or other high order semantics. This result is often called an activation map. Activation layers utilize an activation function which applies some threshold to see if the activation map's values should be passed on through the network. For CNNs, the current activation layer of choice the rectified linear unit (ReLU). The ReLU fucntion can be seen in Equation 2.1. This function sets negative inputs to zero and doesn't allow a response. The ReLU activation layer has became popular due to the fact that it only allows a small percentage of neurons to be active at any one time reducing computational complexity as well as converging quicker than older methods such as the sigmoid activation function. Pooling layers reduce the spatial size of the network. This reduction can help with the problem of overfitting to the training data. Pooling also reduces the number of parameters in an architecture which increases both training time and inference speed. Max pooling is a popular scheme in which a n x n window slides across the previous layer. The value in the window, which has the maximum value, is then used to represent the response and is the only value passed to the next layer. Fully-connected layers are located at the end of the network. As the name suggests, all neurons in these layers are connected to all neurons in the previous layer. Fully-connected layers learn to discriminate high-level features into the certain classes that the CNN has been trained to identify. This is accomplished via the softmax technique which outputs a confidence probability for the assigned class.

$$f(x) = max(0, x) \tag{2.1}$$

CNNs have been shown to be highly effective at handling image-based problems. In particular, CNNs are used in the domains of general image classification, object detection, scene segmentation, facial recognition, and more. However, the particular architecture of the CNN used is not the only factor that determines its importance. For CNNs to optimize their problem space performance they require enormous amounts of sample images in which to train their weights on. There are several of these datasets of images all organized to work for a particular problem such as image classification or facial recognition. Furthermore a dataset is typically separated into three subsets: training, validation and testing [37]. As the name implies, the training set is used to configure the learnable weights in the CNN. This is achieved via the assigned loss function where loss values are captured during the forward pass of an input through the network and those values updated to the weights by backpropagation. The validation set is used during the training process to optimize the user specified hyperparameters of the network. These hyperparameters include the drop out rate and learning rate of the network. Finally, the testing set is used once the network model training is complete to evaluate the performance of the final trained model.

There are several open source datasets available for the production of deep learning models on computer vision tasks. The MS-COCO dataset has over 330,000 images and is geared for object detection, segmentation and image captioning. CIFAR-10 is an image classification dataset with 60,000 images representing 10 classes of images. Given that the goal of this thesis work was to evaluate both image classification and object detection we selected two datasets to train our models, namely: ImageNet-2012 and VOC2007. The ImageNet-2012 dataset is comprised of approximately 1.2 million images representing 1,000 possible classes [38]. For the object detection evaluation we selected the VOC2007 training dataset consisting of 9,963 images with 24,640 annotated objects representing 20 possible classes [39].

Beyond the design of the CNN architecture itself and the availability of large scale datasets, the turning point for the CNN to become wildly influential in the computer vision scene was in the increased performance and availability of the underlying hardware, namely the GPU architecture, that could handle the high computational constraints needed. The first GPU was developed by Nvidia in 1999 and was designed to render computer graphics at faster rates for the demands of high-end video games. Eight years later in 2007 Nvidia released the CUDA (Compute Unified Device Architecture) development platform that provided programmers an API to leverage the parallel computational power of the GPU for general purpose tasks. However, it wasn't until 2009 when the first paper was published that outlined the use of the GPU for deep belief networks [22]. It was therefore not only the theoretical design breakthrough gained with AlexNet but also similar timing of the new hardware backbone of the GPU that allowed deep learning to become the current state-of-the-art in image classification tasks.

When leveraging a GPU for deep learning it is important to distinguish between two stages of development: training and inference. Training is when the the network processes vast numbers of examples which allow the model weights to be adjusted iteratively to fit the designed learning parameters and create a statistical model that represents the training data. For deep learning networks the larger the dataset the greater the potential for higher performance - unlike traditional machine learning techniques which converge with moderate amounts of examples. Inference is when a novel image is presented to a network and scored based on the model weights from training.

Although GPUs in particular have became the de facto method in the training and inference stages of deep learning models they are not particularly suited for operational use cases when systems must be designed to conform to low SWaP constraints. GPUs require a large power footprint as well as an attached CPU and associated software to control training and inference. The additional control pipeline further adds to the power demands of the GPU. Such a large system also does not lend itself to deployed applications such as drones or autonomous vehicles. Thus, such use cases of deep learning inference require systems with far smaller footprints that can still achieve similar performance which are often implemented in more power efficient and smaller CPUs but at the loss of the inference performance gained by GPU use.

2.3.2 Previous Work in FPGA-Based Deep Learning

As laid out in the previous section, GPU-based hardware is the predominant means for deep learning implementation. However, as noted in recent surveys such as [40] and [41], interest in FPGA-based solutions is growing in the research community due to energy efficiency and the low SWaP footprint of FPGAs. Furthermore, as indicated in [42], [43] and [44], although most current FPGA-based CNNs in literature focus strictly on model performance in hardware, it would be ideal to extend FPGA-based solutions to include video streaming given their optimal characteristics for Edge deployment.

Given that FPGAs provide much better performance with fixed point operations over floating point [45], most work in the implementation of deep learning models in FPGA have focused on effective model weight quantization. For instance, in [46] CNN model weight are quantized to form binarized neural networks which utilize 2-bit model weights. Although this extreme weight quantization did allow the authors to deploy a CNN in a small FPGA it could not achieve comparable model performance to the original floating point model in GPU. A similar project, FINN [47] was another attempt to leverage binary networks for FPGA deployment of deep learning models. This framework provided very fast image classification throughput speeds on the FPGA albeit with lower model performance.

Ternary weight models (3-bit), have been proposed in [48] and [49] to increase performance of their binary counterpart while still achieving implementation into smaller FPGAs. Although ternary do achieve greater performance than binary networks, and with more energy efficiency, they still fall short of the SOTA model performances achieved by the original floating point models. Another novel quantization method from [50] proposed an efficient 4-bit weight quantization by using a significantly smaller number of sampled weights than the number of original weights. However, this strategy had similar results to its ternary model counterpart.

The fpgaConvNet project [51] provides an altogether different approach from the previous methods in an attempt to provide a framework that automatically maps a floating point model to an FPGA. This framework achieves this by ingesting a high-level representation of a CNN model and performing SDF graph partitioning essentially representing the CNN as a directed acyclic graph (DAG) where a node maps to a CNN layer. The framework then processes this graph to optimize FPGA resource allocation. The framework produces custom fixed point weights for the targeted CNN. The fpgaConvNet framework was able to achieve very high performance rates while greatly improving the energy efficiency of the FPGA implementation over the previously mentioned methods. Although this framework shows the potential in FPGA-based CNN deployment its drawback is the difficulty of representing the candidate CNN into complex mathematical structures requiring several difficult steps which further extend the prototyping process.

An improvement over fpgaConvNet is the FP-DNN project [52] which attempts to provide a full end-to-end framework which automates the process of mapping a deep learning model into FPGA fabric. The FP-DNN framework, unlike fpgaConvNet, uses a model description from TensorFlow as the starting point for implementation. Here FP-DNN uses a hybrid RTL-HLS library to compile the input description and then leverages a graph coloring algorithm to map the FPGA resources. FP-DNN provides similar performance results as fpgaConvNet but with a more user-friendly HLS style program flow.

Taking inspiration from the fpgaConvNet and FP-DNN frameworks, there have arisen several attempts at fully automated frameworks for mapping floating point deep learning models into FPGAs. The survey in [53] lays out additional tools such as DeepBurning [54], Caffeine [55] and Snowflake [56] as other potential FPGA deep learning frameworks. Although these frameworks all provide important advances in the area of FPGA CNN implementation, given their nature as research-grade tools we have found them to be extremely difficult to both install and use. However, motivated by these frameworks as well as increasing interest from the research community, Xilinx has began offering production grade tools to assist in the implementation of deep learning models into their line of FPGAs.

As with the other frameworks, the Xilinx tools leverage weight quantization to reduce the size of the deep learning models as well as improve processing performance in FPGA fabric. In particular, 8-bit integer representation of model weights was found to strike the balance between optimal model performance and power efficiency [57]. Besides lower power consumption during computation, the 8-bit quantized weights also reduce the power needed to perform memory transfers since the data is represented with far lower bit-width.

Xilinx provides a tool to quantize floating point models into 8-bit representations via its DNNDK (Deep Neural Network Development Kit) framework [12]. The DNNDK framework is itself a set of tools that take as an input a fully trained deep learning model from one of several popular libraries (TensorFlow, Caffe, etc.), performs several takes such as pruning and weight quantization, and ultimately compiles the new quantized model into an HLS description of a specialized IP core called a Deep-Learning Processor Unit (DPU). The DPU IP core can then be incorporated into a larger FPGA hardware pipeline such as those designed in this thesis. Being a production grade tool, DNNDK benefits from both ease of install and use as well as a dedicated support community. Its principal drawback is that it is only currently supported by the Zynq-7000 and Zynq Ultrascale+ series of Xilinx FPGAs. In an effort to promote the use of the DNNDK framework, Xilinx has developed it's "AI Model Zoo" [58] which contains various tutorials, example models and other support which was leveraged by the work in this thesis.

2.4 Impact of Image Quality on CNN Performance

This final section of the literature review is concerned with the impact and mitigation of image quality on CNN performance. First we review how CNN performance is degraded from poor image quality. Next, we explore techniques to improve noisey images. Finally, we explore quantitative metrics to rate the quality of an image.

2.4.1 Image Degradation and CNN Performance

Image and video quality has traditionally been concerned with providing the best experience for a human consumer. However, with the recent utilization of deep learning methods for computer vision tasks there is an emerging body of research related to how image quality impacts automated classification performance. Additionally, CNNs are trained on datasets where the intention is to provide well suited examples, therefore images are likely to be well composed and free of noise. However, in real-world imaging environments it cannot be

assumed that input images will be of high quality [59]. Regardless of the application of the CNN network, the quality of the input image will invariably influence the performance of the resulting classification. Common quality issues stem from illumination variance (either poor illumination or over saturation), occulsion of the object, object pose, compression artifacts and noise from the image acquisition device [60]. When utilizing standard classification datasets CNNs are on par with humans, however, when the same datasets are corrupted with various noises the humans still maintain their high levels of accuracy while CNN performance quickly drops off [61]. Therefore, for an Edge deployed system such as an autonomous vehicle, it is important to understand the impact of image degradation on overall performance when human life is at stake.

Work in [62] examines the impact of blur on input images to the Inception network and proposes a new joint architecture in an attempt to offset the effects of blurring. The work of [63] also specifically considers the blurring degradation and suggests that the best method to overcome blur is to ensure that the model learns to generate blur invariant representations in its hidden layers. The authors in [64] also look at the impact of blur on image classifiers and propose the best mitigation is through fine-tuning a model on noisy images that reflect the anticipated degradation. Finally, the work in [65] benchmarks several image classifiers against 15 types of noise and proposes that robustness to noise should be a principal concern for research given that state-of-the-art CNNs already achieve high accuracy on high quality images.

Work from another IRIS lab graduate student, Matthew Seals, explored the robustness of object detectors (specifically Faster RCNN) on a wide array of image corruptions [20]. This work found that certain degradations caused more severe inference performance than others and thus sorted degradations into high, medium and low impact. In particular, brightness and contrast were found to be consistently most degrading to performance.

2.4.2 Corrective Measures for Noise Reduction

Although several works have explored the drop in CNN performance when confronted with degraded images fewer have suggested means to mitigate these effects. The work in [66] explores this impact of degradation upon CNN performance and proposes a novel dual-channel model to overcome the issues. The first channel in this model takes an unprocessed input through a pre-trained network and a second version of the input is then processed and sent through a a version of the model that is fine-tuned to a particular noise. The result of these two channels is then fused to create the final class score. The authors in [67] take a different approach by instead applying an adversarial training approach to an image classifier in an attempt to make it more robust. They achieve this adversarial training by leveraging additive noise to fine-tune the CNN model to a variety of noises.

Both of these strategies suggest altering the model itself in a manner to overcome the noise although in standard imaging pipelines a pre-processing step is conventional immediately after acquiring an image from a sensor. This pre-processing step is essential when the imaging sensor is deployed in an unconstrained environment where variations in illumination and sensor noise are highly common. Numerous techniques from the image processing domain already exist that are designed to overcome this common imaging noises to provide optimal image quality. We propose in this thesis to use these image processing techniques as part of

the pre-processing step in our full system pipeline to provide robustness to common in-the-wild imaging noises.

2.4.3 Quantitative Image Quality Metrics

Several image quality analysis (IQA) metrics exist that propose to assign a quantitative score based on the quality of an image. Additionally, these metrics are typically organized into two categories: (1) full-reference metrics, and (2) no-reference based (aka blind) methods. As the name indicates, full-reference metrics require an original unaltered image in which to establish a baseline to determine relative quality on a somehow altered variant image. No-reference metrics rely on more complex models in which to assign a quality based on only data presented in the image under investigation. For this thesis we focused on full-reference metrics since all experimental evaluations operate on known datasets. These full-reference metrics are explored below.

The Peak Signal to Noise Ratio (PSNR) provides a measure of the ratio between the optimal signal (or image in our case) to the noise introduced into the signal. The PSNR is typically represented by a logarithmic decibel scale. PSNR leverages the concept of the mean squared error (MSE) as defined in equation 2.2, where I is an ideal image and K is the same image with some noise introduced. Using MSE, the PSNR can then be defined as in equation 2.3 where MAX is the maximum pixel value (255 in 8-bit images).

$$MSE = \frac{1}{mn} \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} [I(i,j) - K(i,j)]^2 \tag{2.2}$$

$$PSNR = 20log_{10} \frac{MAX}{\sqrt{MSE}} \tag{2.3}$$

However, after several experiments we concluded that the PSNR, although a standard noise quantitative method, did not provide the level of granularity we desired in evaluating these techniques. Therefore, we adopted another image quality assessment metric, Structural Similarity Index (SSIM) [68] to visualize and report the impacts of degradation and correction. The SSIM is a full-reference technique which compares an original image (assumed to be of perfect quality) to an associated degraded image to determine the degree of degradation. Given that our experiment is conducted on known datasets, we felt this metric would provide the greatest degree of confidence in quantifying image quality.

The SSIM metric was designed to be an improvement over the PSNR score. SSIM measures the differences of luminance, contrast and structure between the two frames and reports a quality score in the range of 0 to 1 (where 1 is a perfect quality). The SSIM equation is defined as in equation 2.4.

$$SSIM(x,y) = \frac{(2\mu_x\mu_y + c_1)(2\sigma_{xy} + c_2)}{({\mu_x}^2 + {\mu_y}^2 + c_1)({\sigma_x}^2 + {\sigma_y}^2 + c_2)} \tag{2.4}$$

SSIM is calculated over various corresponding sets of two windows, x and y, of size NxN between the two frames. Here μ_x represents the average of x, μ_y represents the average

of y, ${\sigma_x}^2$ represents the variance of x, ${\sigma_y}^2$ represents the variance of y, σ_{xy} represents the covariance of x and y, and c_1 and c_2 are two variables used to stabilize the denominator.

The luminance l(x,y) between the two frames is based on the component of the SSIM equation represented in equation 2.5. The contrast c(x,y) between the two frames is based on the component of the SSIM equation represented in equation 2.6. Finally, the structure s(x,y) between the two frames is based on the component of the SSIM equation represented in equation 2.7, where $c_3 = c_2 \backslash 2$.

$$l(x,y) = \frac{2\mu_x\mu_y + c_1}{{\mu_x}^2 + {\mu_y}^2 + c_1} \tag{2.5}$$

$$c(x,y) = \frac{2\sigma_x\sigma_y + c_2}{{\sigma_x}^2 + {\sigma_y}^2 + c_2} \tag{2.6}$$

$$s(x,y) = \frac{\sigma_{xy} + c_3}{\sigma_x\sigma_y + c_3} \tag{2.7}$$

The SSIM score therefore provides a thorough metric for comparison between an unaltered image and a duplicate image containing some sort of alteration. Given that in this thesis we are using two known datasets as our benchmarking baseline and then applying our own degradations and corrections, the use of the full-reference IQA metric of SSIM was determined to be a logical choice for quantitative metrics in our experiments. Such a metric can provide insight into the connection between image quality and CNN inference performance.

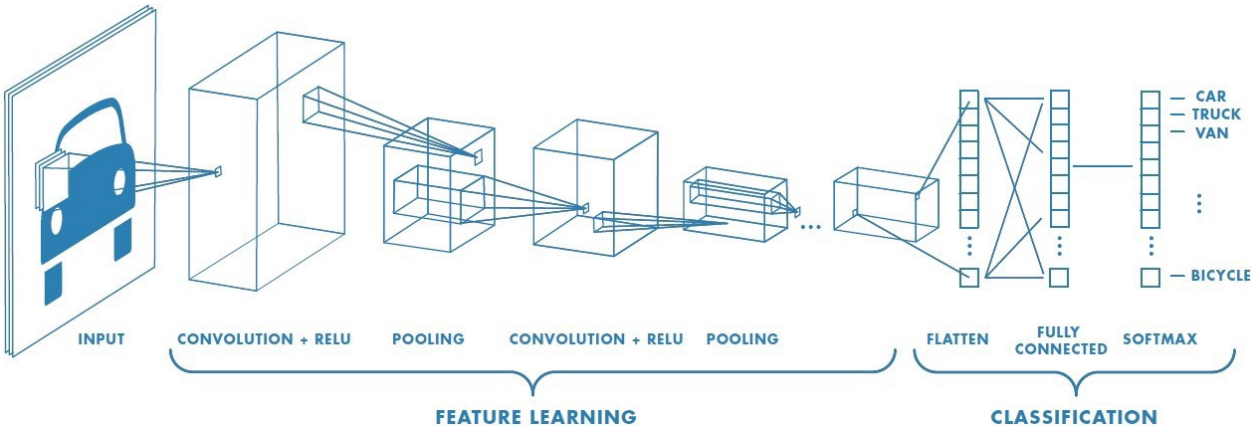


Figure 2.14: Standard CNN architecture representing the standard layers [9]. *The feature learning portion of the network is shown on the left and the classification layers on the right.*

Chapter 3

Video Processing on FPGAs

As laid out in the introduction, this chapter covers the initial contribution of this thesis which was the development of an FPGA-based video processing pipeline. In particular, this pipeline was iteratively developed to target different FPGA boards with varying degrees of functionality over the course of the thesis work. These various design iterations are detailed in the first section of this chapter and are summarized in table 3.1. The following section of this chapter summarizes our associated library of video processing IP cores that could be placed in this pipeline to manipulate the video stream. The code for these IP cores was recompiled as needed to target the different FPGA boards used in the pipeline iterations. As mentioned in our contributions, this base FPGA design was then extended as needed to provide the framework for the experiments in our other contributions as laid out in subsequent chapters. Our final FPGA architecture achieves 30 frames-per-second processing on video streams with 1920x1080 resolution.

3.1 FPGA Video Pipeline Architectures Developed

The implementation of our video processing pipeline was performed as iterative steps targeting 3 boards with 4 versions of the implementation. The original system was from the previous work of an IRIS lab student targeting the Zybo FPGA [17]. The next two designs were created targeting the Z1 FPGA board: the first leveraged Xilinx's PYNQ framework with an extension to revise the design to include our custom video processing cores and the second was a complete custom FPGA design with minimal hardware to maximize performance. Finally, the last pipeline targeted the ZCU104 Ultrascale+ board and was developed with a mixture of the Xilinx reVision software along with inspiration from our custom Z1 design.

The first video design provided both the framework in which to begin our library of video processing IP cores as well as provided inspiration for extensions to include more high-speed processing. The thesis work from [17] targeted the older Xilinx Zybo Zynq-7000 board. This pipeline consumed a video file stored on a SD card, provided side-by-side processing of software based functions on the PS and hardware based functions on the PL, and displayed the two processed video streams simultaneously over a VGA video connection. This pipeline

is primarily mentioned here due to its great influence upon the subsequent iterations of our thesis video processing pipeline.

The motivation to move from the previous Zybo FPGA-based pipeline to an implementation on the Z1 FPGA was two-fold. First, the Z1 board has a larger Zynq fabric allowing for larger and more complex systems given that it has approximately triple the capacity of system resources compared to the Zybo as seen in table 2.2. Secondly, the Z1 board provides both dedicated HDMI in and out connections that can handle resolutions of up to 1080p. These two dedicated connections allow for streaming video in real-time from a source unlike the Zybo board that could either capture an input video and process later or process video from a file for the output, but not achieve true video streaming.

The first video pipeline we designed for the Z1 leverage Xilinx's PYNQ project as a starting point [33]. The PYNQ project is designed to allow software developers easier access to FPGA prototyping without requiring the knowledge of hardware design tools. Similar to the previous Zybo project, PYNQ runs a master Python overlay system on the PS of the Zynq fabric while allowing for interfacing to hardware modules on the PL. The PYNQ project is open-source by nature and can be used to instantiate several custom hardware designs or leverage the Xilinx official PYNQ system design. For this implementation of the pipeline we used the Xilinx PYNQ version 2.4 design and extended the hardware architecture to accommodate our custom video processing IP cores via the Xilinx Vivado and HLS tools. We then updated the PYNQ PS Python control scripts to reference our new hardware designs. Although the PYNQ system did provide HDMI in/out full streaming video processing capabilities at 1080p/30fps, the underlying PYNQ architecture consumed nearly all of the FPGA resources. Therefore, we realized that to extend a system to include CNN IP cores would require a different approach.

To overcome the resource hungry architecture of the PYNQ framework we next decided to implement a completely custom video processing system targeting the Z1. To minimize resources used, and to accommodate adding CNN hardware cores later, our design was targeted to be "bare metal" and only use the PS portion of the Zynq fabric to instantiate the hardware at power on. This custom hardware design can be seen in the system block layout from the Vivado tool in figure 3.1. Here we have highlighted the 9 principal hardware IP cores leveraged in the video processing pipeline.

Block 1 in figure 3.1 is the TMDS port mapping to the physical HDMI In connector on the Z1 board. Block 2 is a "DVI to RGB Decoder" (see left IP in figure 3.2) IP that converts the HDMI TMDS signal into a 24-bit RGB bus with associated vertical and horizontal synchronization signals to be consumed by the system. The PixelClk output signal drives the Video Timing Controller block for system syncing. The RefClk input signal is determined by FPGA clk. The "Video In to AXI4-Stream" IP core in block 3 of figure 3.1 converts the RGB signal into the AXI4-Stream protocol for use in both HLS cores and VDMAs, and is needed to create the required handshaking signals to sync communication. The AXI4-Stream protocol is the communication protocol used by all internal system components. This IP can be observed on the right of figure 3.2.

Block 4 of figure 3.1 is a placeholder that can be filled with any of our custom video processing cores from our library. This core is implemented in the hardware chain in-between the Video In to AXI4-Stream core and the Video Direct Memory Access core. In figure 3.3,

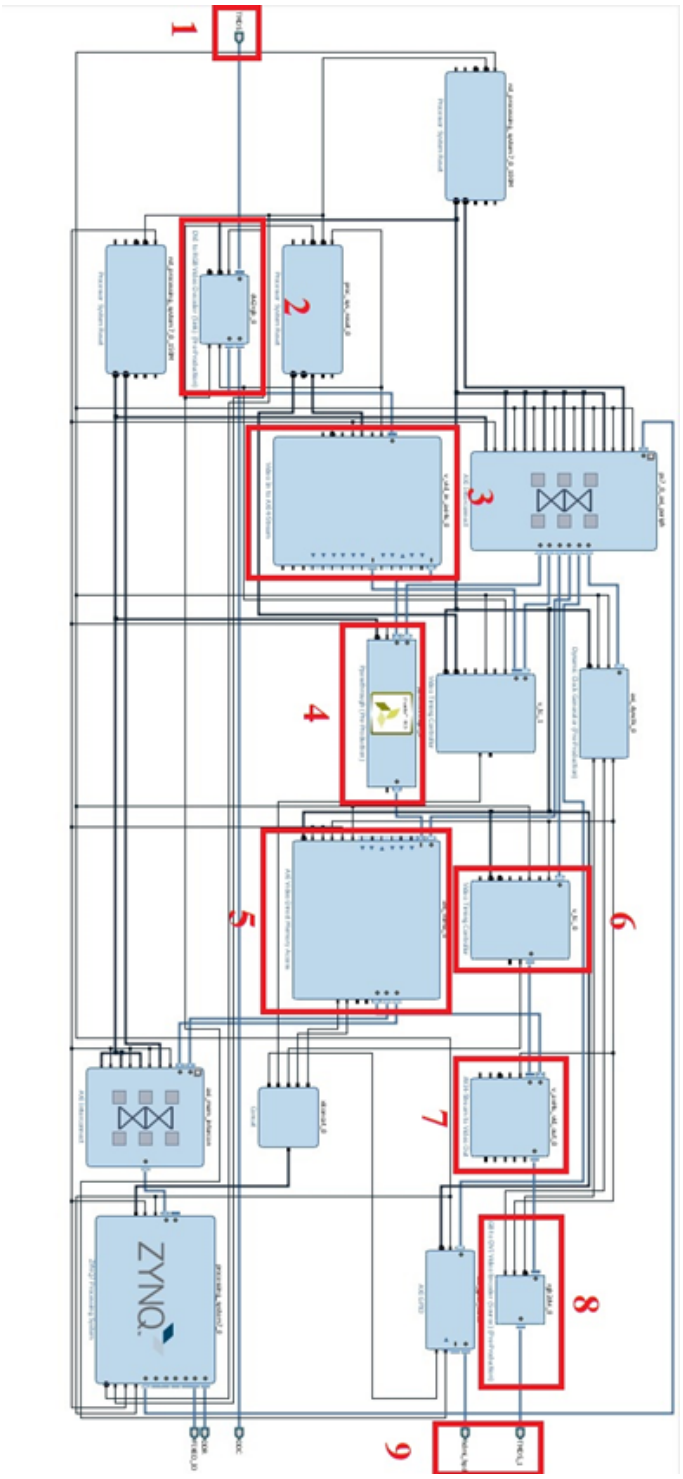


Figure 3.1: Custom video processing pipeline targeting Z1 FPGA board. *The 9 principal IP cores associated with the video processing pipeline are outlined in red. These IP cores are detailed in other figures and text in this section. Of note is block 4 which is where video processing cores from our custom library are implemented.*

Table 3.1: Iterations of video processing systems. *Over the course of the thesis work, four iterations of the video processing pipeline were developed which targeted three different FPGA development boards.*

FPGA Board	Existing Hardware	PYNQ Overlay	Custom Hardware	Xilinx SDX + reVision
Zybo	✓			
Z1		✓	✓	
ZCU104 Ultrascale+		✓	✓	✓

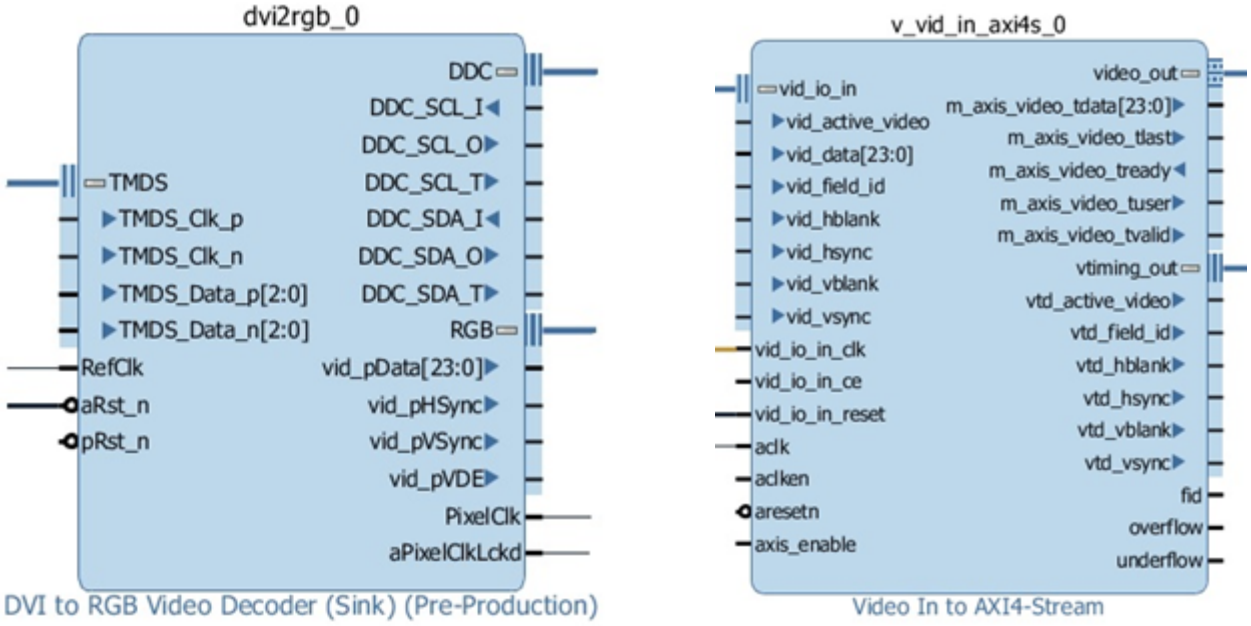


Figure 3.2: DVI to RGB Decoder and Video In to AXI4-Stream IP cores. *The DVI to RGB Decoder IP core (left) converts the HDMI TMDS signal into a 24-bit RGB signal to be consumed downstream in the video processing pipeline. The Video In to AXI4-Stream IP core (right) converts the RGB signal in AXI4-Stream format which is leveraged by the internal IP cores within the video processing pipeline.*

the passthrough IP core is shown for illustrative purposes. Most notable is the TDATA line that has been modified for a 24-bit data width to work with the AXI4 protocol.

Block 5 in figure 3.1 is an AXI VDMA core. The VDMA is for reading/writing an AXI4-Stream frame into memory mapped DDR on the FPGA allowing frame storage without requiring it to be processed first by the PS (ARM Core). The core itself can be seen on the left of figure 3.4. Block 6 in figure 3.1 is the Video Timing Controller (VTC). The VTC core's vtiming_in signal is synced with the Video In to AXI4-Stream and is used to sync the AXI4-Stream to Video Out IP ensuring proper timing and horizontal/vertical syncing in the system. This VTC core can be seen on the right of figure 3.4.

Block 7 in figure 3.1 is an AXI4-Stream to Video Output IP core. This core converts the AXI4-Stream signal back into a 24-bit RGB bus carrying the associated vertical and horizontal synchronization signals needed for output display (see left IP in figure 3.5). Block 8 in 3.1 is a RGB to DVI Encoder core (see right IP in figure 3.5) that encodes the 24-bit RGB signal, horizontal and vertical sync signals to HDMI TMDS for output display. Timing for this core is driven by the PixelClk and SerialClk (which are controlled by the Video Timing Controller IP). Block 9 in figure 3.1 is the TMDS port that represents the physical HDMI Out connector on the Z1 board along with the Hot Plug Detect (HPD) signal which finishes the flow of video frames through the custom pipeline.

Our Z1 custom video processing pipeline achieved 1080p resolution with nearly 30fps processing while consuming much fewer resources than the previous PYNQ design. However, we ultimately abandoned the Z1 board but maintained the inspiration from this design to our final iteration of the pipeline.

Our final pipeline design targeted the Xilinx ZCU104 Ultrascale+ FGPA board. As seen in table 2.2, this board has significantly more FPGA fabric resources, better PS specifications as well as far better video streaming capabilities. An additional feature is that this board is supported by Xilinx's reVision framework [69]. The reVision framework is a combination of Xilinx's advanced SDX design tool and HLS library allowing for larger-scale FPGA designs. For this final pipeline we were able to modify an existing base project from [69] to include some additional functionality inspired from our Z1 system to complete the final pipeline. Furthermore, as detailed in Chapter 4 of this thesis, the ZCU104 Ultrascale+ FPGA is also supported by Xilinx's DNNDK tool chain allowing our research to implement hardware-based CNNs for our other thesis experiments.

3.2 Library of Video Processing Cores Developed

The next section provides details of the library of custom video processing IP cores designed and validated to be used with any of the FPGA architectures detailed above. All of these IP cores were developed using the Vivado HLS tool which compiles C or C++ code into equivalent Verilog or VHDL for use on an FPGA chip. Additionally, the Vivado HLS Video library was used for both managing video signals into data structures and for some stream-based processing. Several cores also leverage pixel-access to create their desired results.

An important note here is that a portion of these IP cores were re-implemented for this thesis from previous IRIS lab work in [17]. Additionally, a portion of these IP cores were developed in connection with another IRIS lab graduate student (Samuel "Clay" Leach). In

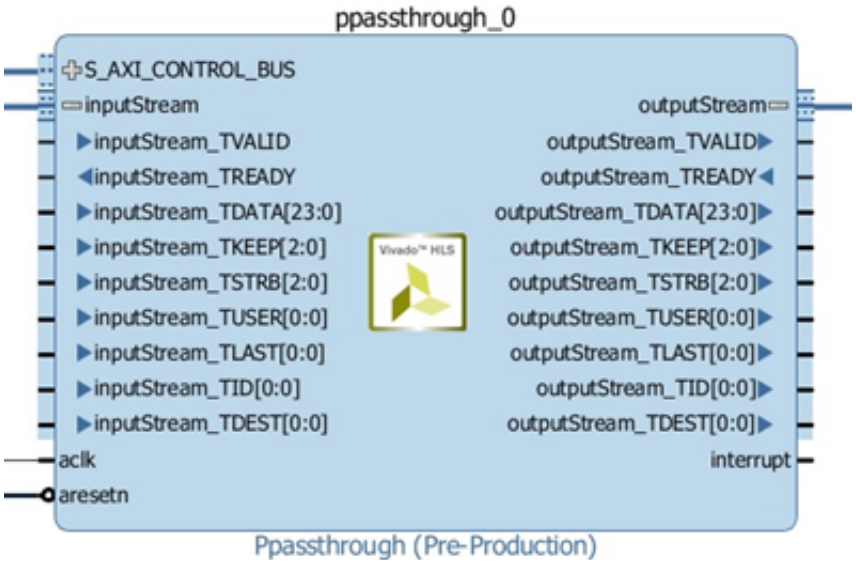


Figure 3.3: Custom HLS Video Processing IP core. *This IP core converts the RGB signal in AXI4-Stream format which is leveraged by the internal IP cores within the video processing pipeline.*

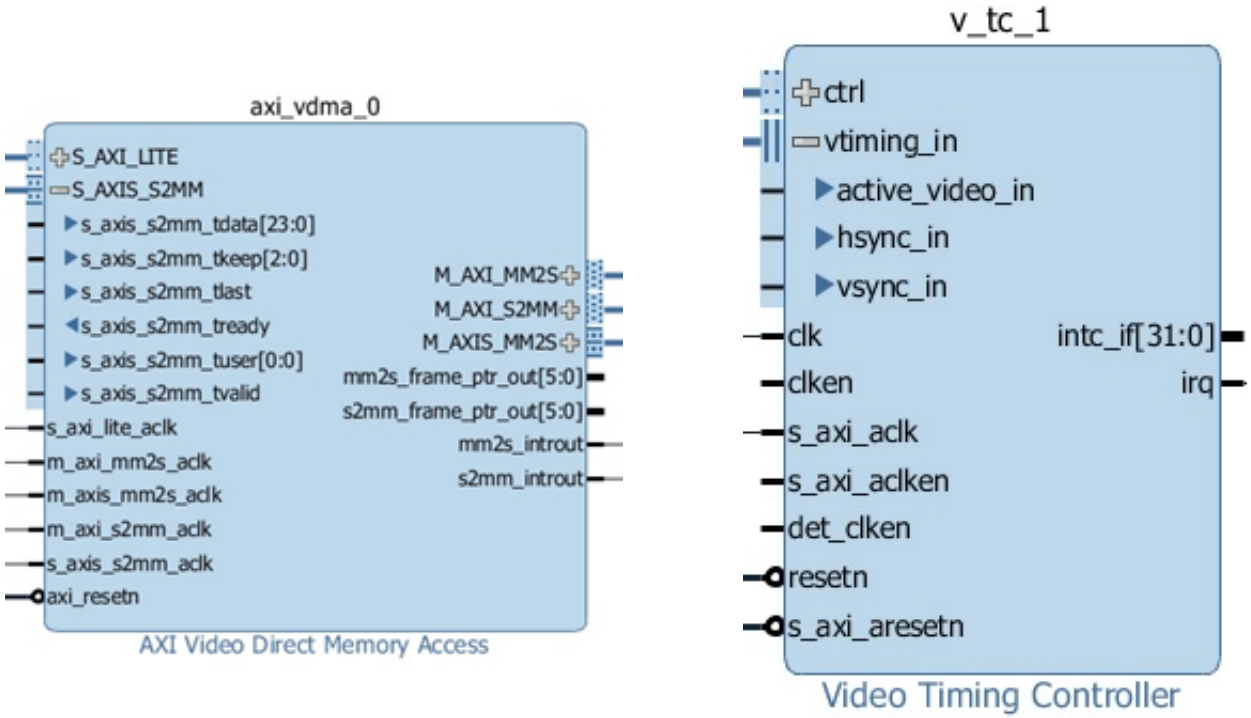


Figure 3.4: VDMA and Video Timing Controller IP cores. *The VDMA IP core (left) is used for managing video frames in memory while the Video Timing Controller IP core (right) ensures proper syncing of all video frames throughout the entire pipeline.*

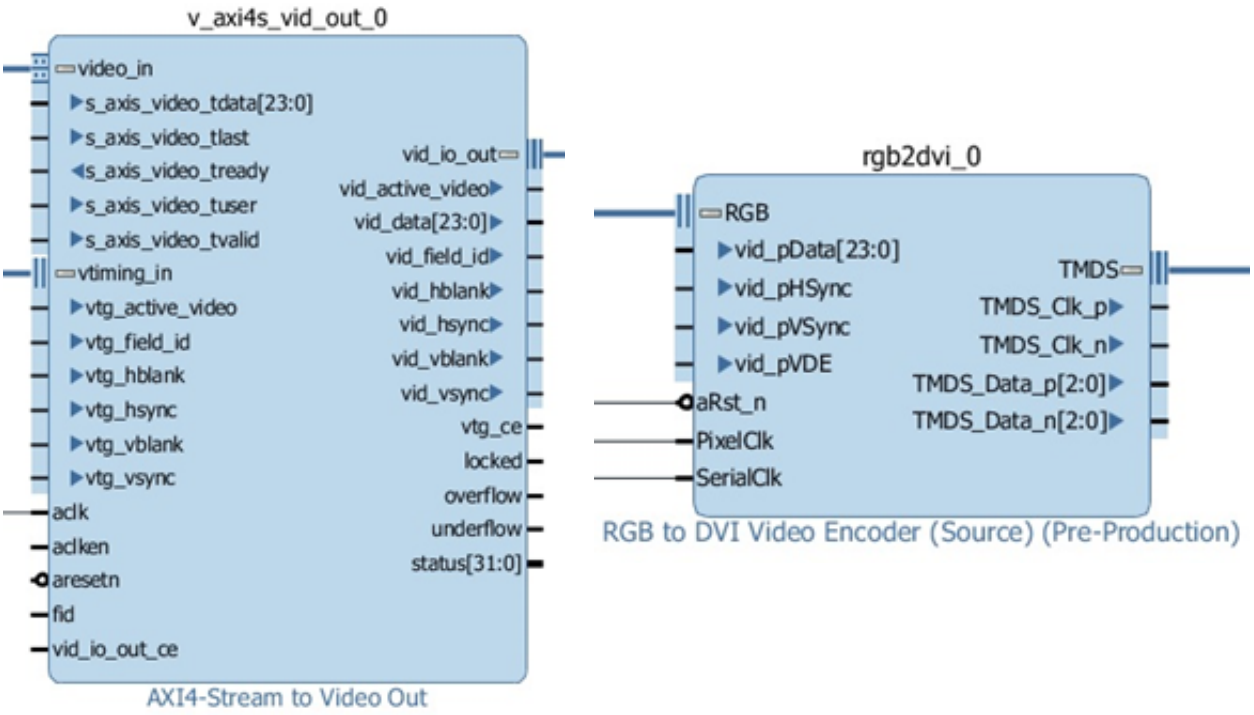


Figure 3.5: AXI4-Stream to Video Output and RGB to DVI Encoder IP cores. *The AXI4-Stream to Video Output IP core (left) is used for converting the AXI4-Stream data back to 24-bit RGB and the RGB to DVI Encoder IP core (right) encodes the RGB signal, plus syncing signals, to be ready for transmission via TMDS.*

table 3.2 is a detailed summary of the design and performance of a portion of these IP cores that we find to be most interesting.

Stream Passthrough IP Core

The initial IP core created for video processing purposes was a core to simply "passthrough" the input signal to the output stream. The first Passthrough IP core simply captured the HDMI input stream in the AXI stream video interface and then passed unaltered to the HDMI output stream. This core demonstrates, and provides a debugging check, that a single frame from the input source can be passed unaltered onto the output sink. The flowchart depicting this process for the IP core can been seen in figure 3.6.

After completing the design flow within Vivado HLS a Stream Passthrough IP core was created that could be implemented within a video pipeline architecture. The top-level block diagram of this IP core appears in figure 3.7. The Vivado HLS process also provides diagnostic and implementation summaries to catalog the particular implementation of a HLS created IP core. Of particular interest are the timing/latency summaries in figure 3.8 and the hardware utilization summary in figure 3.9. A comparison of hardware utilization across the different boards appears in table 3.3. It is worth noting that this hardware IP core has the lowest utilization of any in the custom library. This custom IP core was validated using a grayscale image, color image and via a color video sequence. These results are displayed in the figures 3.10 and 3.11.

Pixel Access Passthrough IP Core

An extension to the Stream Passthrough IP core is one that transmits the incoming video frame while exposing access to each pixel. This pixel access allows for more elaborate processing functions. Additionally, the additional hardware code consumes more fabric resources than its simple stream counterpart. The flowchart depicting this process for the IP core can been seen in figure 3.12.

After completing the design flow within Vivado HLS a Stream Passthrough IP core was created that could be implemented within a video pipeline architecture. The top-level block diagram of this IP core appears in figure 3.13. The Vivado HLS process also provides diagnostic and implementation summaries to catalog the particular implementation of a HLS created IP core. Of particular interest are the timing/latency summaries in figure 3.14 and the hardware utilization summary in figure 3.15. A comparison of hardware utilization across the different boards appears in table 3.4. This custom IP core was validated using a grayscale image and color image. These results are displayed in the figures 3.16 and 3.17.

Grayscale Conversion IP Core

Converting a color image to grayscale is an important pre-processing step for many other applications. As such, a custom IP core was created to perform this conversion quickly in hardware. The conversion is achieved by leveraging Xilinx's HLS video function CvtColor(). This function directly implements a grayscale conversion function as the linear function based on the values of the red, green and blue channels of a color pixel. The conversion function is seen in equation 3.1 below.

Table 3.2: List of implemented video processing IP cores and associated FPGA boards. *The majority of our custom IP cores have been implemented across all FPGA boards and custom video processing pipelines explored in this thesis.*

NO.	VIDEO IP	ZYBO	Z1	PYNQ SYSTEM (Z1)	ZCU104
1	Stream Passthrough	✓	✓	✓	✓
2	Pixel-Access Passthrough		✓	✓	✓
3	Grayscale Conversion	✓	✓	✓	✓
4	Sobel Edge Detector	✓	✓	✓	✓
5	Threshold	✓	✓	✓	✓
6	Sobel-then-Threshold	✓	✓	✓	✓
7	Dilation		✓	✓	✓
8	Erosion		✓	✓	✓
9	Inversion	✓	✓	✓	✓
10	Gaussian Blur	✓	✓	✓	✓
11	Sharpen	✓	✓	✓	✓
12	Pixel Operations (including contrast reduction)	✓	✓	✓	✓
13	Salt-and-Pepper Noise		✓	✓	✓
14	Kernel Convolution (including median filter)	✓	✓	✓	✓
15	Lookup Table (including gamma correction)	✓	✓	✓	✓
16	Splitscreen Display		✓	✓	
17	Doublescreen Display		✓	✓	



Figure 3.6: Stream Passthrough operation IP core flow chart. *The HDMI source signal is captured via the AXI interconnect and stored as an input frame within the Zynq PL. This frame is then placed back onto the AXI interconnect and then to HDMI sink.*

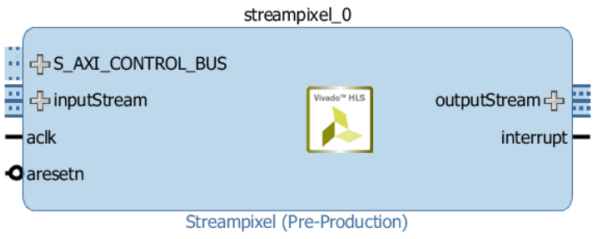


Figure 3.7: Stream Passthrough IP core block diagram.

Timing (ns)

Summary

Clock	Target	Estimated	Uncertainty
ap_clk	10.00	5.95	1.25

Latency (clock cycles)

Summary

Latency		Interval		
min	max	min	max	Type
4	2081163	4	2081164	dataflow

Figure 3.8: Stream Passthrough IP timing/latency.

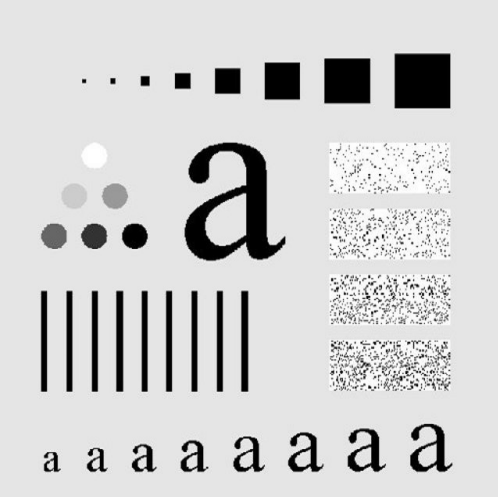
Summary

Name	BRAM_18K	DSP48E	FF	LUT
DSP	-	-	-	-
Expression	-	-	0	2
FIFO	0	-	30	120
Instance	-	1	365	404
Memory	-	-	-	-
Multiplexer	-	-	-	-
Register	-	-	-	-
Total	0	1	395	526

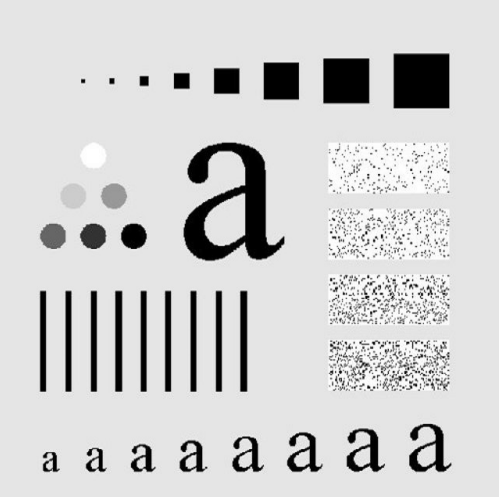
Figure 3.9: Stream Passthrough IP hardware utilization.

Table 3.3: Comparison of FPGA board resources utilization for Stream Passthrough IP core.

BOARD	LUTs	Flip-Flops	BRAM
Zybo	1.99%	1.12%	0.00%
Z1	0.99%	0.37%	0.00%
ZCU104 Ultrascale+	0.10%	0.01%	0.00%



(a) *Original grayscale image.*



(b) *Stream Passthrough processed.*

Figure 3.10: Stream Passthrough - grayscale image results (image source [10]).



(a) *Original color image.*



(b) *Stream Passthrough processed.*

Figure 3.11: Stream Passthrough - color image results (image source [10]).

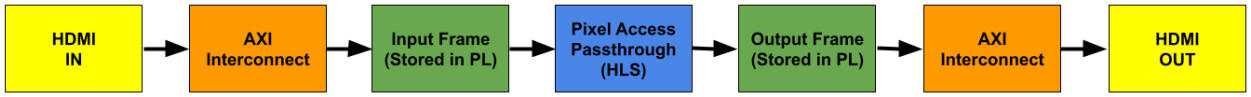


Figure 3.12: Pixel Access Passthrough operation IP core flow chart. *The HDMI source signal is captured via the AXI interconnect and stored as an input frame within the Zynq PL. This frame is then copied pixel-by-pixel into a second frame and then placed back onto the AXI interconnect and then to HDMI sink.*

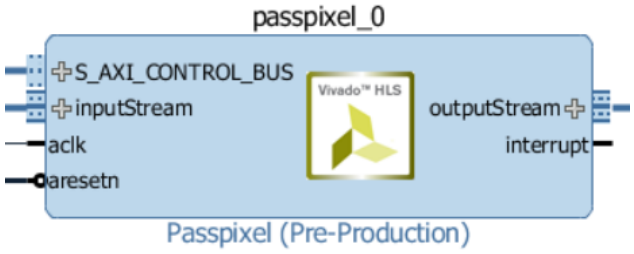


Figure 3.13: Pixel Access Passthrough IP core block diagram.

Timing (ns)

Summary

Clock	Target	Estimated	Uncertainty
ap_clk	10.00	8.75	1.25

Latency (clock cycles)

Summary

Latency		Interval		
min	max	min	max	Type
6	2081163	4	2081164	dataflow

Figure 3.14: Pixel Access Passthrough IP timing/latency.

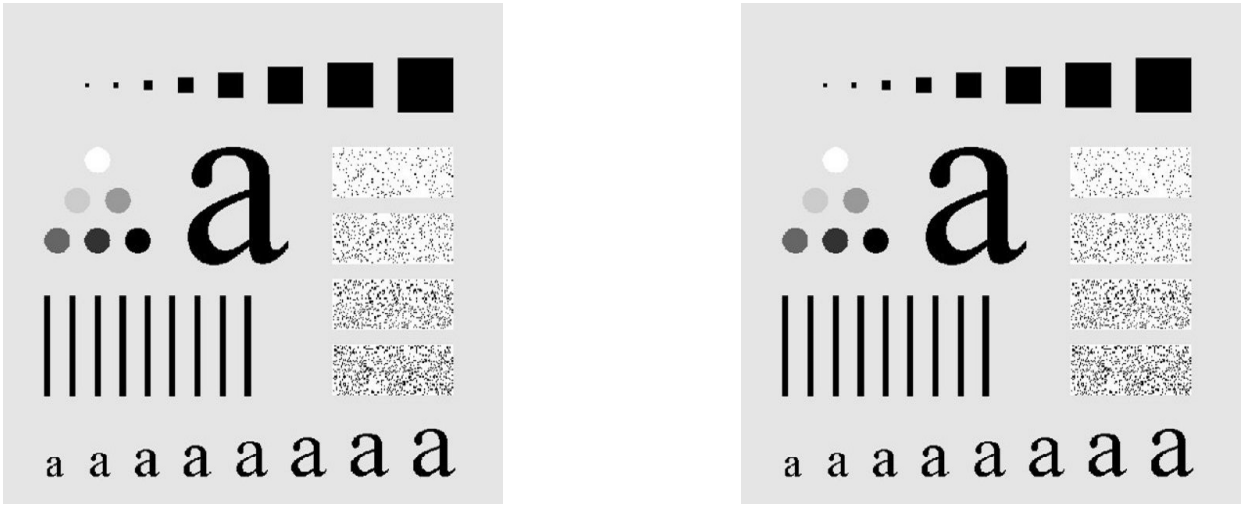
Summary

Name	BRAM_18K	DSP48E	FF	LUT
DSP	-	-	-	-
Expression	-	-	0	2
FIFO	0	-	30	120
Instance	-	1	365	404
Memory	-	-	-	-
Multiplexer	-	-	-	-
Register	-	-	-	-
Total	0	1	395	526

Figure 3.15: Pixel Access Passthrough IP hardware utilization.

Table 3.4: Comparison of FPGA board resources utilization for Pixel IP core.

BOARD	LUTs	Flip-Flops	BRAM
Zybo	N/A	N/A	N/A
Z1	0.99%	0.37%	0.16%
ZCU104 Ultrascale+	0.10%	0.01%	0.00%



(a) *Original grayscale image.* **(b)** *Pixel Access Passthrough processed.*

Figure 3.16: Pixel Access Passthrough - grayscale image results (image source [10]).



(a) *Original color image.* **(b)** *Pixel Access Passthrough processed.*

Figure 3.17: Pixel Access Passthrough - color image results (image source [10]).

$$Pixel_{gray} = 0.2126 Pixel_{red} + 0.7152 Pixel_{green} + 0.0722 Pixel_{blue} \tag{3.1}$$

In both the IP core code and the C++ equivalent code, each pixel of an input frame is used to calculate the corresponding grayscale value. This value is then assigned to each of the original R,G and B channels maintaining the grayscale value while allowing for display to a monitor. The flowchart depicting this process for the IP core can been seen in figure 3.18.

After completing the design flow within Vivado HLS a Grayscale IP core was created that could be implemented within a video pipeline architecture. The top-level block diagram of this IP core appears in figure 3.19. The Vivado HLS process also provides diagnostic and implementation summaries to catalog the particular implementation of a HLS created IP core. Of particular interest are the timing/latency summaries in figure 3.20 and the hardware utilization summary in figure 3.21. A comparison of hardware utilization across the different boards appears in table 3.5. This custom IP core was validated using a color image. The results are displayed in the figure 3.22.

For the sake of brevity, we detail here the performance of only three of our custom hardware video processing IP cores. The synopsis of the remaining IP cores in our custom library are detailed in the Appendix of this thesis. Although the individual cores do not reflect novel contributions, the implementation of this library was required for the creation of our FPGA-based Edge architecture.



Figure 3.18: Grayscale operation IP core flow chart. *The HDMI source signal is captured via the AXI interconnect and stored as an input frame within the Zynq PL. HLS code is then used to convert the R,G and B channels into grayscale equivalent. This value is then duplicated back to a 3-channel grayscale image and placed into the output frame before sending it out of the PL via AXI interconnect and then to HDMI sink.*

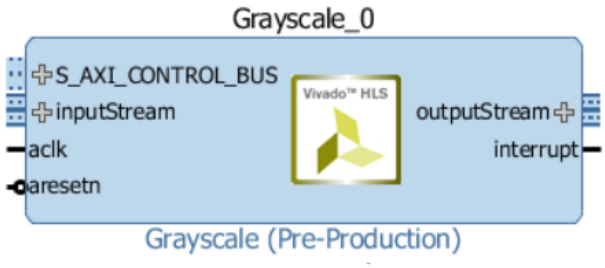


Figure 3.19: Grayscale IP core block diagram.

Timing (ns)

Summary

Clock	Target	Estimated	Uncertainty
ap_clk	10.00	9.40	1.25

Latency (clock cycles)

Summary

Latency		Interval		
min	max	min	max	Type
9	2081163	4	2081164	dataflow

Figure 3.20: Grayscale IP timing/latency.

Summary

Name	BRAM_18K	DSP48E	FF	LUT
DSP	-	-	-	-
Expression	-	-	0	2
FIFO	0	-	35	140
Instance	-	3	531	473
Memory	-	-	-	-
Multiplexer	-	-	-	-
Register	-	-	-	-
Total	0	3	566	615

Figure 3.21: Grayscale IP hardware utilization.

Table 3.5: Comparison of FPGA board resources utilization for Grayscale IP core.

BOARD	LUTs	Flip-Flops	BRAM
Zybo	2.33%	1.61%	0.00%
Z1	1.16%	0.53%	0.00%
ZCU104 Ultrascale+	0.12%	0.01%	0.00%



(a) *Original color image.* **(b)** *Grayscale processed.*

Figure 3.22: Grayscale - color image results.

Chapter 4

Benchmarking of CNNs in FPGA versus GPU

This chapter details the results of our experiments in benchmarking FPGA-based CNNs versus GPU-based CNNs for suitability to Edge deployment. To determine suitability we consider the inference performance of each CNN hardware implementation in conjunction with the relevant SWaP characteristics which impacts potential Edge deployment. In our experimental design we benchmark both hardware setups with two different CNNs: ResNet-50 for image classification and YOLO version 3 for object detection. The decision to benchmark two CNNs across two domain spaces was to establish the generalizability of our findings. Our findings establish that our FPGA-based system has near identical inference performance (a minimal drop of approximately 1% for both models) as the GPU-based system while also realizing a reduction in all SWaP characteristics. In particular, the FPGA-based solution achieved 2.4fps/W (ResNet-50) and 0.45fps/W (YOLO version 3) as compared to 0.65fps/W (ResNet-50) and 0.13fps/W (YOLO version 3) for the equivalent models on the GPU-based solution. Providing near identical inference performance while leveraging far superior SWaP characteristics establishes the FPGA-based solution as the superior Edge choice.

4.1 ResNet-50 FPGA Architecture

In this section we detail the architecture of ResNet-50, the training of the network, and our demonstration of this network in FPGA hardware. This section serves as a technical review for readers interested for setup of our benchmarking experiments. The following section lays out the experimental metrics as well as discusses our findings.

The ResNet neural network architecture was developed at Microsoft labs and was the winner of the 2015 ImageNet challenge [11]. The design of ResNet was based on the intuition that the deeper the layers of a network the higher the potential performance due to the increase in the number of parameters in which to capture the problem space. However, this strategy was shown to not be sufficient given that the VGG architecture displays decreased performance once reaching a limiting depth [70]. This phenomenon of diminishing performance is referred to as the "vanishing gradients" problem. It arises when the depth of

a network reaches a sufficient point such that the gradients calculated from the loss function approach zero which in turn causes the model weights to not update and effectively end learning new values.

A research team at Microsoft found a means of overcoming this limitation by the introduction of residual blocks within the deep neural architecture. These residual blocks, described in [11] and shown in figure 4.1, allow the gradients to propagate backwards from later layers back to other layers further up the network effectively mitigating the "vanishing gradients" problem. These residual blocks have allowed for various versions of the ResNet model to be designed as discussed in [11] and referenced in figure 4.2. Of these variations, the 50-layer network is commonly used as a backbone network for several current network implementations. With 50 layers, ResNet-50 has over 23 million trainable weights. As an image classifier, ResNet-50 remains a very common backbone architecture used for feature extraction in the design of new networks to address specific use-cases of classification. Therefore, its hardware implementation and performance provide insight to the potential performance of several real world applications.

Before implementation of the network into hardware could take place, a specific model of ResNet-50 needed to be produced via training. The model used in this thesis work was trained on the ImageNet-2012 training dataset (approximately 1.2 million images) and tested on the a subset of the ImageNet-2012 testing dataset (5,000 images). The ImageNet-2012 dataset is comprised of images representing 1,000 classes for the image classification task. The images themselves in the ImageNet-2012 dataset are 3-channel RGB with a resolution of 224x224 pixels.

As mentioned in the literature review, that are several existing methods in which to prepare deep learning models for hardware implementation targeting FPGA deployment. In our work we have elected to utilize Xilinx's DNNDK (Deep Neural Network Development Kit) tool chain to achieve this goal. The process of model quantization for hardware implementation is described in [12] and can be observed in figure 4.3. The method begins with a trained floating point model. The DNNDK tool can handle several models file types from the various major deep learning libraries (TensorFlow, Caffe, etc.). Next a pre-processing step removes useless nodes and performing batch folding normalization (a method to increase inference speed in deep learning). The 32-bit floating point model weights are then quantized to 8-bit integer versions with the assistance of a calibration dataset (a subset of the validation set that the original model was trained on). This quantized model is then built into an FPGA IP core, called a DPU (deep processing unit), via standard Vivado HLS tools. The DPU IP core is the final hardware implementation of the neural network that can be incorporated into a larger hardware design. The size of each of the models is outlined in table 4.1 where a decrease of 80% from the original floating point model is observed in the quantized model.

The hardware version of the ResNet-50 model was implemented both in a batch processing mode (via ethernet connection to the FPGA board) or in a streaming system. The streaming system is ideal for deployment use-cases as laid out in previous sections of this thesis. However, the batch processing mode is highly efficient when desiring to evaluate the performance of the hardware model given that per frame analysis can be performed and metrics calculated for a dataset at one time.

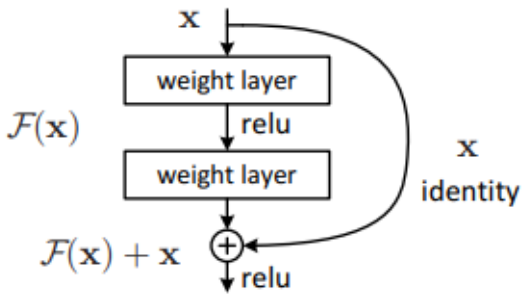


Figure 4.1: A standard residual layer in the ResNet architecture scheme [11].

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		[3×3, 64; 3×3, 64] ×2	[3×3, 64; 3×3, 64] ×3	[1×1, 64; 3×3, 64; 1×1, 256] ×3	[1×1, 64; 3×3, 64; 1×1, 256] ×3	[1×1, 64; 3×3, 64; 1×1, 256] ×3
conv3_x	28×28	[3×3, 128; 3×3, 128] ×2	[3×3, 128; 3×3, 128] ×4	[1×1, 128; 3×3, 128; 1×1, 512] ×4	[1×1, 128; 3×3, 128; 1×1, 512] ×4	[1×1, 128; 3×3, 128; 1×1, 512] ×8
conv4_x	14×14	[3×3, 256; 3×3, 256] ×2	[3×3, 256; 3×3, 256] ×6	[1×1, 256; 3×3, 256; 1×1, 1024] ×6	[1×1, 256; 3×3, 256; 1×1, 1024] ×23	[1×1, 256; 3×3, 256; 1×1, 1024] ×36
conv5_x	7×7	[3×3, 512; 3×3, 512] ×2	[3×3, 512; 3×3, 512] ×3	[1×1, 512; 3×3, 512; 1×1, 2048] ×3	[1×1, 512; 3×3, 512; 1×1, 2048] ×3	[1×1, 512; 3×3, 512; 1×1, 2048] ×3
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

Figure 4.2: Layer definitions for various versions of the ResNet model [11]. *Of particular importance for this thesis work is the layer configuration for the ResNet-50 variant.*

Table 4.1: ResNet-50 model sizes before and after quantization. *The size of the original floating point ResNet-50 model is reduced by 80% via the quantization method leveraged by the DNNDK tool. This quantized model is used to build the hardware DPU (deep processing unit) for FPGA.*

ResNet-50 Model	
Number of Parameters	23,000,000
Bytes needed for 32-bit float model	92,000,000
Bytes needed for 8-bit model (20% prune)	18,400,000
% decrease of achieved by quantized model	80%

4.2 ResNet-50 FPGA versus GPU Benchmarking

Given that that ResNet-50 is an image classification network, we determined that the most appropriate evaluation metric is that of accuracy. For our purposes, accuracy is defined by the equation in 4.1, where TP represents the number of true positive matches, TN represents the number of true negative matches, FP represents the number of false positive matches and FN represents the number of false negative matches.

$$Accuracy = \frac{TP + TN}{TP + FP + TN + FN} \tag{4.1}$$

Following the conventions used when evaluating image classifiers, we have elected to evaluate our model with both Top-1 and Top-5 accuracy metrics. Top-1 accuracy is readily described by 4.1 where during model evaluation only the output class with the highest probability is considered. With Top-5 accuracy, the equation remains the same but we evaluate accuracy based on the top 5 highest probability classes returned by the model. If the correct class is within the highest 5 probabilities then with top-5 accuracy the network was considered to have achieved a successful match.

Our 8-bit quantized hardware IP model was implemented on the Xilinx ZCU104 evaluation board and compared against the 32-bit floating point equivalent on a Nvidia RTX 2070 Super GPU. The results of this comparison appear in table 4.2 and validate similar results as described by Xilinx in [58]. As expected, the quantized network produces a slightly less accurate model in both top-1 and top-5 accuracy. Specifically, the FPGA had a 1.1% drop in the top-1 accuracy and a 0.54% drop in top-5 accuracy as compared to the GPU-based solution. This drop in accuracy is due to the lower capability of precision inherent to the quantization process. However, the drop in accuracy comes with increased hardware deployment savings for FPGA-based Edge solutions as show in table 4.3. The FPGA had a slight boost in frames-per-second performance, 144fps for FPGA versus 140fps for GPU. Although the FPGA solution is approximately half the size and weight of the GPU solution, of particular note is the performance-to-power metrics. When consuming maximum system power, the FPGA-based system achieves 2.4 fps/W whereas the GPU based-solution achieves 0.65 fps/W. These far superior SWaP characteristics, coupled with near identical model performance, make the FPGA-based ResNet-50 model the better selection for Edge deployment.

The following figures display examples of model inference as compared between the GPU and FPGA implementations for sample images from the testing set. Figure 4.4 provides an example testing image where both the GPU and FPGA models made the proper prediction of lighthouse. Figure 4.5 has the GPU predicting the correct class (snowplow) but the FPGA model predicting incorrectly (forklift). Figure 4.6 shows an instance where both the GPU and FPGA provided incorrect inferences to the ground truth of cell phone - the GPU model labeled the image remote control and the FPGA model labeled the image cassette player.

With these results we are able to demonstrate that a hardware based implementation of ResNet-50 provides comparable accuracy performance to a GPU implementation while meeting the strict SWaP constraints required in the deployment of Edge AI systems. While incurring only an approximate 1% drop in both top-1 and top-5 accuracy, the FPGA implementation provides a dramatic reduction in the required space, weight and power

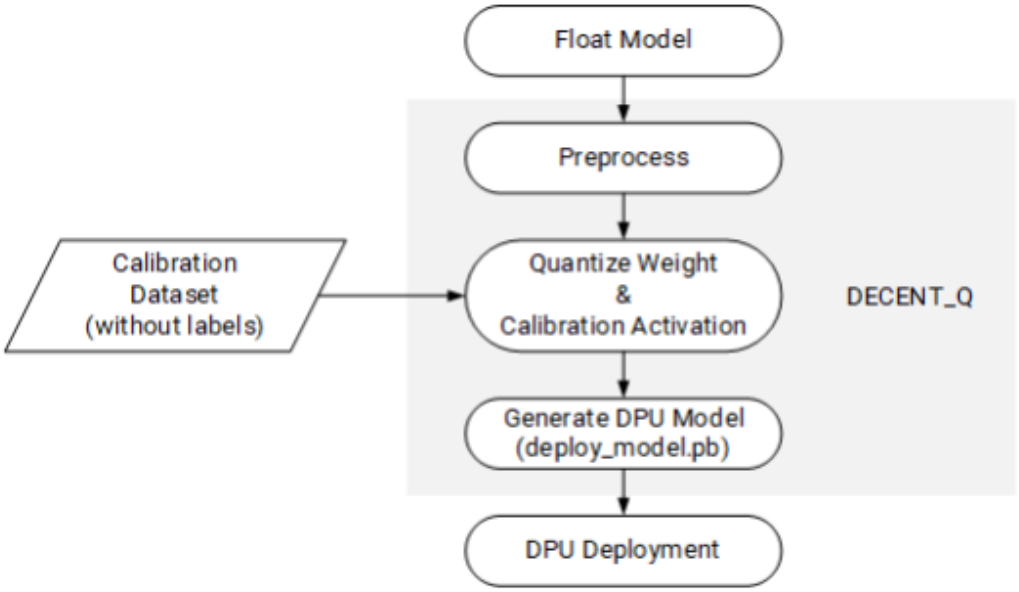


Figure 4.3: Work flow for model quantization via the Xilinx DNNDK tool [12].

Table 4.2: Comparison of GPU vs. FPGA performance on ResNet-50. *The GPU model leverages 32-bit floating point weights whereas the FPGA model is based on 8-bit integer weights produced by a quantization method.*

Implementation	Top-1 Accuracy	Top-5 Accuracy
Nvidia RTX 2070 Super	74.44%	91.85%
ZCU104 Ultrascale+	73.34%	91.31%



(a) *GPU Inference.* **(b)** *FPGA Inference.*

Figure 4.4: ResNet-50 example testing image where both GPU and FPGA models make the correct inference (lighthouse).



(a) *GPU Inference.*



(b) *FPGA Inference.*

Figure 4.5: ResNet-50 example testing image where the GPU model provides correct inference (snowplow) but the FPGA model is incorrect (forklift).



(a) *GPU Inference.*



(b) *FPGA Inference.*

Figure 4.6: ResNet-50 example testing image where both the GPU model and FPGA provides incorrect inference for the ground truth class of cellphone.

consumption making it ideal for such applications as UAVs, robotics or Edge surveillance cameras.

4.3 YOLO Version 3 FPGA Architecture

In this section we detail the architecture of YOLO version 3, the training of the network, and our demonstration of this network in FPGA hardware. This section serves as a technical review for readers interested for setup of our benchmarking experiments. The following section lays out the experimental metrics as well as discusses our findings.

Object detection is a more computationally complex computer vision problem than image classification. Object detection both detects an object and localizes it within an input image via a bounding box. Additionally, object detection can handle multiple instances of the same class within a single image unlike the task of image classification which only classifies the dominant class in the image.

Numerous object detection architectures exist and can be broadly placed into two categories, namely: two-stage detectors and one-stage detectors. The strategy of two-stage detectors is to first examine the image an propose regions of interest (ROIs) in the image where there is a high probability for the existence of a class. After ROIs are detected in the image (along with the probability of the existence of a class in the ROIs) in the first step, these are then utilized in the second pass for final object detection and localization. These two separate tasks comprise the "two stages" in a two-stage object detection network. Two-stage detectors typically are the most accurate but are slower than one-stage detectors. Networks such as R-CNN [71], Fast R-CNN [72] and Faster R-CNN [73] are examples of two-stage object detectors.

One-stage object detection networks only process the image once. This is achieved by framing object detection as a regression problem where object class probabilities and bounding box coordinates are learned. Since these architectures process an input image only once they are well suited for real-time image processing. However, one-stage detector networks' increase in speed often comes at the price of reduced accuracy. One of the more famous of the one-stage detector networks is YOLO.

You Only Look Once (YOLO) is the name of a set of versions of one-stage CNN architectures whose goal is object detection. The original YOLO paper [13] from 2016 introduces what is now known as version 1 of the architecture as seen in figure 4.7. The principal steps of YOLO are to first divide the input image into a series of grids. The architecture then predicts the probability of bounding boxes within these grids and predicts the various class scores in the boxes. Then the architecture selects appropriate bounding boxes via the confidence threshold and the intersection-over-union (IoU, the area of overlap as compared to the area of union) as seen in equation 4.2 and leverages non-max suppression to eliminate duplicate bounding boxes for a single object. This informs the final object detection and localizations based on whether the detection meets the IoU and confidence thresholds.

$$IoU = \frac{area\ of\ overlap}{area\ of\ union} \tag{4.2}$$

YOLO has proven to remain a powerful contender as a real-time object detector with several revisions to improve model accuracy while maintaining speed of inference (version 2 [74] and version 3 [75]). For this thesis, YOLO version 3 was selected for implementation into hardware based upon its high inference speed which is ideal for Edge AI systems.

Before implementation of the network into hardware could take place, a specific model of YOLO version 3 needed to be produced via training. The model used in this thesis work was trained on the VOC2007 training dataset consisting of 9,963 images with 24,640 annotated objects and tested on the VOC2007 testing dataset (4,952 images). The VOC2007 dataset is comprised of images representing 20 classes for the object detection task. The images themselves in the VOC2007 dataset are 3-channel RGB with a resolution of 416x416 pixels.

The tools and steps used to implement the tensorflow model of YOLO version 3 into FPGA hardware are the same as were used in the previous section for ResNet-50. Once again a 32-bit floating point model was quantized to a 8-bit integer model via Xilinx's DNNDK tool chain as seen in figure 4.3. Once completed, the YOLO version 3 DPU IP core could be incorporated into a larger hardware design. The size of each of the models is outlined in table 4.4 where a decrease of 78.75% from the original floating point model is observed in the quantized model.

As with the ResNet-50 DPU core, the hardware version of the YOLO version 3 model was implemented both in a batch processing mode (via ethernet connection to the FPGA board) or in a streaming system. The streaming system is ideal for deployment use-cases as laid out in previous sections of this thesis. However, the batch processing mode is highly efficient when evaluating the performance of the hardware model given that per frame analysis can be performed and metrics calculated for a dataset at one time.

4.4 YOLO Version 3 FPGA versus GPU Benchmarking

Since the YOLO version 3 is an object detection network, we determined that the most appropriate evaluation metric is that of mean average precision (mAP). Precision is defined in equation 4.3 where TP represents the number of true positive matches and FP references to the number of false positive matches. Average precision (AP) can be defined as the area under the precision-recall curve as defined in equation 4.4. Mean average precision is then the average of AP per number of classes (n) as defined in equation 4.5. Of additional note, the IoU threshold for this model was set to the default YOLO value of 0.5.

$$Precision = \frac{TP}{TP + FP} \tag{4.3}$$

$$AP = \int_0^1 p(r)dr \tag{4.4}$$

$$mAP = \frac{1}{n}\sum_1^n AP(i), where\ n = number\ of\ classes \tag{4.5}$$

Table 4.3: Comparison of GPU v.s FPGA considering SWaP (size, weight and power) characteristics in relation to accuracy performance with ResNet-50 model. *Although an FPGA is a self-contained unit, the GPU requires other support hardware to run which is indicated by the * for additional computing ecosystem hardware. Also of note is that for this implementation the FPGA wattage** includes all I/O resources.*

System	Top-1 Accuracy	Top-5 Accuracy	Physical Dimensions	Physical Weight	Max. Power	Through-put
Nvidia RTX 2070 Super	74.44%	91.85%	267mm x 116mm x 35mm*	2,336g*	215W*	140 fps
ZCU104 Ultrascale+	73.34%	91.31%	179mm x 149mm x 1.5mm	1,133 g	60W**	144 fps

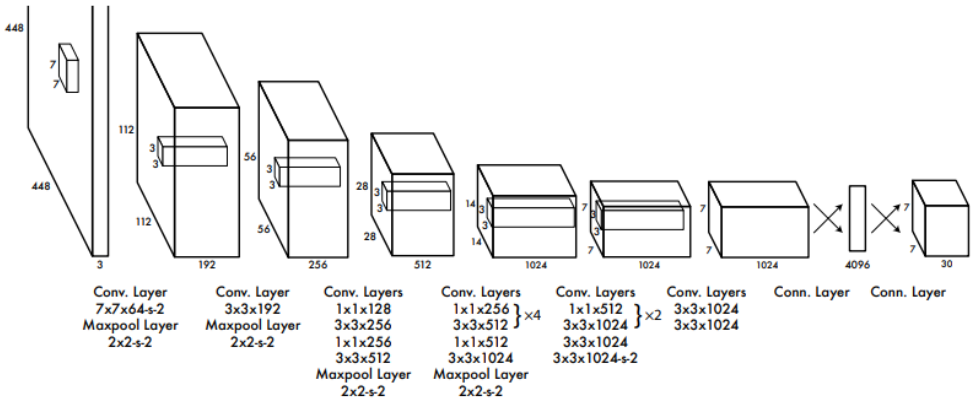


Figure 4.7: CNN architecture design of YOLO version 1 [13].

Table 4.4: YOLO version 3 model sizes before and after quantization. *The size of the original floating point YOLO version 3 model is reduced by 78.75% via the quantization method leveraged by the DNNDK tool. This quantized model is used to build the hardware DPU (deep processing unit) for FPGA.*

YOLO version 3 Model	
Number of Parameters	59,000,000
Bytes needed for 32-bit float model	236,000,000
Bytes needed for 8-bit model (15% prune)	50,150,000
% decrease of achieved by quantized model	78.75%

Our 8-bit quantized hardware IP model was implemented on the Xilinx ZCU104 evaluation board and compared against the 32-bit floating point equivalent on a Nvidia RTX 2070 Super GPU. The results of this comparison appear in table 4.5 and validate similar results as described by Xilinx in [58]. As expected, the quantized network produces a slightly less accurate model based on the mAP scores. Specifically, the FPGA had a 1.02% drop in mAP as compared to the GPU-based solution. This drop in performance is due to the lower capability of precision inherent to the quantization process. However, the drop in accuracy comes with increased hardware deployment savings for Edge solutions as show in table 4.6. In contrast to the ResNet-50 model, the GPU-based model had a slight boost in frames-per-second performance, 29fps for the GPU versus 27fps for the FPGA. Although the FPGA solution is approximately half the size and weight of the GPU solution, of particular note is the performance-to-power metrics. When consuming maximum system power, the FPGA-based system achieves 0.45 fps/W whereas the GPU based-solution achieves 0.13 fps/W. These far superior SWaP characteristics, coupled with near identical model performance, make the FPGA-based YOLO version 3 model the better selection for Edge deployment.

The following figures display examples of model inference as compared between the GPU and FPGA implementations for sample images from the testing set. Figure 4.8 provides an example testing image where both the GPU and FPGA models made the proper prediction (two instances of the class cat). Figure 4.9 has the GPU predicting the correct class (3 instances of potted plant) but the FPGA model predicting incorrectly (3 correct instances of potted plant plus an incorrect instance of potted plant). Figure 4.10 shows an instance where both the GPU and FPGA provided incorrect inferences to the ground truth of a single instance of class cat - the GPU-based system labeled two instances of class cat and the FPGA-based system labeled three instances of class cat.

As with the image classifier ResNet-50, we are able to demonstrate that a hardware based implementation of YOLO version 3 provides comparable accuracy performance to a GPU-based implementation while meeting the strict SWaP constraints required in the deployment of Edge AI systems. With an approximate 1% drop in mean average precision (mAP), the FPGA implementation provides a dramatic reduction in the required space, weight and power consumption making it ideal for Edge-based AI solutions.

4.5 Findings From Benchmarking FPGA-based and GPU-based CNNs

This chapter has detailed our efforts benchmarking FPGA-based implementations against GPU-based implementations of CNNs. To demonstrate the generalizable nature of our results, two different CNN architectures were selected: ResNet-50 and YOLO version 3. Our findings establish that our FPGA-based system proves to be the superior choice for Edge deployment providing its demonstrated near identical inference performance (a minimal drop of approximately 1% for both models) as compared to the GPU-based system while also realizing a reduction in all SWaP characteristics. In particular, the FPGA-based solution achieved 2.4fps/W (ResNet-50) and 0.45fps/W (YOLO version 3) as compared to 0.65fps/W (ResNet-50) and 0.13fps/W (YOLO version 3) for the equivalent models on the GPU-based

Table 4.5: Comparison of GPU vs. FPGA performance on YOLO version 3. *The GPU model leverages 32-bit floating point weights whereas the FPGA model is based on 8-bit integer weights produced by a quantization method.*

Implementation	mean average precision (mAP)
Nvidia RTX 2070 Super	78.46%
ZCU104 Ultrascale+	77.44%



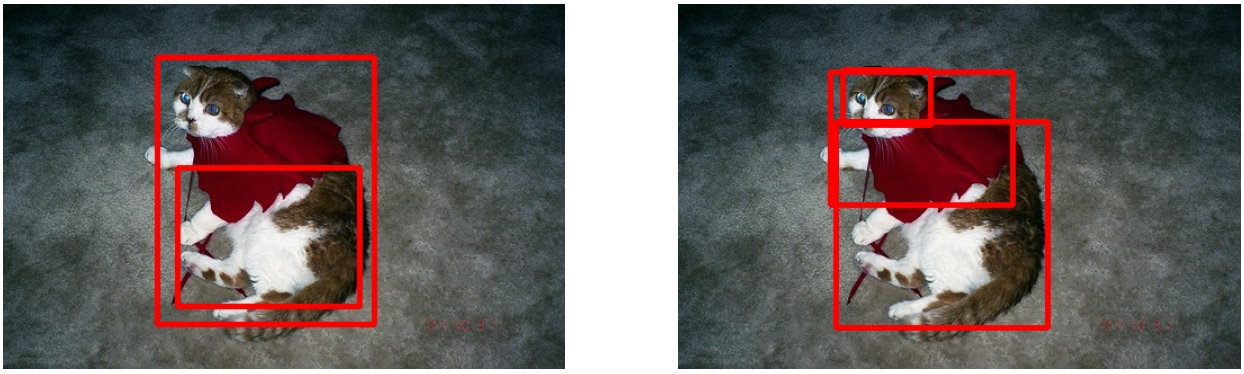
(a) *GPU Inference.* (b) *FPGA Inference.*

Figure 4.8: YOLO version 3 example testing image where both GPU and FPGA models make the correct inference (2 instances of class cat).



(a) *GPU Inference.* (b) *FPGA Inference.*

Figure 4.9: YOLO version 3 example testing image where the GPU model provides correct inference (3 instances of potted plant class) but the FPGA model is incorrect (4 instances of potted plant class).



(a) *GPU Inference.* **(b)** *FPGA Inference.*

Figure 4.10: YOLO version 3 example testing image where both the GPU and FPGA models provide incorrect inference for the ground truth class of one instance of class cat.

Table 4.6: Comparison of GPU v.s FPGA considering SWaP (size, weight and power) characteristics in relation to accuracy performance with YOLO version 3 model. *Although an FPGA is a self-contained unit, the GPU requires other support hardware to run which is indicated by the * for additional computing ecosystem hardware. Also of note is that for this implementation the FPGA wattage** includes all I/O resources.*

System	mean average precision (mAP)	Physical Dimensions	Physical Weight	Max. Power	Through-put
Nvidia RTX 2070 Super	78.46%	267mm x 116mm x 35mm*	2,336g*	215W*	29 fps
ZCU104 Ultrascale+	77.44%	179mm x 149mm x 1.5mm	1,133 g	60W**	27 fps

solution. Efficiency in power consumption while processing video, coupled with smaller physical dimensions and weight, make the FPGA-based solution ideal for Edge systems such as UAVs and autonomous vehicles.

Chapter 5

Image Degradation and Correction

In the previous chapter we demonstrated that the FPGA system provides near identical inference performance compared to the GPU counterpart. However, images in unconstrained scenarios where Edge systems are deployed are often degraded due to various factors. Therefore, determining if the FPGA model would provide similar robustness to noise as compared to the GPU model allows for one measure of explainability when an incorrect inference is made. In this chapter we show a baseline statistical analysis of the impact of two degradation methods, along with corresponding correction methods, on the evaluated datasets from Chapter 4 (ImageNet 2012 and VOC2007) via a quantitative image quality metric which illustrates that neither contain an inherent bias towards our evaluated methods.

Although numerous degradation types exist (and several modeled types were explored as video processing cores in Chapter 3), for the sake of brevity we chose to explore the effects caused by two highly common image degradation types: contrast reduced images and salt-and-pepper noise. Videos and images with reduced contrast are common in non-ideal lighting conditions or in foggy/hazy environments. Salt-and-pepper noise is common in inexpensive sensors used by low quality cameras or can be caused by excessive mechanical disturbance from sensors deployed in harsh conditions. These image degradation types were implemented via video processing hardware IP cores to alter the input image and quantify the effect to stream quality. Additionally, for the Edge AI vision system to be optimal, it should possess a means to correct image degradation. Given the two explored degradation types outlined above, appropriate correction techniques were also implemented in video processing hardware IP cores. Gamma correction (by means of lookup table) was implemented to offset the effects of reduced contrast frames. The parameters of the gamma correction IP can be adjusted to account for the degree to which a frame is impacted. For salt-and-pepper degradation a median filter IP core was implemented whose parameters can also be adjusted as needed for the amount of image noise.

In particular it should be noted that we chose to evaluate noises and corrections within the spatial domain for this thesis work. We additionally hypothesize that the SSIM IQA metric of an input image will correlate to the CNN accuracy on that same image. In particular, the higher the SSIM score the higher the accuracy and the lower the SSIM score the lower the accuracy during inference.

5.1 Image Degradation Experimental Setup and Evaluation

As mentioned in the previous chapter, the CNN networks under consideration span various vision applications and as such have corresponding datasets to be evaluated on. Namely, ImageNet-2012 and VOC2007 datasets were selected as the basis for evaluation. To determine the effects of degradation and correction on CNN performance it is also of interest to see how these functions alter the quality of the dataset as a whole.

Figure 5.1 provides a visual representation of the contrast reduction process. The original image is displayed in the top left. This original is then split into its three color channels (red, green, and blue). Each of these channels has the original dynamic range reduced by a desired percentage (in this example at 50%). The degraded channels are then combined to create the new contrast reduced image as seen in the bottom left image. In our experiments we explore contrast reduction with parameters of 100% (no change in contrast) to 2.5% contrast reduction.

Figure 5.2 provides a visual display of the impact of the contrast reduction degradation upon a sample image. These images are degraded so that the response provides specific SSIM scores ranging from 1.0 (the original image in the top left) and then proceeding by a drop in score of 0.1 steps until the heavily degraded image in the bottom right is created with a SSIM score of 0.1.

Figure 5.3 provides a visual representation of the salt-and-pepper degradation process. The original image is displayed in the top left. A probability of noise is then selected which informs the algorithm to the probability of a pixel being randomly set to white or to black (in this example a noise probability of 0.01 was selected). After the randomized salt-and-pepper noise is calculated it is then combined with the original image to produce the final degraded result. In our experiments we explore salt-and-pepper degradation with probability parameters of 0 (no noise) to 0.1.

Figure 5.4 provides a visual display of the impact of the salt-and-pepper degradation upon a sample image. As with the previous figure, the degradation is applied such that the top left image has a SSIM score of 1.0 (original image) and follows 0.1 steps in score until the bottom right image is produced with a SSIM score of 0.1. These images serve as a visual reference for other degradation based experiments in this thesis.

To quantify and visualize the effects of the selected degradation types and associated corrections listed above, a series of experiments were ran on the two base datasets themselves. The selected experiments apply the functions over a series of values chosen to reflect the range in which the quality is most impacted. Plots of these results, organized by dataset and degradation type, are laid out below. In each of the degradation plots in this section, the curve reports the average SSIM score across ***all*** images in the dataset for the degradation parameter.

The results of the contrast reduction degradation type applied to the ImageNet dataset can be seen in Figure 5.5. Additionally, a series contrast reduction degradation parameters for a sample ImageNet image appear in Figure 5.6. These images demonstrate the effect of the contrast reduction degradation for the parameter range of 100.0% (original image) to 2.5%. For the contrast reduction based degradation it was found that parameters beyond

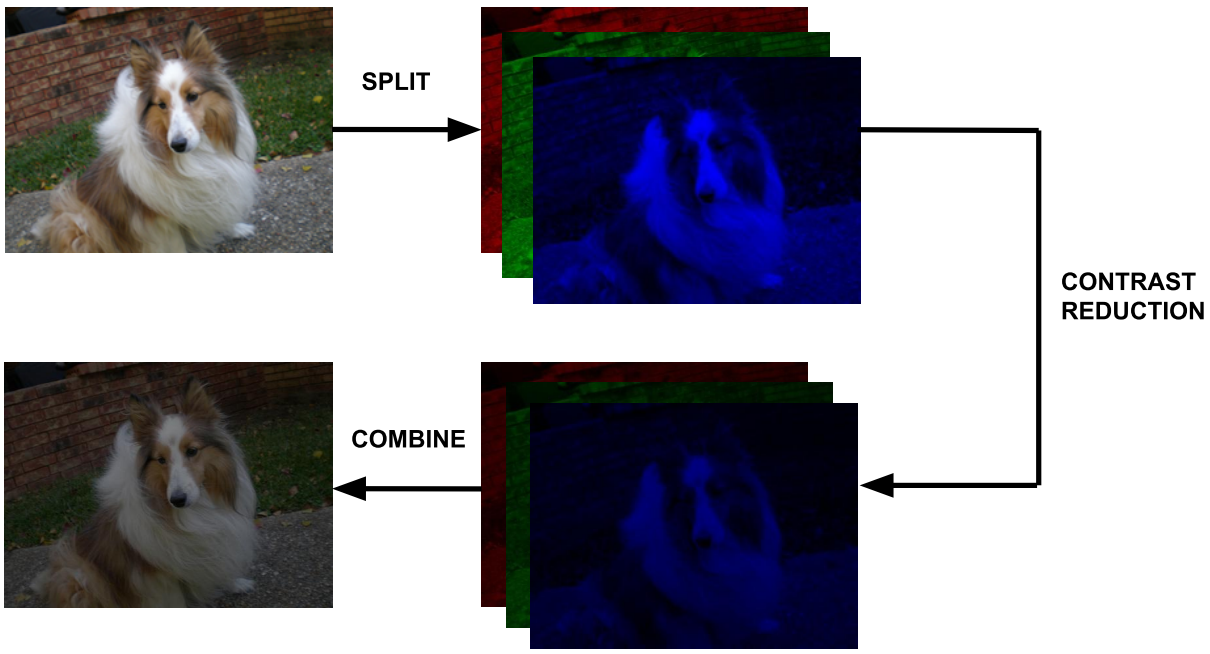


Figure 5.1: Process of contrast reduction degradation. *The original image in the top left is first split into its three color channels (red, green, blue). Each of the channels has the dynamic range reduced according to the desired percentage (in this example 50% contrast reduction was selected). Each degraded channel is then combined to create the resulting degraded color image as seen in the bottom left image.*

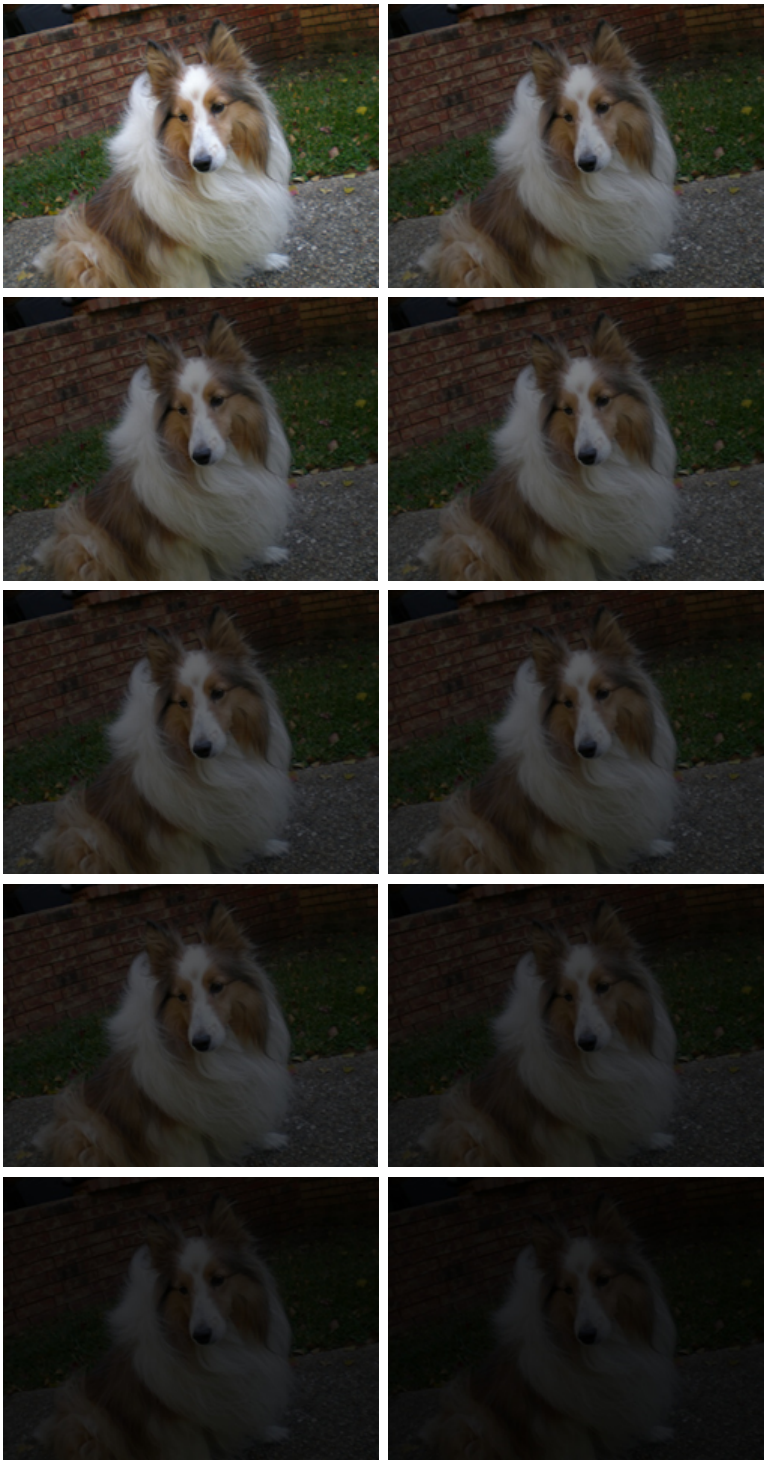


Figure 5.2: Demonstration of SSIM on contrast reduction degradation. *The images from top left to bottom right are degraded by contrast reduction to provide SSIM scores of 1.0, 0.9, 0.8, ..., 0.1 to provide a visual demonstration of quality degradation.*

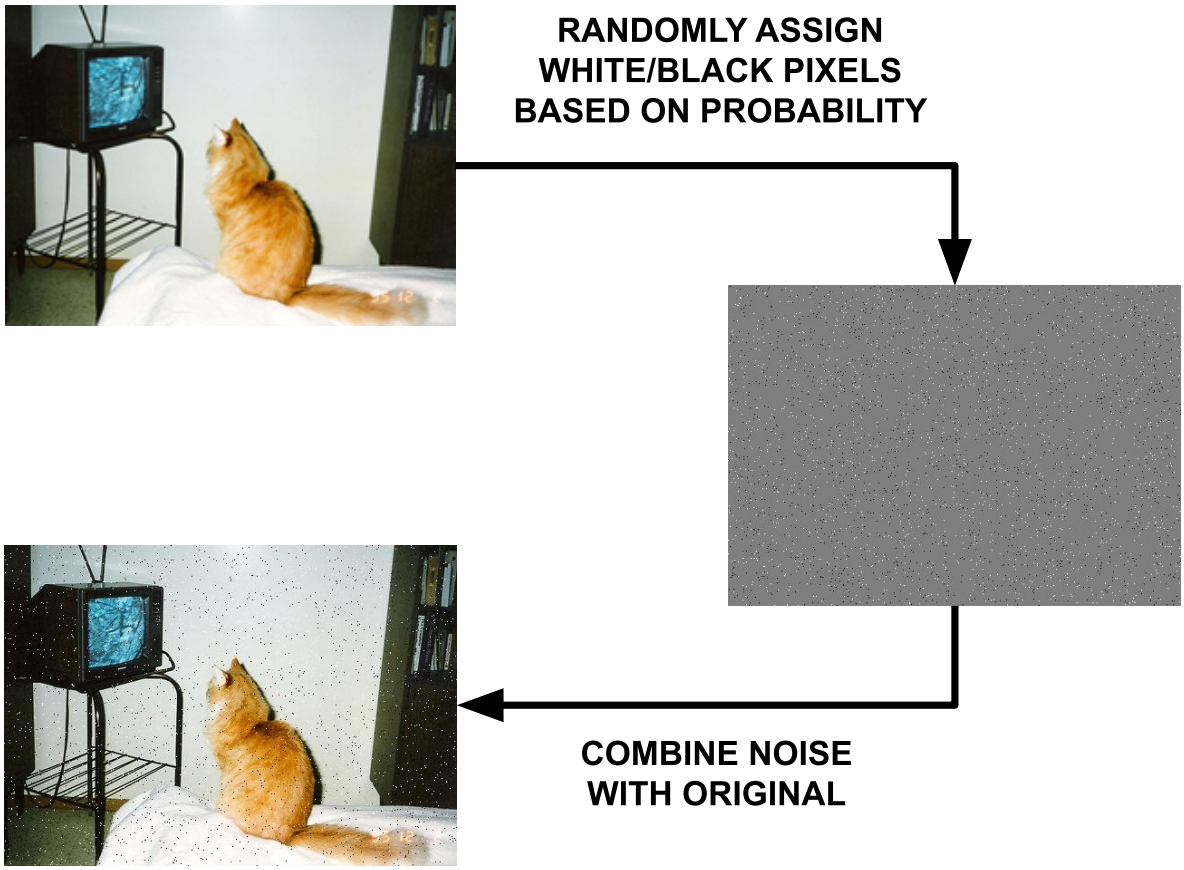


Figure 5.3: Process of salt-and-pepper degradation. *The original image appears in the top left. The desired probability for salt-and-pepper is assigned and pixels are set to either white or black accordingly (in this example a noise probability of 0.01 was selected). The randomized noise is then combined with the original image to produce the salt-and-pepper degradation.*

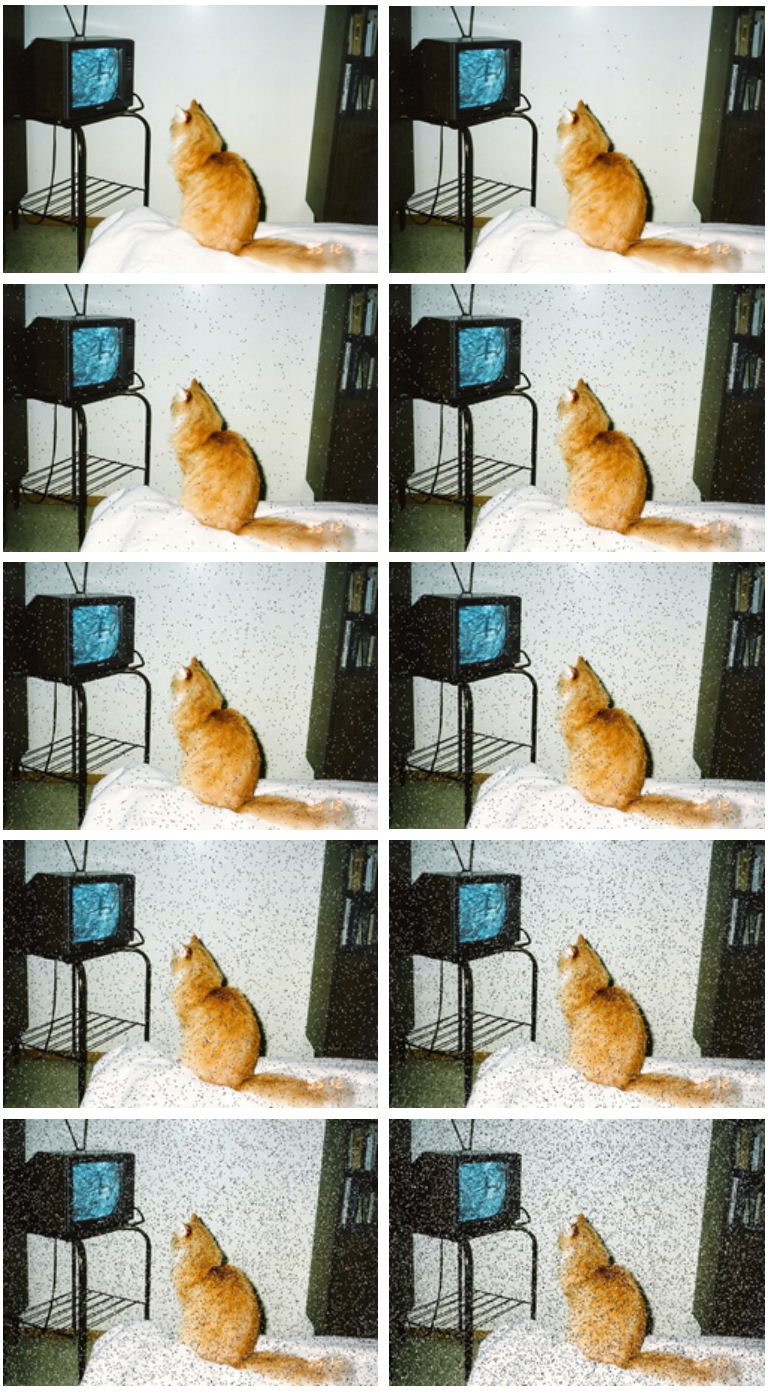


Figure 5.4: Demonstration of SSIM on salt-and-pepper degradation. *The images from top left to bottom right are degraded by salt-and-pepper to provide SSIM scores of 1.0, 0.9, 0.8, ..., 0.1 to provide a visual demonstration of quality degradation.*

the value of 2.5% produced very little change in the SSIM score and therefore the plots are bounded here. For the ImageNet dataset, the contrast reduction degradation type was shown to be the second most degrading.

The results of the salt-and-pepper degradation type applied to the ImageNet dataset can be seen in Figure 5.7. Additionally, a series of salt-and-pepper degradation parameters for a sample ImageNet image appear in Figure 5.8. These images demonstrate the effect of the salt-and-pepper degradation for the parameter range of 0 to 0.1. For the salt-and-pepper based degradation it was found that parameters beyond the value of 0.1 produced very little change in the SSIM score and therefore the plots are bounded here. For the ImageNet dataset, the salt-and-pepper degradation type was shown to be the most degrading.

The results of the contrast reduction degradation type applied to the VOC2007 dataset can be seen in Figure 5.9. Additionally, a series of contrast reduction degradation parameters for a sample VOC2007 image appear in Figure 5.10. These images demonstrate the effect of the contrast reduction degradation for the parameter range of 100.0% (original image) to 2.5%. For the contrast reduction based degradation it was found that parameters beyond the value of 2.5% produced very little change in the SSIM score and therefore the plots are bounded here. For the VOC2007 dataset (as with the ImageNet dataset), the contrast reduction degradation type was shown to be the second most degrading.

The results of the salt-and-pepper degradation type applied to the VOC2007 dataset can be seen in Figure 5.11. Additionally, a series of salt-and-pepper degradation parameters for a sample VOC2007 image appear in Figure 5.12. These images demonstrate the effect of the salt-and-pepper degradation for the parameter range of 0 to 0.1. For the salt-and-pepper based degradation it was found that parameters beyond the value of 0.1 produced very little change in the SSIM score and therefore the plots are bounded here. For the VOC2007 dataset (as with the ImageNet dataset), the salt-and-pepper degradation type was shown to be the most degrading.

Of additional interest was whether or not these degradation types may impact one dataset more severely than another dataset. If so, this may indicate that a particular type of subject or object in a dataset is more susceptible to a particular degradation. Figures 5.13 and 5.14 plot the SSIM performance curves for each dataset on each of the two degradation types. As with the previous degradation plots, the curves report the average SSIM score across ***all*** images in a particular dataset.

In Figure 5.13, the effects of contrast reduction degradation can be observed across the two datasets. Both the ImageNet and VOC2007 datasets, which contain various classes of objects, appear to respond almost identically to the contrast reduction based degradation.

In Figure 5.14, the salt & pepper degradation is shown which was the most degrading for both of the datasets. Once again, the ImageNet and VOC2007 datasets respond similarly.

5.2 Image Correction Experimental Setup and Evaluation

Now that the degree of degradation based on parameter for each dataset of interest has been quantified by the SSIM metric, we explore how much quality can be regained from applying

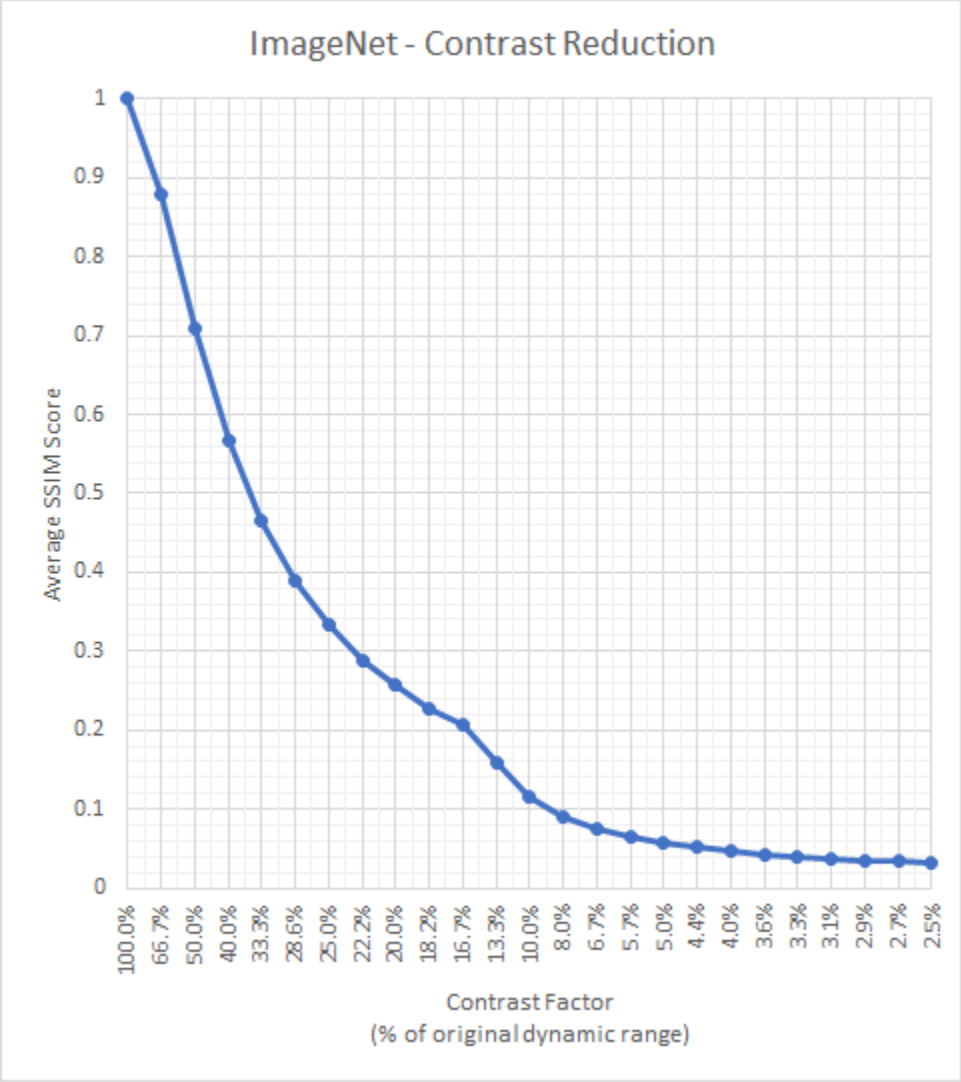


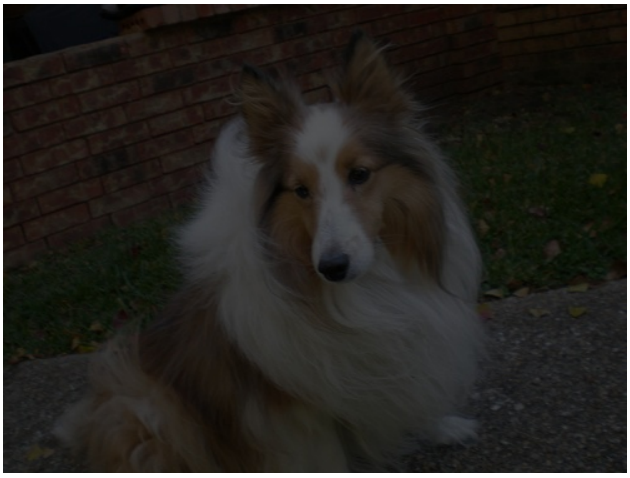
Figure 5.5: ImageNet - contrast reduction. *As the degradation parameter increases the SSIM score drops accordingly. With contrast reduction of 50% the average SSIM score for the entire data set quickly drops to 0.72. Between 50% and 10% contrast reduction the rate at which the SSIM score is impacted begins to level off. Beyond a parameter of 2.5% the value of the SSIM score flattens and is not reported here. The response of this experiment demonstrates that the SSIM score can be used to predict quality which may be a predictor for CNN inference accuracy.*



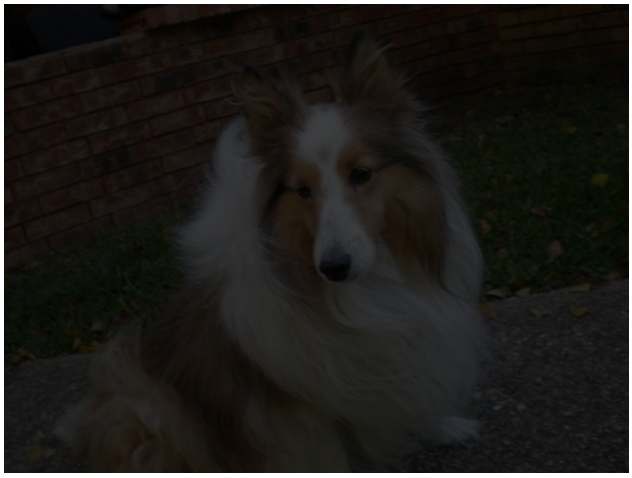
(a) *Original Image. SSIM = 1.0.*



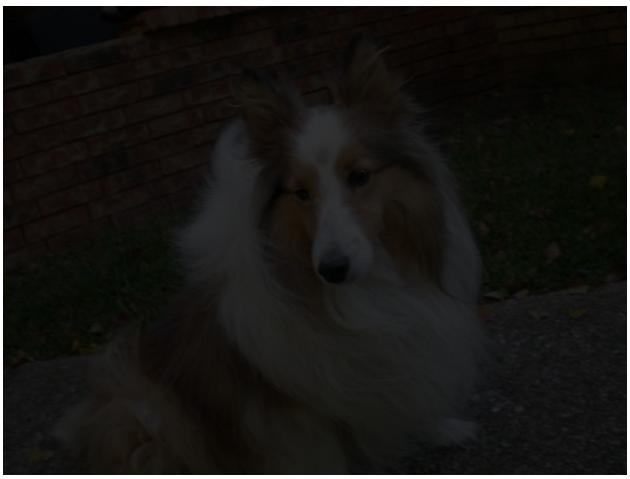
(b) *Contrast = 50.0%. SSIM = 0.6881.*



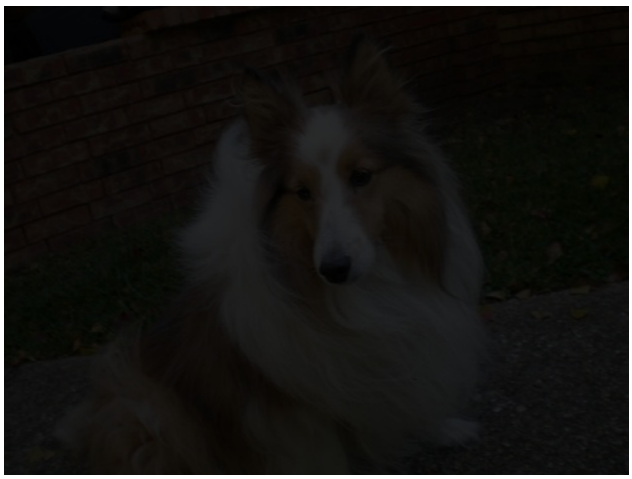
(c) *Contrast = 33.3%. SSIM = 0.4472.*



(d) *Contrast = 25.0%. SSIM = 0.3154.*



(e) *Contrast = 20.0%. SSIM = 0.2385.*



(f) *Contrast = 16.7%. SSIM = 0.1895.*

Figure 5.6: ImageNet - contrast reduction sample images.

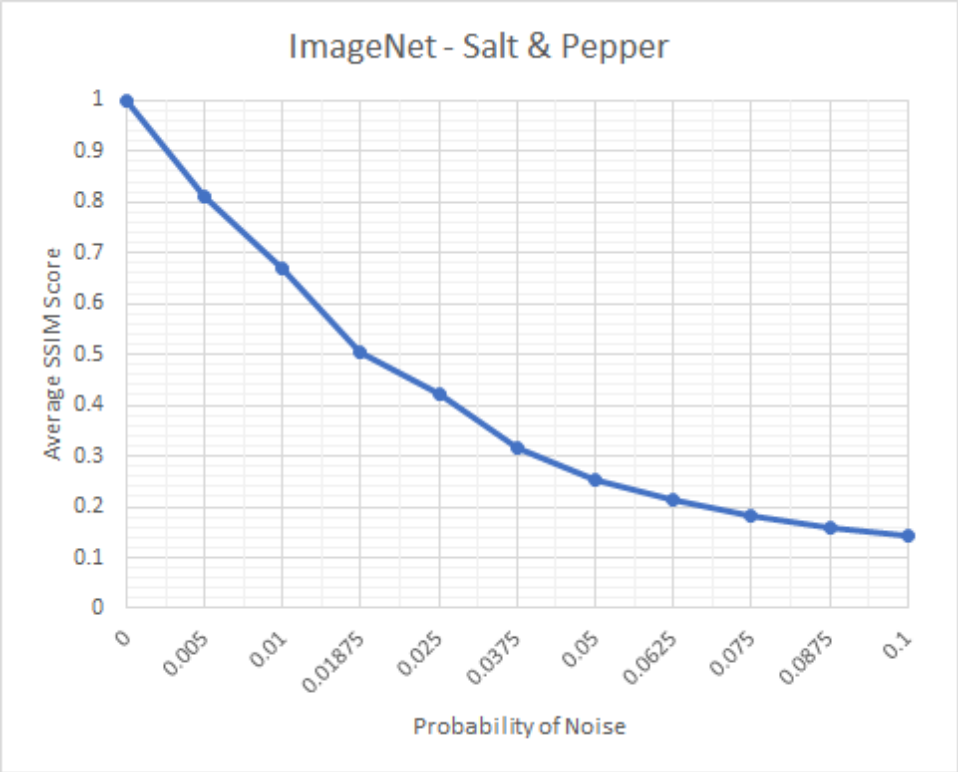


Figure 5.7: ImageNet - salt & pepper. *As the degradation parameter increases the SSIM score drops accordingly. As compared to the contrast reduction noise, the impact upon the SSIM score is more gradual for salt-and-pepper degradation. At a probability of 0.01875 the average SSIM score for the entire dataset drops to 0.5. From 0.0187 to 0.1 the rate of SSIM reduction slows and beyond a parameter of 0.1 the value of the SSIM score flattens and is not reported here. The response of this experiment demonstrates that the SSIM score can be used to predict quality which may be a predictor for CNN inference accuracy.*



(a) *Original Image. SSIM = 1.0.*



(b) *Probability = 0.01. SSIM = 0.6177.*



(c) *Probability = 0.025. SSIM = 0.3529.*



(d) *Probability = 0.05. SSIM = 0.1794.*



(e) *Probability = 0.075. SSIM = 0.1129.*



(f) *Probability = 0.1. SSIM = 0.0801.*

Figure 5.8: ImageNet - salt-and-pepper sample images.

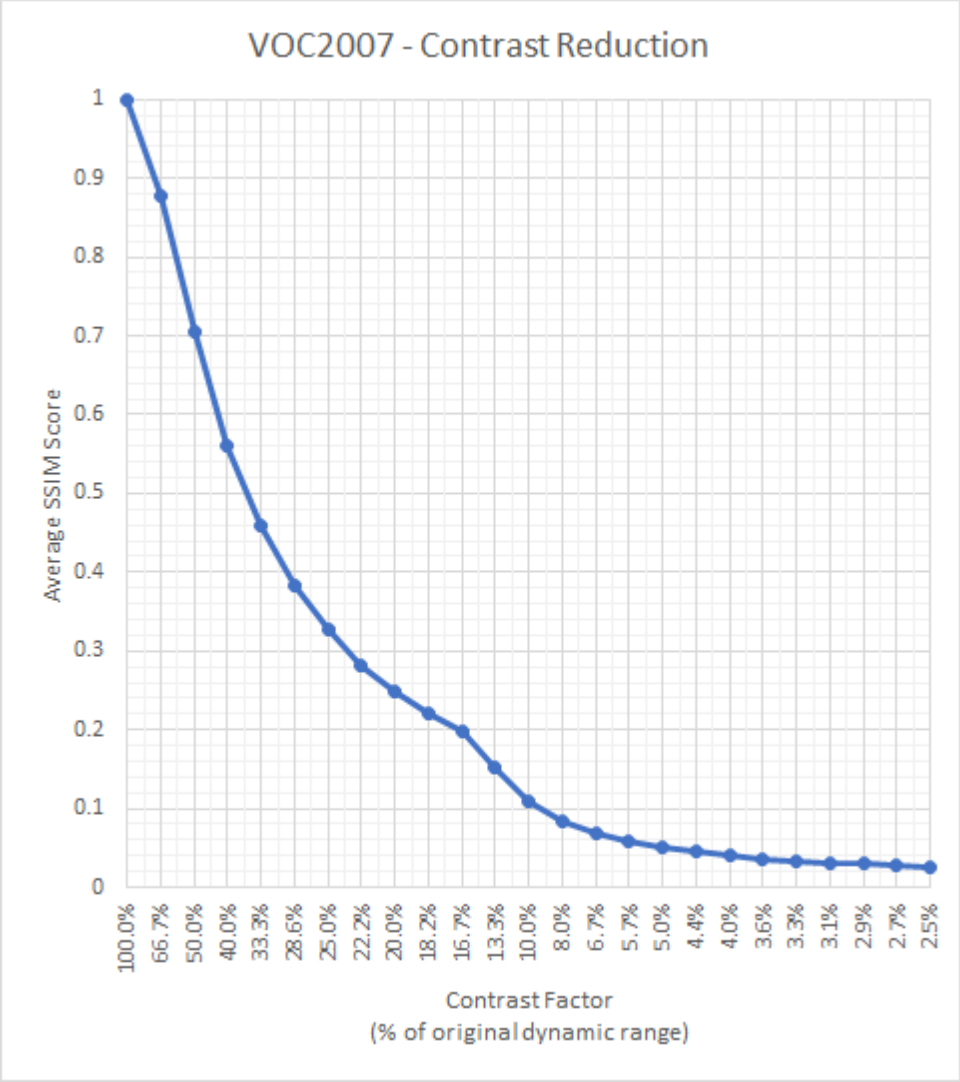
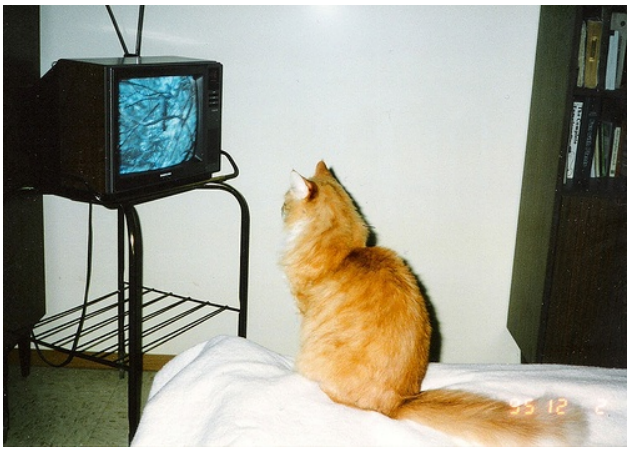
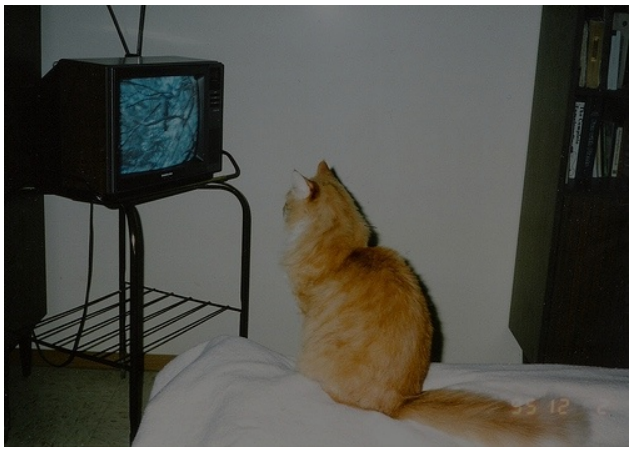


Figure 5.9: VOC2007 - contrast reduction. *As the degradation parameter increases the SSIM score drops accordingly. It can be observed that the VOC2007 dataset's SSIM response to contrast reduction is near identical to that of ImageNet. At contrast reduction of 50% the average SSIM score for the entire dataset is 0.71%. As with the ImageNet contrast reduction experiment, beyond a parameter of 2.5% the value of the SSIM score flattens and is thus not reported here. Given that both datasets respond to contrast reduction in a near identical manner, the evidence to support a potential correspondence between SSIM score for image quality and CNN inference accuracy is increased.*



(a) *Original Image. SSIM = 1.0.*



(b) *Contrast = 50.0%. SSIM = 0.7117.*



(c) *Contrast = 33.3%. SSIM = 0.4623.*



(d) *Contrast = 25.0%. SSIM = 0.3368.*



(e) *Contrast = 20.0%. SSIM = 0.2593.*



(f) *Contrast = 16.7%. SSIM = 0.2107.*

Figure 5.10: VOC2007 - contrast reduction sample images.

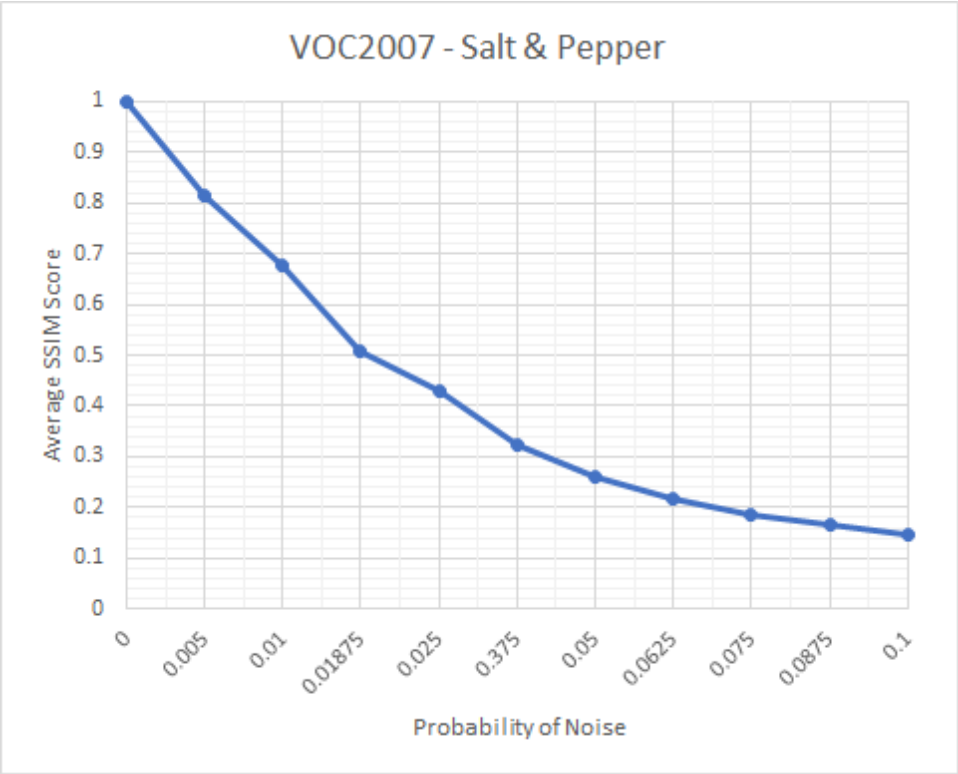
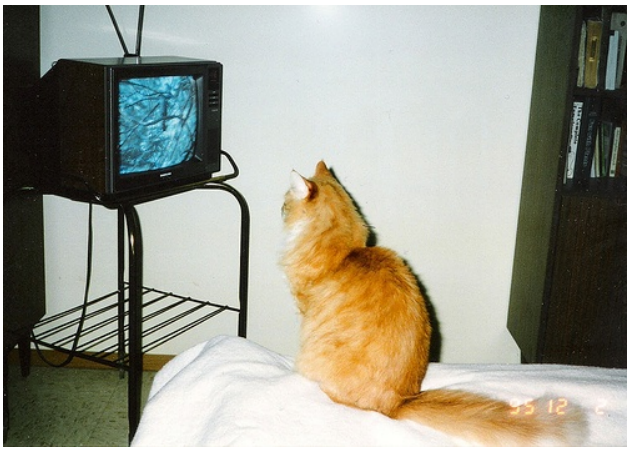
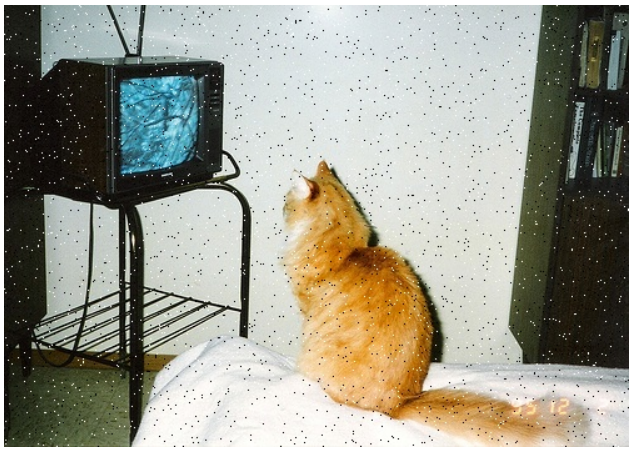


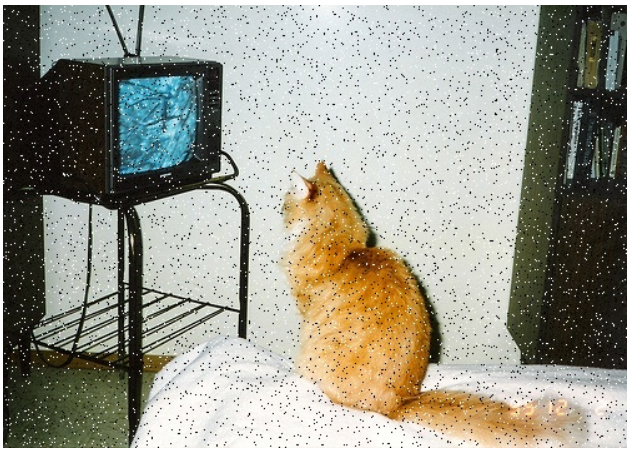
Figure 5.11: VOC2007 - salt & pepper. *As the degradation parameter increases the SSIM score drops accordingly. It can be observed that the VOC2007 dataset's SSIM response to salt-and-pepper degradation is near identical to that of ImageNet. At a probability of 0.01875 the average SSIM score for the entire dataset drops to 0.51. As with the ImageNet salt-and-pepper experiment, beyond a parameter of 0.1 the value of the SSIM score flattens and is thus not reported here. This finding demonstrates that both datasets have near identical SSIM score response to both selected degradation types. This correspondence provides evidence for leveraging the SSIM image quality score to predict CNN inference performance.*



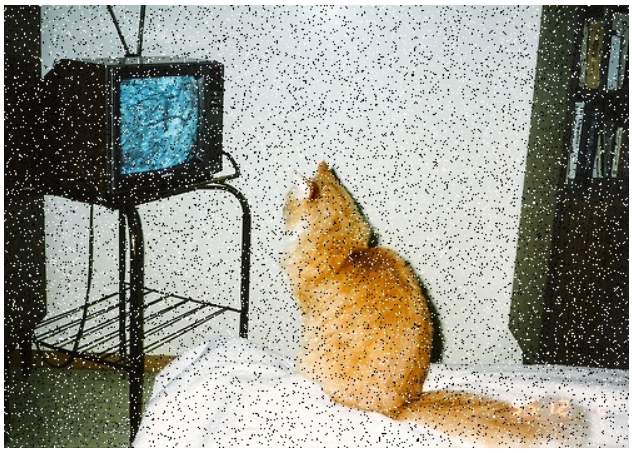
(a) *Original Image. SSIM = 1.0.*



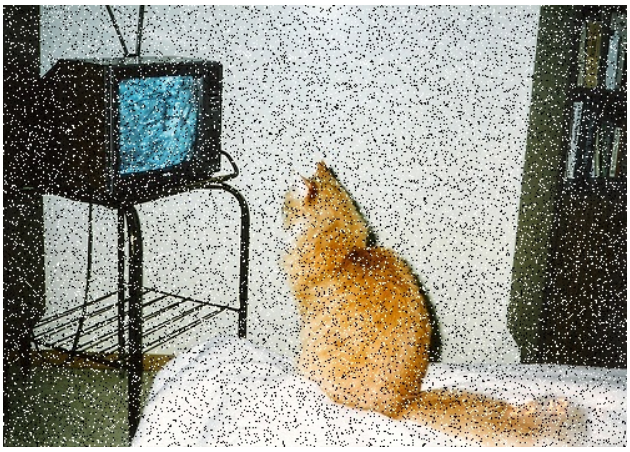
(b) *Probability = 0.01. SSIM = 0.6557.*



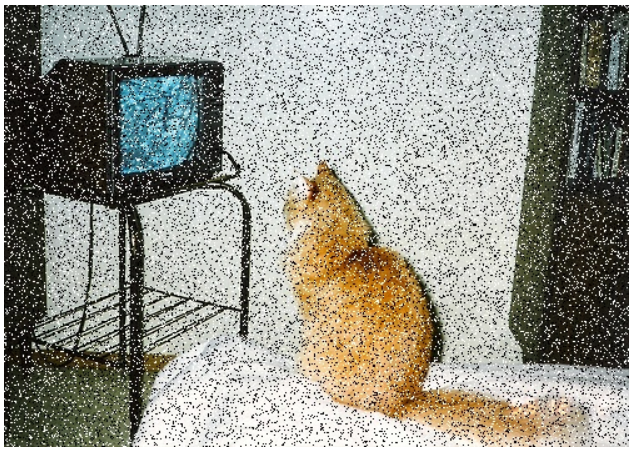
(c) *Probability = 0.025. SSIM = 0.3999.*



(d) *Probability = 0.05. SSIM = 0.2299.*



(e) *Probability = 0.075. SSIM = 0.1613.*



(f) *Probability = 0.1. SSIM = 0.1294.*

Figure 5.12: VOC2007 - salt-and-pepper sample images.

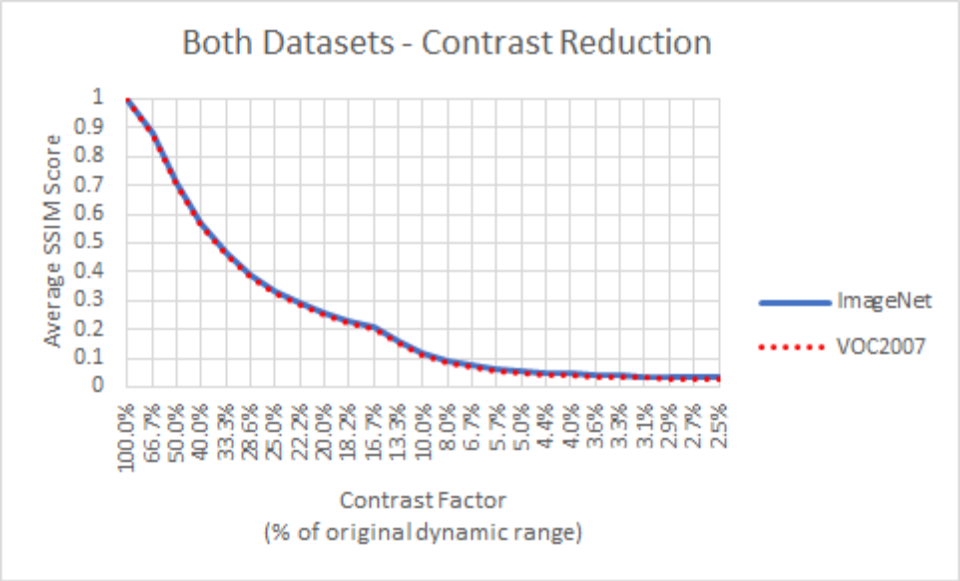


Figure 5.13: Comparison - contrast reduction. *The ImageNet and VOC2007 datasets have almost identical responses to the degradation. Therefore, the SSIM score for the contrast reduction degradation can be used as a predictor for CNN inference accuracy regardless of dataset.*

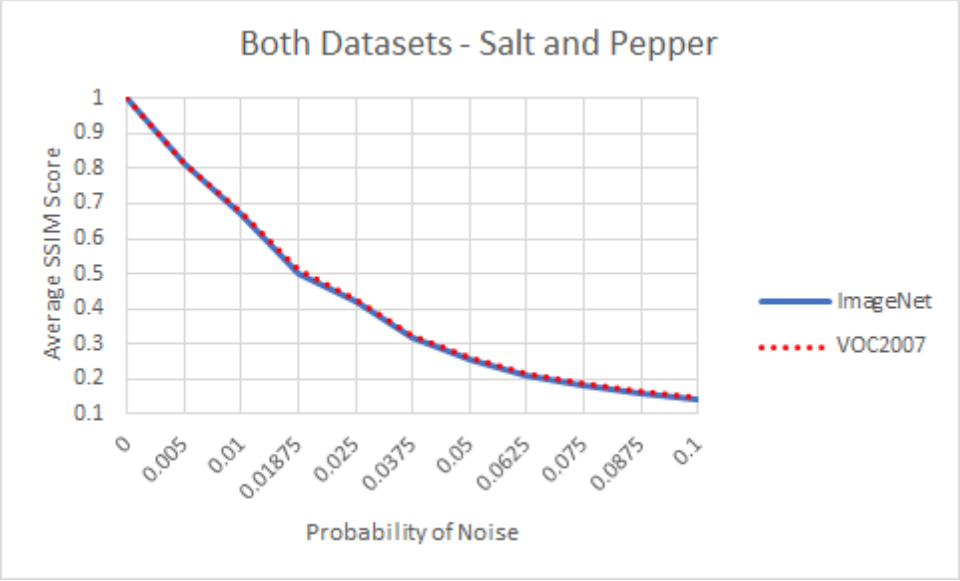


Figure 5.14: Comparison - salt & pepper. *This degradation type was the most degrading for both datasets. The ImageNet and VOC2007 datasets have almost identical responses. Therefore, the SSIM score for the salt-and-pepper degradation can be used as a predictor for CNN inference accuracy regardless of dataset.*

the associated corrective measure. For the degradation type of contrast reduction we have selected gamma correction as our method to compensate for frame quality degradation. For the salt-and-pepper degradation type we have selected the median filter as a means to correct.

As in the previous section, here we evaluate the effects of these corrections to the corresponding degradation types across the two image datasets of interest: ImageNet-2012 and VOC2007. Once again, plots in this section report the average SSIM score across ***all*** images in the dataset for the associated correction and degradation parameters. These plots will demonstrate to what degree, as reported by the SSIM score, the effects of a particular degradation can be mitigated in hopes of improving the performance of a CNN network.

Figure 5.15 provides a visual representation of the gamma correction applied to contrast reduction process. The original image in the top left is first split into its three color channels (red, green, blue). Each of the channels has the dynamic range reduced according to the desired percentage (in this example 50% contrast reduction was selected). Each degraded channel is then combined to create the resulting degraded color image as seen in the top right image. To correct the degradation, this image is then split into its three color channel images. Each color channel then is gamma corrected with the selected parameter (in this example an optimal gamma value of 1.2 is used). The corrected channels are then combined to produce the final corrected image as seen in the bottom left. The degraded image's SSIM score was 0.6881 but has a significant raise to 0.9661 after optimal gamma correction is applied. In our experiments we consider the parameters of 1.0 (no correction) to 2.0 for gamma correction.

Figure 5.16 provides a visual representation of the median filter correction applied to salt-and-pepper degradation process. The original image appears in the top left. The desired probability for salt-and-pepper is assigned and pixels are set to either white or black accordingly (in this example a noise probability of 0.01 was selected). The randomized noise is then combined with the original image to produce the salt-and-pepper degradation. Next, a median filter kernel size is selected to be used for correction (in this example we use a kernel size of 3x3). Finally, the degraded image has the selected median filter applied which produces the final corrected image as seen in the bottom left. The degraded image's SSIM score was 0.6557 but has a significant raise to 0.8388 after optimal median filter correction is applied.In our experiments we consider the kernels of sizes 3x3 to 11x11 for median filter correction.

The results of gamma correction applied to the contrast reduction degradation type on the ImageNet dataset can be seen in Figure 5.17. Contrast parameters of 66.7% to 2.5% coupled with gamma parameters of 1 to 2 are applied. When the gamma parameter is 1.0 this reflects no correction to the degradation. Note that for each curve there is a peak in SSIM score corresponding to an optimal gamma parameter value. For example, the red arrow points to the average SSIM score of 0.3338 for ImageNet when a contrast reduction of 25% is applied. However, the blue arrow points to when an optimal gamma correction of 1.4 is applied providing a corrected average SSIM score of 0.9191 - an increase of 0.5853. Additionally, a series of optimal gamma corrections applied to contrast reduction degradation parameters for a sample ImageNet image appear in Figure 5.18. These images demonstrate the effect of optimal gamma correction applied to the contrast reduction degradation for several parameters. For each pair it can be observed that gamma correction greatly mitigates the effect of the contrast reduction degradation.

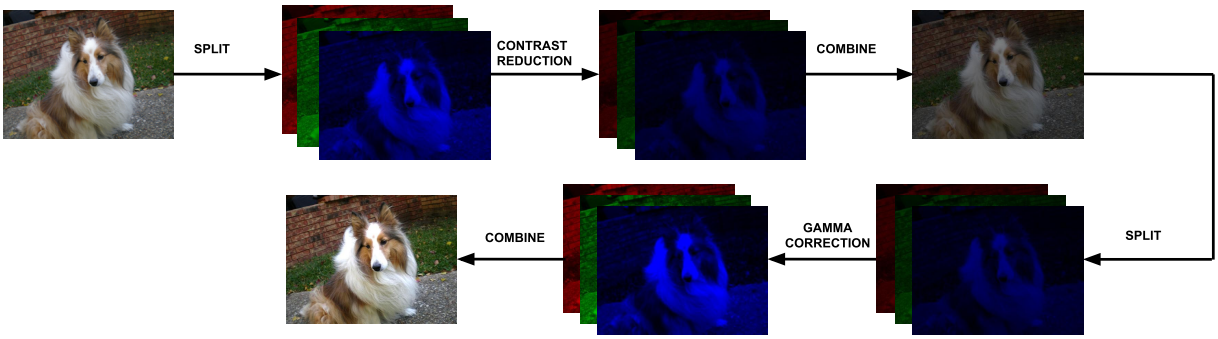


Figure 5.15: Process of gamma correction applied to contrast reduction degradation. *The original image in the top left is first split into its three color channels (red, green, blue). Each of the channels has the dynamic range reduced according to the desired percentage (in this example 50% contrast reduction was selected). Each degraded channel is then combined to create the resulting degraded color image as seen in the top right image. To correct the degradation, this image is then split into its three color channel images. Each color channel then is gamma corrected with the selected parameter (in this example an optimal gamma value of 1.2 is used). The corrected channels are then combined to produce the final corrected image as seen in the bottom left. The degraded image's SSIM score was 0.6881 but has a significant raise to 0.9661 after optimal gamma correction is applied.*

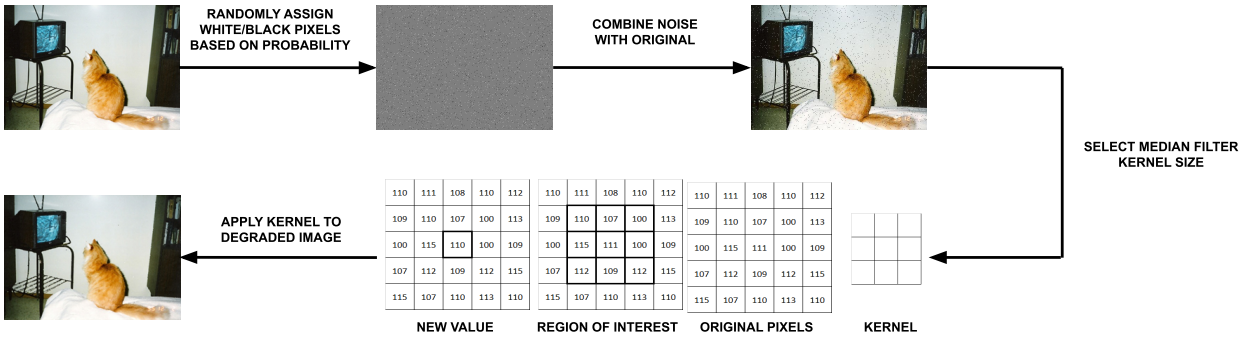


Figure 5.16: Process of median filter correction applied to salt-and-pepper degradation. *The original image appears in the top left. The desired probability for salt-and-pepper is assigned and pixels are set to either white or black accordingly (in this example a noise probability of 0.01 was selected). The randomized noise is then combined with the original image to produce the salt-and-pepper degradation. Next, a median filter kernel size is selected to be used for correction (in this example we use a kernel size of 3x3). Finally, the degraded image has the selected median filter applied which produces the final corrected image as seen in the bottom left. The degraded image's SSIM score was 0.6557 but has a significant raise to 0.8388 after optimal median filter correction is applied.*

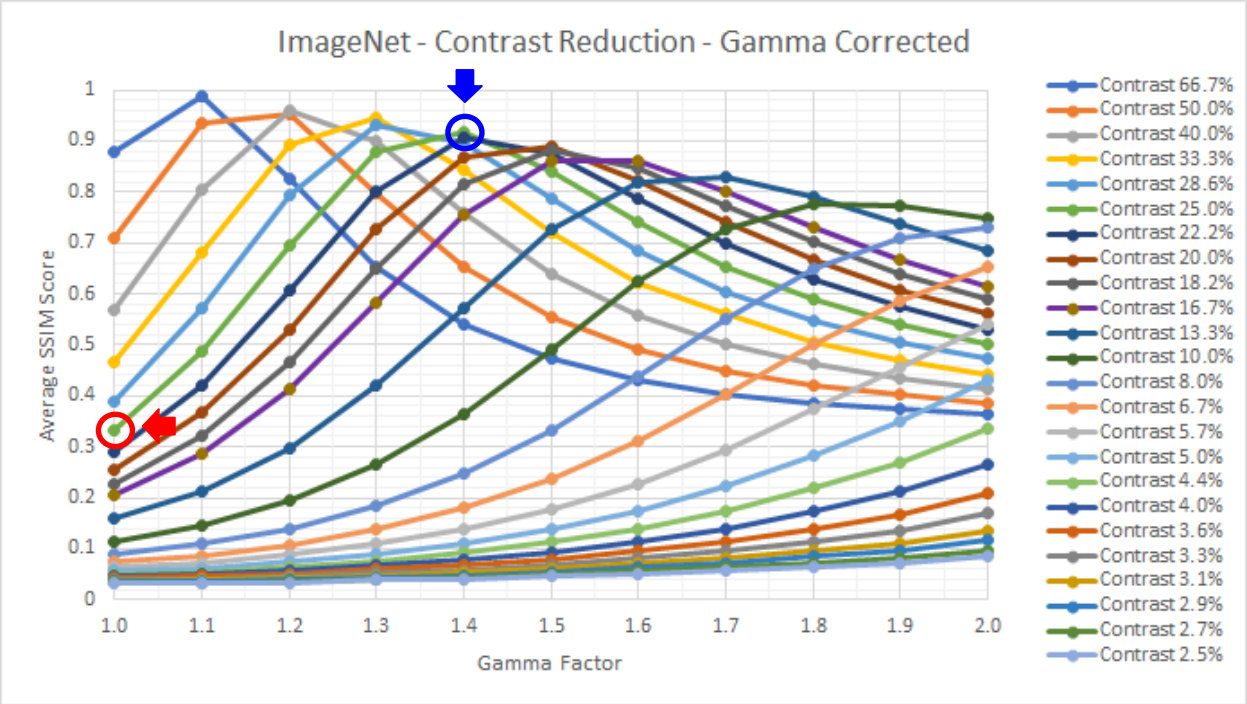


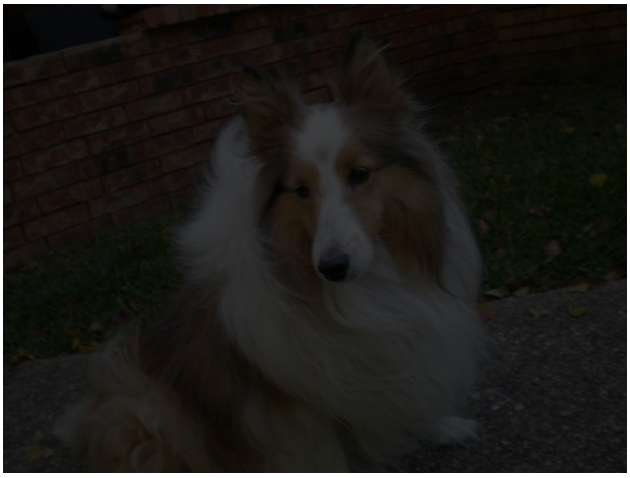
Figure 5.17: ImageNet - gamma correction applied to contrast reduction. *Each curve represents a single contrast reduction degradation for a specific parameter whose SSIM score is then altered by applying gamma correction. When the gamma parameter is 1.0 this reflects no correction to the degradation. Note that for each curve there is a peak in SSIM score corresponding to an optimal gamma parameter value. For example, the red arrow points to the average SSIM score of 0.3338 for ImageNet when a contrast reduction of 25% is applied. However, the blue arrow points to when an optimal gamma correction of 1.4 is applied providing a corrected average SSIM score of 0.9191 - an increase of 0.5853.*



(a) *Contrast = 50.0%. SSIM = 0.6881.*



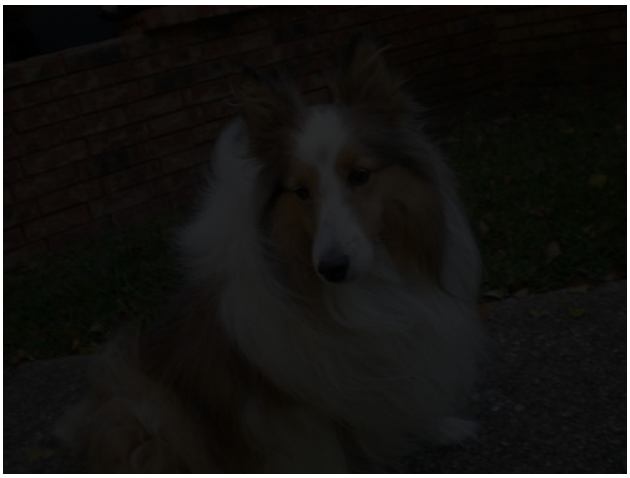
(b) *Gamma = 1.2 on Contrast 50.0%. SSIM = 0.9661.*



(c) *Contrast = 25.0%. SSIM = 0.3154.*



(d) *Gamma = 1.4 on Contrast 25.0%. SSIM = 0.9409.*



(e) *Contrast = 16.7%. SSIM = 0.1895.*



(f) *Gamma = 1.6 on Contrast 16.7%. SSIM = 0.8837.*

Figure 5.18: ImageNet - gamma correction applied to contrast reduction sample images.

The results of median filter applied to the salt-and-pepper degradation type on the ImageNet dataset can be seen in Figure 5.19. Salt-and-pepper parameters of 0.005 to 0.1 coupled with median filter kernel parameters of 3x3 to 11x11 are applied as well as "None" for when no correction is applied. Although our experiments used various kernel sizes a kernel size of 3x3 always provides the optimal correction. For example, the red arrow points to the average SSIM score of 0.2545 for ImageNet when a salt-and-pepper degradation with probability of 0.05 is applied. However, the blue arrow points to a 3x3 optimal median filter correction which provides a SSIM score of 0.8381 - an increase of 0.5836. In Figure 5.20 we consider no correction (i.e. "None") and the improvement of SSIM score with only a 3x3 median filter kernel for all considered probabilities of noise. Once again we explore the case of a salt-and-pepper degradation with probability 0.05 where the red arrow points to the average SSIM score of ImageNet for this degradation and the blue arrow points to the improvement in average SSIM score after a 3x3 median filter is applied. This figure displays the potential for very high quality recovery from images that are highly degraded from the salt-and-pepper noise. Additionally, a series of optimal median filter corrections applied to salt-and-pepper degradation parameters for a sample ImageNet image appear in Figure 5.21. These images demonstrate the effect of optimal median filter correction applied to the salt-and-pepper degradation for several parameters. For each pair it can be observed that median filter correction greatly mitigates the effect of the salt-and-pepper degradation.

The results of gamma correction applied to the contrast reduction degradation type on the VOC2007 dataset can be seen in Figure 5.22. Contrast parameters of 66.7% to 2.5% coupled with gamma parameters of 1 to 2 are applied. When the gamma parameter is 1.0 this reflects no correction to the degradation. Note that for each curve there is a peak in SSIM score corresponding to an optimal gamma parameter value. For example, the red arrow points to the average SSIM score of 0.3266 for VOC2007 when a contrast reduction of 25% is applied. However, the blue arrow points to when an optimal gamma correction of 1.4 is applied providing a corrected average SSIM score of 0.9115 - an increase of 0.5849. Additionally, a series of optimal gamma corrections applied to contrast reduction degradation parameters for a sample VOC2007 image appear in Figure 5.23. These images demonstrate the effect of optimal gamma correction applied to the contrast reduction degradation for several parameters. For each pair it can be observed that gamma correction greatly mitigates the effect of the contrast reduction degradation.

The results of median filter applied to the salt-and-pepper degradation type on the VOC2007 dataset can be seen in Figure 5.24. Salt-and-pepper parameters of 0.005 to 0.1 coupled with median filter kernel parameters of 3x3 to 11x11 are applied as well as "None" for when no correction is applied. Although our experiments used various kernel sizes a kernel size of 3x3 always provides the optimal correction. For example, the red arrow points to the average SSIM score of 0.2601 for VOC2007 when a salt-and-pepper degradation with probability of 0.05 is applied. However, the blue arrow points to a 3x3 optimal median filter correction which provides a SSIM score of 0.8075 - an increase of 0.5474. In Figure 5.25 we consider no correction (i.e. "None") and the improvement of SSIM score with only a 3x3 median filter kernel for all considered probabilities of noise. Once again we explore the case of a salt-and-pepper degradation with probability 0.05 where the red arrow points to the average SSIM score of VOC2007 for this degradation and the blue arrow points to the improvement in average SSIM score after a 3x3 median filter is applied. This figure displays

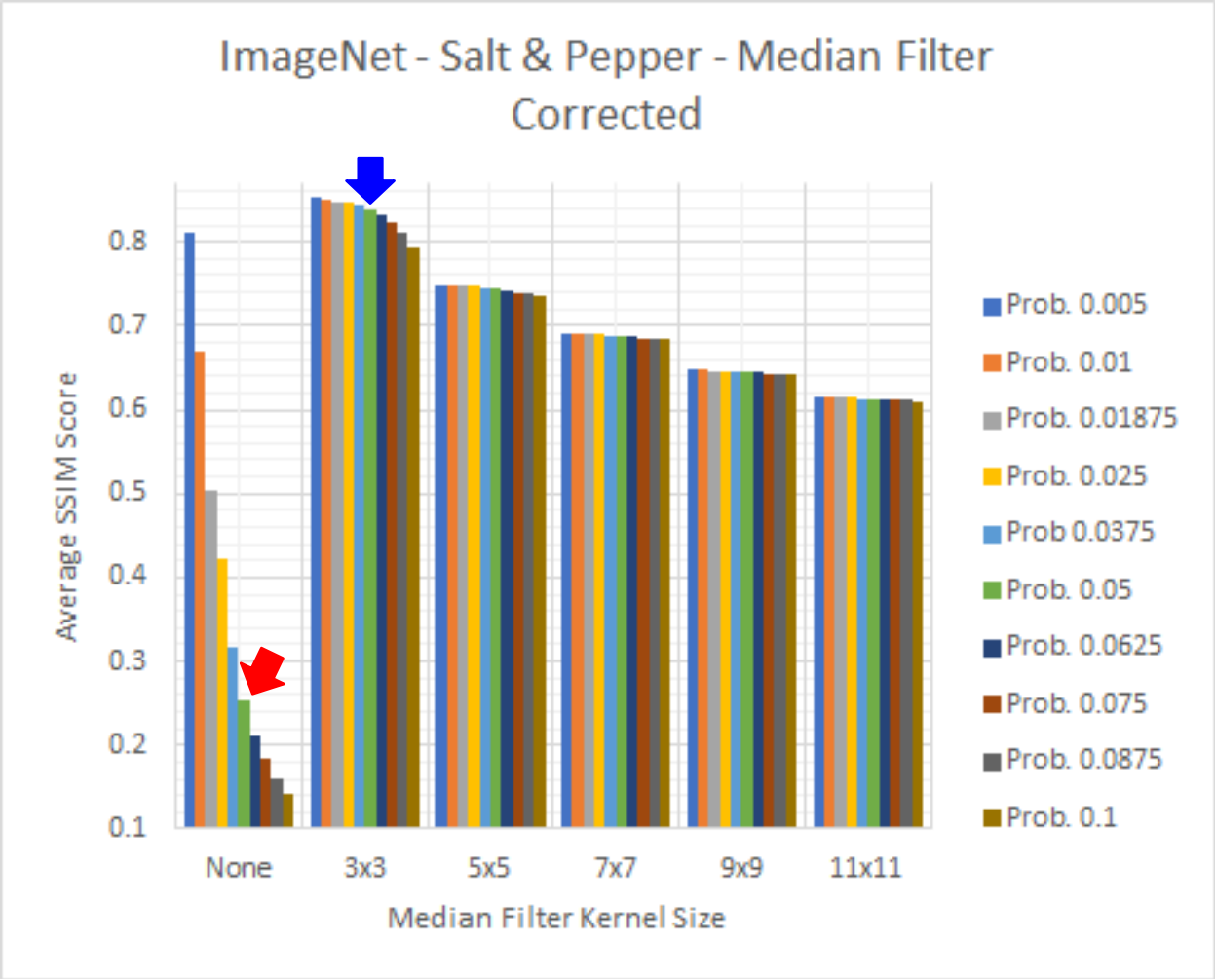


Figure 5.19: ImageNet - median filter correction applied to salt-and-pepper. *Each bar represents a single salt-and-pepper degradation for a specific parameter whose SSIM score is then altered by applying median filter correction. A median filter kernel size of "None" means no correction has been applied and appears here for reference purposes. Note that although our experiments used various kernel sizes a kernel size of 3x3 always provides the optimal correction. For example, the red arrow points to the average SSIM score of 0.2545 for ImageNet when a salt-and-pepper degradation with probability of 0.05 is applied. However, the blue arrow points to a 3x3 optimal median filter correction which provides a SSIM score of 0.8381 - an increase of 0.5836.*

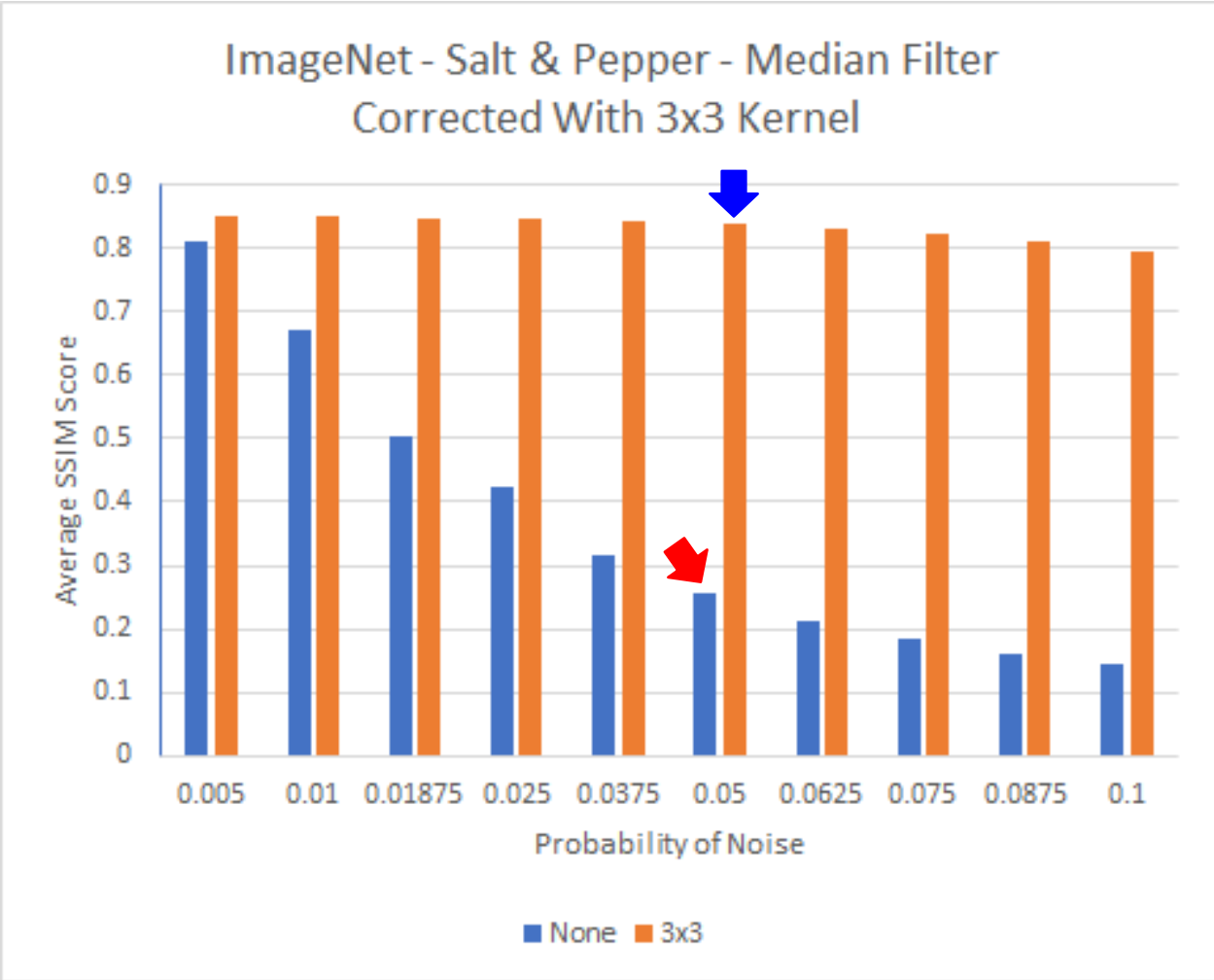


Figure 5.20: ImageNet - 3x3 median filter correction applied to salt-and-pepper. *For each probability of noise across the horizontal axis, we plot the SSIM score with no correction "None" as well as with 3x3 median filter kernel correction to illustrate the very high potential for correction from the salt-and-pepper degradation. For example, the red arrow points to the average SSIM score of 0.2545 for ImageNet when a salt-and-pepper degradation with probability of 0.05 is applied. However, the blue arrow points to a 3x3 optimal median filter correction which provides a SSIM score of 0.8381 - an increase of 0.5836.*



(a) *Prob. = 0.01. SSIM = 0.6223.*



(b) *Median = 3x3 on Prob. = 0.01. SSIM = 0.9293.*



(c) *Prob. = 0.05. SSIM = 0.1787.*



(d) *Median = 3x3 on Prob. = 0.05. SSIM = 0.9159.*



(e) *Prob. = 0.1. SSIM = 0.0806.*



(f) *Median = 3x3 on Prob. = 0.1. SSIM = 0.8655.*

Figure 5.21: ImageNet - median filter correction applied to salt-and-pepper sample images.

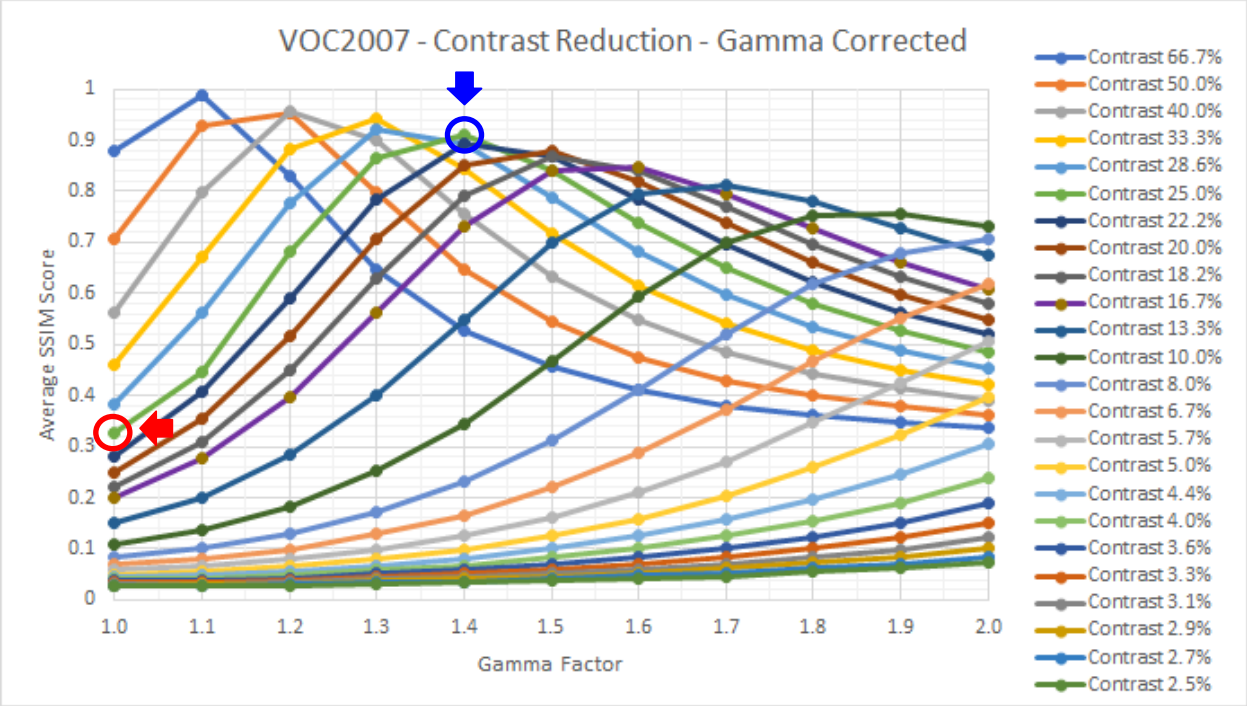
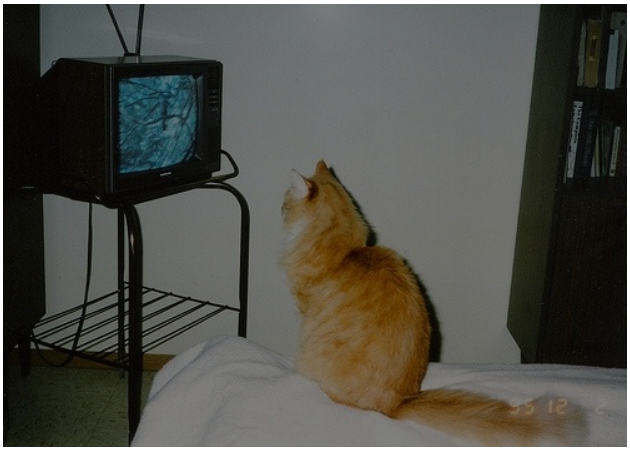
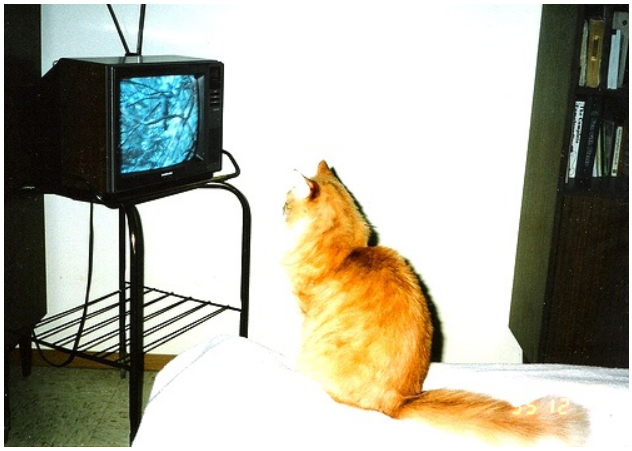


Figure 5.22: VOC2007 - gamma correction applied to contrast reduction. *Each curve represents a single contrast reduction degradation for a specific parameter whose SSIM score is then altered by applying gamma correction. When the gamma parameter is 1.0 this reflects no correction to the degradation. Note that for each curve there is a peak in SSIM score corresponding to an optimal gamma parameter value. For example, the red arrow points to the average SSIM score of 0.3266 for VOC2007 when a contrast reduction of 25% is applied. However, the blue arrow points to when an optimal gamma correction of 1.4 is applied providing a corrected average SSIM score of 0.9115 - an increase of 0.5849.*



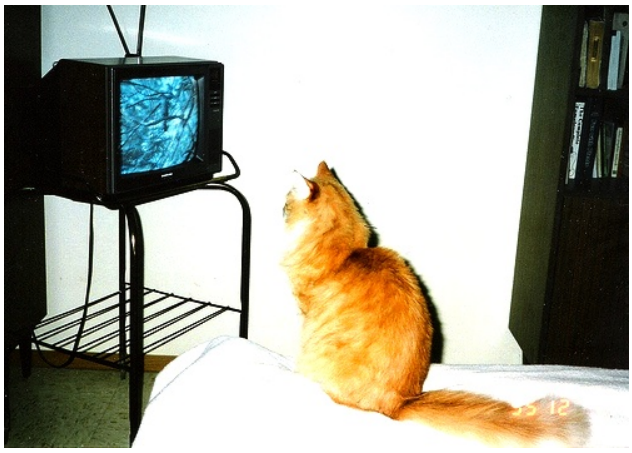
(a) *Contrast = 50.0%. SSIM = 0.7117.*



(b) *Gamma = 1.2 on Contrast 50.0%. SSIM = 0.9163.*



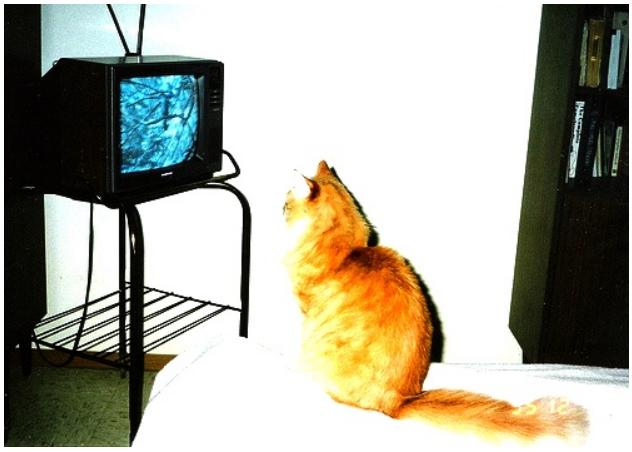
(c) *Contrast = 25.0%. SSIM = 0.3368.*



(d) *Gamma = 1.4 on Contrast 25.0%. SSIM = 0.8861.*



(e) *Contrast 16.7%. SSIM = 0.2107.*



(f) *Gamma = 1.6 on Contrast 16.7%. SSIM = 0.8267.*

Figure 5.23: VOC2007 - gamma correction applied to contrast reduction sample images.

the potential for very high quality recovery from images that are highly degraded from the salt-and-pepper noise. Additionally, a series of optimal median filter corrections applied to salt-and-pepper degradation parameters for a sample VOC2007 image appear in Figure 5.26. These images demonstrate the effect of optimal median filter correction applied to the salt-and-pepper degradation for several parameters. For each pair it can be observed that median filter correction greatly mitigates the effect of the salt-and-pepper degradation.

5.3 Findings From Image Degradation and Correction Experiments

In this chapter we evaluated the impact of two degradation methods (contrast reduction and salt-and-pepper noise) as well as their respective correction methods (gamma correction and median filter, respectively) on image quality for the ImageNet-2012 and VOC2007 datasets. In Table 5.1 we detail examples from both the contrast reduction and contrast reduction gamma corrected experiments as well as the salt-and-pepper and salt-and-pepper median filter corrected experiments. For each of the two datasets we provide the average SSIM score on degradation with a specific parameter as well as the improved average SSIM score with the optimal correction parameter. As demonstrated by the findings in this table, we show that significant improvement to image quality can be achieved even for highly degraded images if the optimal correction parameter is selected. Leveraging the optimal parameter for correction is key in the final chapter's experiments when we wish to maximize CNN inference accuracy. For both datasets contrast reduction beyond 5% and salt-and-pepper degradation beyond 0.1 probability were found to be difficult to correct such that any significant recovery in SSIM score could be achieved. This would indicate that after extreme image degradation that improvement for CNN inference accuracy is unlikely.

The experiments in this chapter show that the datasets (ImageNet-2012 and VOC2007) used in our evaluations did not contain any inherent bias to any of our selected degradation or correction methods. Additionally, these experiments demonstrate that we could implement real-world noises and correct the degradation, as well as provide a quantitative score to each image to provide statistical significance in our final experiments. We selected SSIM as our quantitative image quality metric to establish quality baseline to provide intuition on how these methods will ultimately impact the inference performance for the related CNNs evaluated in the following chapter. Based upon the findings from our experiments as detailed in Table 5.1, the higher the SSIM score for an image the higher the potential associated CNN inference accuracy can be predicted.

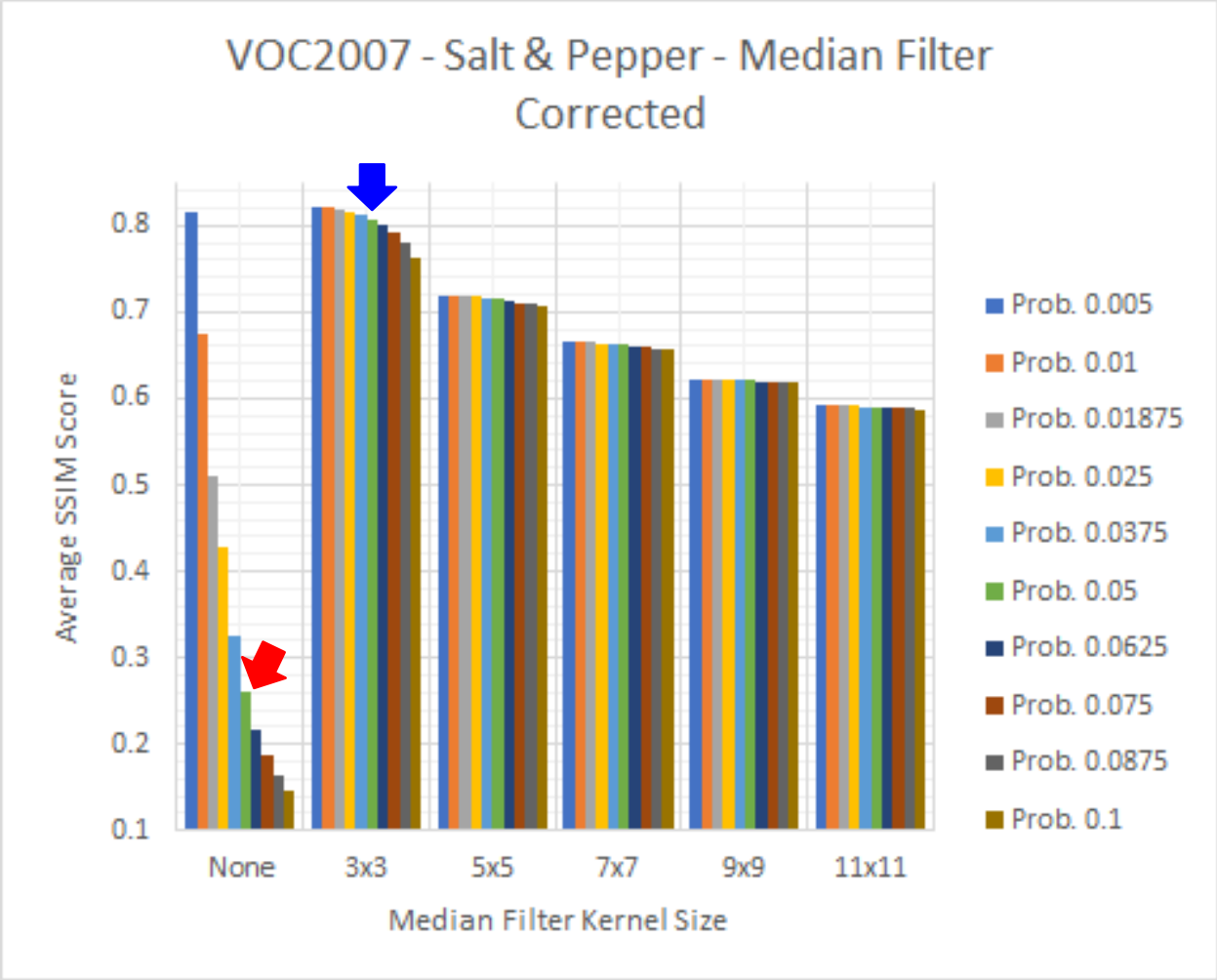


Figure 5.24: VOC2007 - median filter correction applied to salt-and-pepper. *Each bar represents a single salt-and-pepper degradation for a specific parameter whose SSIM score is then altered by applying median filter correction. A median filter kernel size of "None" means no correction has been applied and appears here for reference purposes. Note that although our experiments used various kernel sizes a kernel size of 3x3 always provides the optimal correction. For example, the red arrow points to the average SSIM score of 0.2601 for VOC2007 when a salt-and-pepper degradation with probability of 0.05 is applied. However, the blue arrow points to a 3x3 optimal median filter correction which provides a SSIM score of 0.8075 - an increase of 0.5474.*

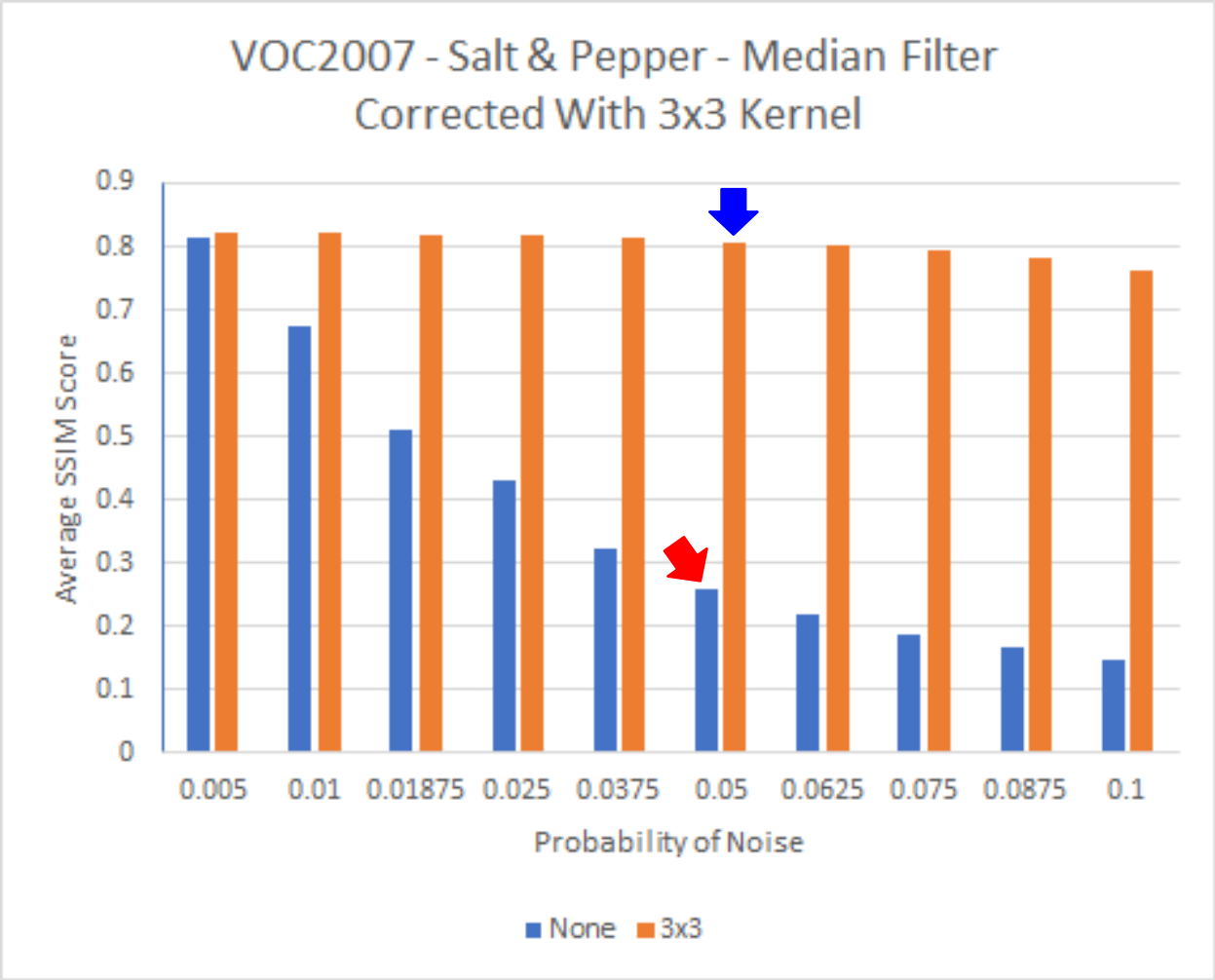
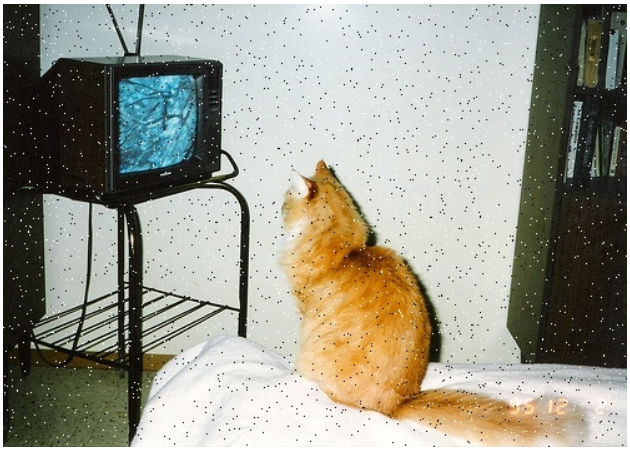


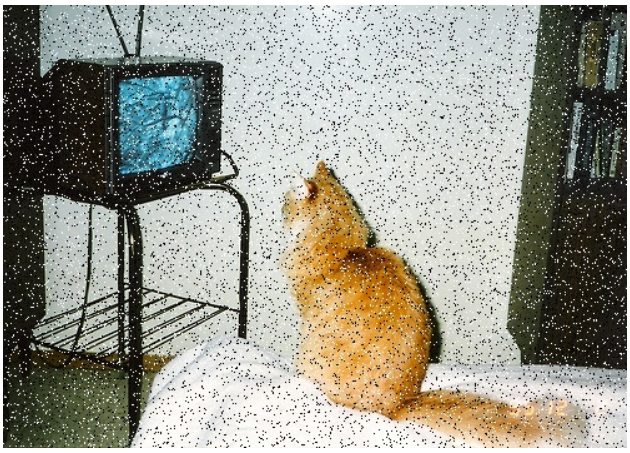
Figure 5.25: VOC2007 - 3x3 median filter correction applied to salt-and-pepper. *For each probability of noise across the horizontal axis, we plot the SSIM score with no correction "None" as well as with 3x3 median filter kernel correction to illustrate the very high potential for correction from the salt-and-pepper degradation. For example, the red arrow points to the average SSIM score of 0.2601 for VOC2007 when a salt-and-pepper degradation with probability of 0.05 is applied. However, the blue arrow points to a 3x3 optimal median filter correction which provides a SSIM score of 0.8075 - an increase of 0.5474.*



(a) *Prob. = 0.01. SSIM = 0.6531.*



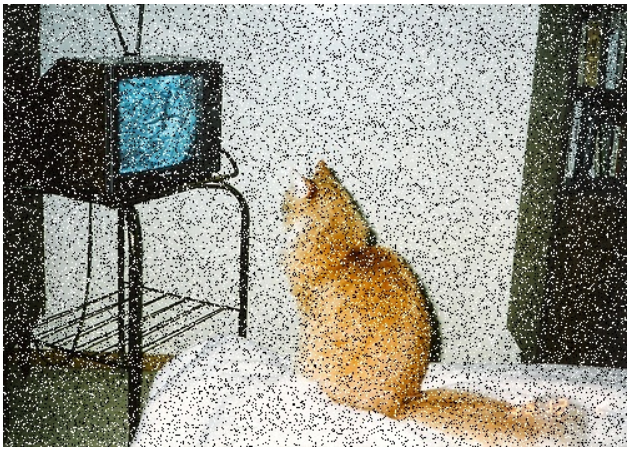
(b) *Median = 3x3 on Prob. = 0.01. SSIM = 0.8388.*



(c) *Prob. = 0.05. SSIM = 0.2256.*



(d) *Median = 3x3 on Prob. = 0.05. SSIM = 0.8259.*



(e) *Prob. = 0.1. SSIM = 0.1303.*



(f) *Median = 3x3 on Prob. = 0.1. SSIM = 0.7827.*

Figure 5.26: VOC2007 - median filter correction applied to salt-and-pepper sample images.

Table 5.1: Comparison of experimental findings for image degradation and correction on ImageNet-2012 and VOC2007 datasets. *This table details examples from both the contrast reduction and contrast reduction gamma corrected experiments as well as the salt-and-pepper and salt-and-pepper median filter corrected experiments. For each of the two datasets we provide the average SSIM score on degradation with a specific parameter as well as the improved average SSIM score with the optimal correction parameter. Leveraging the optimal parameter for correction is key in the final chapter's experiments when we wish to maximize CNN inference accuracy. For both datasets contrast reduction beyond 5% and salt-and-pepper degradation beyond 0.1 probability were found to be difficult to correct such that any significant recovery in SSIM score could be achieved. This would indicate that after extreme image degradation that improvement for CNN inference accuracy is unlikely.*

Experiment	**ImageNet SSIM Score**	**ImageNet SSIM Score Improvement**	**VOC2007 SSIM Score**	**VOC2007 SSIM Score Improvement**
Contrast 50.0%	0.7077	N/A	0.7077	N/A
Contrast 50.0% optimal gamma	0.9529	0.2452	0.9533	0.2456
Contrast 25.0%	0.3338	N/A	0.3266	N/A
Contrast 25.0% optimal gamma	0.9191	0.5853	0.9115	0.5849
Contrast 10.0%	0.1153	N/A	0.1081	N/A
Contrast 10.0% optimal gamma	0.7774	0.6621	0.7551	0.6470
Prob. 0.005	0.8109	N/A	0.8141	N/A
Prob. 0.005 optimal median	0.8519	0.041	0.8219	0.0078
Prob. 0.05	0.2545	N/A	0.2601	N/A
Prob. 0.05 optimal median	0.8381	0.5836	0.8075	0.5474
Prob. 0.1	0.1431	N/A	0.1468	N/A
Prob 0.1 optimal median	0.7942	0.6511	0.7637	0.6169

Chapter 6

Full Pipeline Experimental Benchmarking Results

This chapter compares and evaluates the performance of the full thesis pipeline as seen in figure 1.2 which benchmarks our hardware implemented CNNs against their traditional GPU implementation while addressing model robustness to input quality. As with previous sections, we evaluate our ResNet-50 GPU and FPGA models with the ImageNet-2012 dataset and our YOLO version 3 GPU and FPGA models with the VOC2007 datasets. The experiments in this final chapter prove that our FPGA solution is a better candidate for Edge deployment given the similar robustness of the quantized models as compared to the floating point models when evaluating degraded images, as well as similar robustness to the corresponding recovery of performance when attempting to correct degradations before inference. Therefore, this chapter expands upon the experiments in Chapters 3, 4, and 5 by evaluating image degradation and correction across the two complete hardware pipelines (GPU and FPGA) for the two network models (ResNet-50 and YOLO version 3) to provide comparative benchmarking results.

6.1 Pipeline 1: ResNet-50 Evaluation

The ResNet-50 pipeline is evaluated and compared on both our GPU and FPGA implementations. As with the base model evaluation in Chapter 4, we use the 5,000 image ImageNet testing set for the ResNet-50 experiments. First, we explore the impact of two degradations: contrast reduction and salt-and-pepper noise, upon both GPU and FPGA models. We analyze how these noises impact the separate hardware implementations both individually as well as a side-by-side comparison. As in Chapter 4, we use top-1 and top-5 accuracy as our metric for this evaluation. Next, we evaluate how the associated correction technique for each degradation impacts performance. As before, we explore the GPU and FPGA implementations separately and then perform a comparison. For the correction analysis and comparison we have chosen to only explore the top-1 accuracy as our metric.

6.1.1 Comparison of Contrast Reduction Degradation on GPU and FPGA

The results of the contrast reduction degradation upon the GPU implementation of ResNet-50 as assessed on the ImageNet testing set can be seen in Figure 6.1. With no contrast reduction (a contrast parameter of 100.0%) the top-1 accuracy is observed to be 74.44% and the top-5 accuracy achieves 91.85%. These reported accuracies with no contrast reduction introduced to the testing dataset provide baseline inference accuracy for reference. Both the top-1 and top-5 accuracy decrease minimally until contrast reduction is set to 40.0% (accuracy becoming 71.0% and 89.54%, respectively). This minimal drop in accuracy for these contrast reduction parameters is due to the CNN architecture's modest robustness to contrast changes. Between the contrast reduction parameters of 40.0% and 3.3% the top-1 and top-5 accuracy decreases quickly and at similar rates. In this range of parameters where the model's accuracy is more impacted is ideal for corrective measures to boost accuracy. After a contrast reduction of 2.5% the decrease in accuracy plateaus and is not reported here. Images and video frames degraded beyond 2.5% are shown to be visually improvable with optimal correction but not to a degree such that desirable CNN accuracy can be achieved.

The results of the contrast reduction degradation upon the FPGA implementation of ResNet-50 as assessed on the ImageNet testing set can be seen in Figure 6.2. With no contrast reduction (a contrast parameter of 100.0%) the top-1 accuracy is observed to be 73.34% and the top-5 accuracy achieves 91.31%. These reported accuracies with no contrast reduction introduced to the testing dataset provide baseline inference accuracy for reference. Both the top-1 and top-5 accuracy decrease minimally until contrast reduction is set to 40.0% (accuracy becoming 67.06% and 87.36%, respectively). As with the GPU model, this minimal drop in accuracy for these contrast reduction parameters is due to the CNN architecture's modest robustness to contrast changes. Between the contrast reduction parameters of 40.0% and 4.4% the top-1 and top-5 accuracy decreases quickly and at similar rates. However, the rate of this decrease in accuracy is sharper for the FPGA model as compared to the GPU model. After a contrast reduction of 2.5% the decrease in accuracy plateaus and is not reported here. Images and video frames degraded beyond 2.5% are shown to be visually improvable with optimal correction but not to a degree such that desirable CNN accuracy can be achieved.

Figure 6.3 compares the top-1 accuracy of the GPU implementation of ResNet-50 versus the FPGA counterpart when degrading via the contrast reduction noise. The GPU implementation has a top-1 accuracy of 74.44% as compared to the FPGA implementation of 73.34% when no contrast reduction degradation is present (contrast parameter of 100.0%) showing a mere drop of 1.1% in FPGA accuracy for the baseline performance. Between the parameters of 40.0% and 3.3% we see a sharper rate of decrease in inference accuracy for both hardware implementations. Although there is only an approximate 1% percent point difference with unaltered images, we observe that the FPGA accuracy falls off at a quicker rate than the GPU implementation in this range. The most likely reason for this difference in degradation impact is the nature of the two different model weights. Although the 32-bit floating point weights in the GPU model require higher SWaP characteristics to implement they are more robust to the quantized integer weights of the FPGA model.

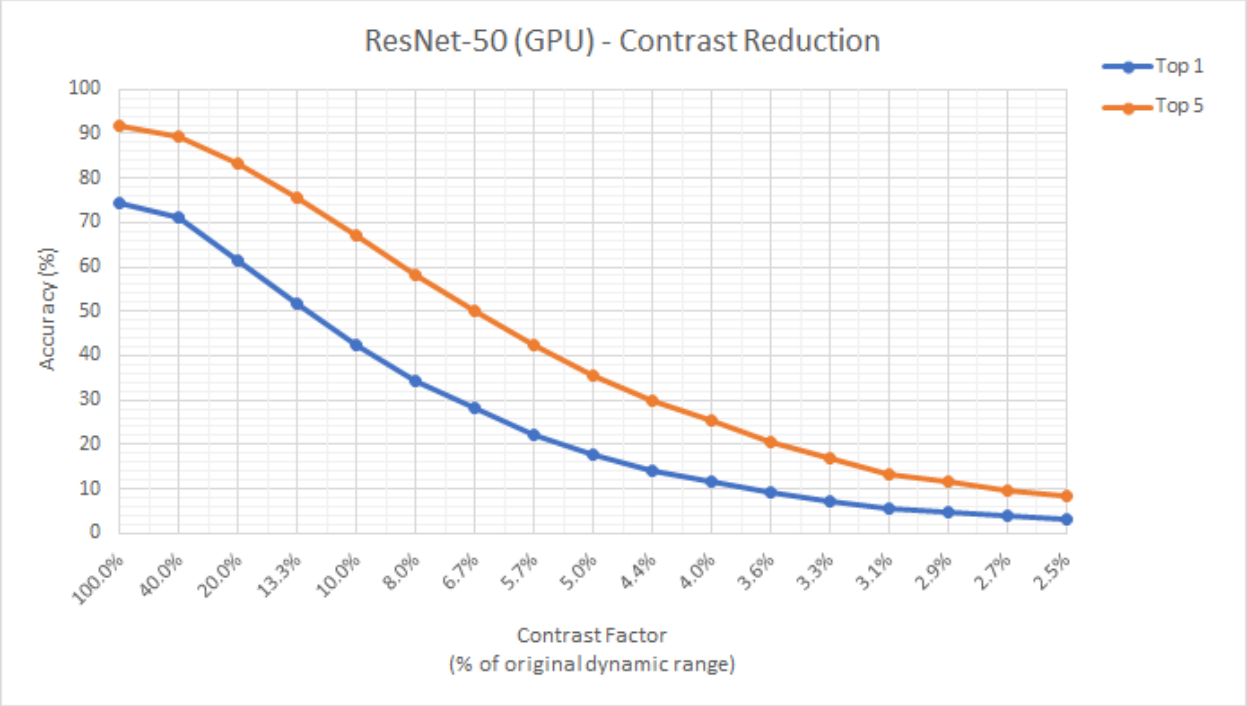


Figure 6.1: Results of contrast reduction degradation on ResNet-50 GPU implementation accuracy for the ImageNet testing set (5,000 images). *With no contrast reduction (a contrast parameter of 100.0%) the top-1 accuracy is observed to be 74.44% and the top-5 accuracy achieves 91.85% which reflects baseline performance for reference. Both the top-1 and top-5 accuracy decrease minimally until contrast reduction is set to 40.0% (accuracy becoming 71.0% and 89.54%, respectively). Between the contrast reduction parameters of 40.0% and 3.3% the top-1 and top-5 accuracy decreases quickly and at similar rates. After a contrast reduction of 2.5% the decrease in accuracy plateaus and is not reported here. Images and video frames degraded beyond 2.5% are shown to be visually improvable with optimal correction but not to a degree such that desirable CNN accuracy can be achieved.*

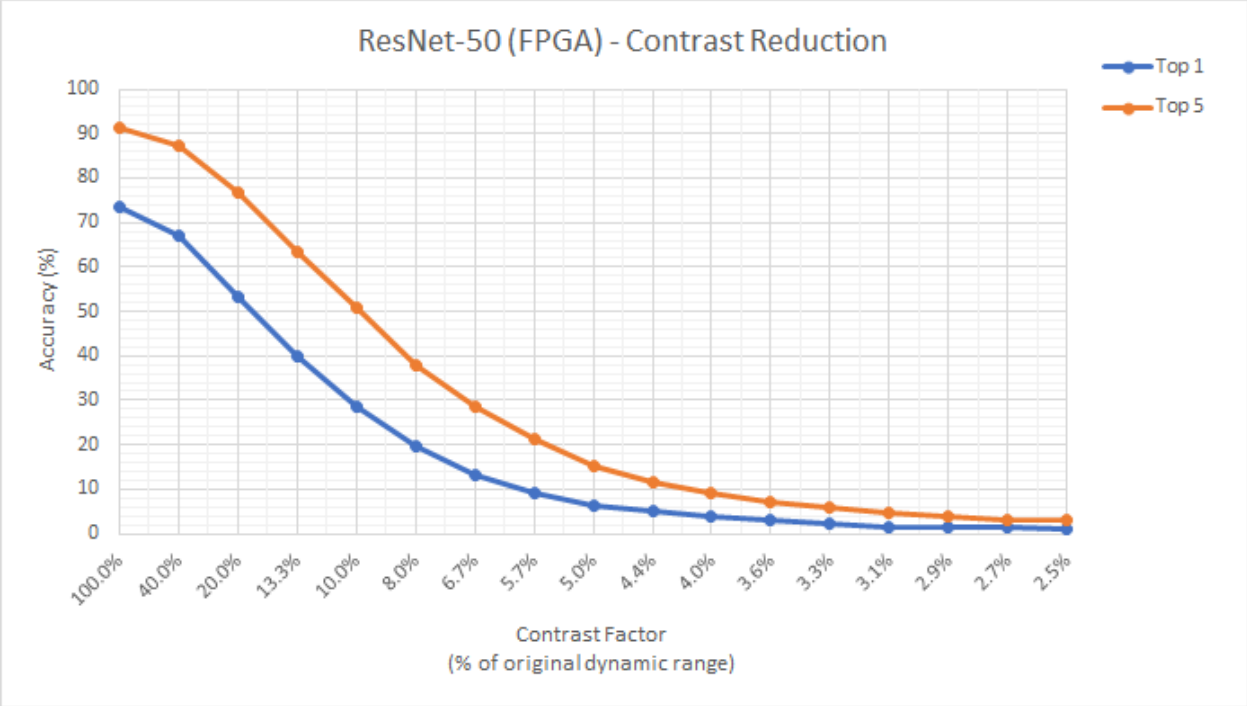


Figure 6.2: Results of contrast reduction degradation on ResNet-50 FPGA implementation accuracy for the ImageNet testing set (5,000 images). *With no contrast reduction (a contrast parameter of 100.0%) the top-1 accuracy is observed to be 73.34% and the top-5 accuracy achieves 91.31% which reflects baseline performance for reference. As with the GPU model, both the top-1 and top-5 accuracy decrease minimally until contrast reduction is set to 40.0% (accuracy becoming 67.06% and 87.36%, respectively). Between the contrast reduction parameters of 40.0% and 4.4% the top-1 and top-5 accuracy decreases quickly and at similar rates. However, the rate of this decrease in accuracy is sharper for the FPGA model as compared to the GPU model. After a contrast reduction of 2.5% the decrease in accuracy plateaus and is not reported here. Images and video frames degraded beyond 2.5% are shown to be visually improvable with optimal correction but not to a degree such that desirable CNN accuracy can be achieved.*

However, as detailed later, the FPGA model's robustness to contrast degradation can be greatly improved with optimal correction measures.

Figure 6.4 compares the top-5 accuracy of the GPU implementation of ResNet-50 versus the FPGA counterpart when degrading via the contrast reduction noise. The GPU implementation has a top-5 accuracy of 91.85% as compared to the FPGA implementation of 91.31% when no contrast reduction degradation is present (contrast parameter of 100.0%) showing a mere drop of 0.54% in FPGA accuracy for the baseline performance. Between the parameters of 40.0% and 3.3% we see a sharper rate of decrease in inference accuracy for both hardware implementations. Although there is only an approximate 1% percent point difference with unaltered images, we observe that the FPGA accuracy falls off at a quicker rate than the GPU implementation in this range. The most likely reason for this difference in degradation impact is the nature of the two different model weights. Although the 32-bit floating point weights in the GPU model require higher SWaP characteristics to implement they are more robust to the quantized integer weights of the FPGA model. However, as detailed later, the FPGA model's robustness to contrast degradation can be greatly improved with optimal correction measures.

Figure 6.5 displays a side-by-side comparison of both the top-1 accuracies and the top-5 accuracies for the FPGA and GPU models when degraded by contrast reduction. The set of blue lines (solid for GPU and dashed for FPGA) plot the top-5 accuracies when contrast reduction is applied. The set of red lines (solid for GPU and dashed for FPGA) plot the top-1 accuracies when contrast reduction is applied. With no contrast reduction (a contrast parameter of 100.0%) the top-1 and top-5 accuracies for both the FPGA and GPU models have an approximate 1% difference and are shown here for a baseline reference. For both cases, the FPGA model accuracy is shown to be more impacted by contrast reduction between the 40.0% and 3.3% parameters as compared to the GPU model. However, this degradation can be corrected with optimally applied methods which are detailed in a subsequent section.

Finally, in this section comparing the results of contrast reduction degradation between the GPU and FPGA models, we provide visual examples from the ImageNet testing set to illustrate our statistical findings. We chose a contrast reduction parameter of 20.0% to accommodate easy human readability in these example images. Figure 6.6 provides an example testing image degraded with contrast reduction where both the GPU and FPGA models still made the proper prediction (lens cap). Figure 6.7 has the GPU predicting the correct class (bee) with the contrast reduced image but the FPGA model predicting incorrectly (tarantula). Figure 6.8 shows an instance where both the GPU and FPGA provided incorrect inferences of the degraded image to the class of lighthouse - megalith and radio telescope, respectively. To the best of our knowledge, there were no instances where the FPGA model provided a correct inference for a degraded image that the GPU model predicted incorrectly. This finding further provides evidence that differences between the inference accuracies of the two models arise from the lower precision weights leveraged by the FPGA model. However, as detailed in Chapter 4 and reviewed at the end of this section, the lower precision weights result in a slightly less accurate model but provide the hardware platform for significant gains in SWaP parameters to maximize Edge performance.

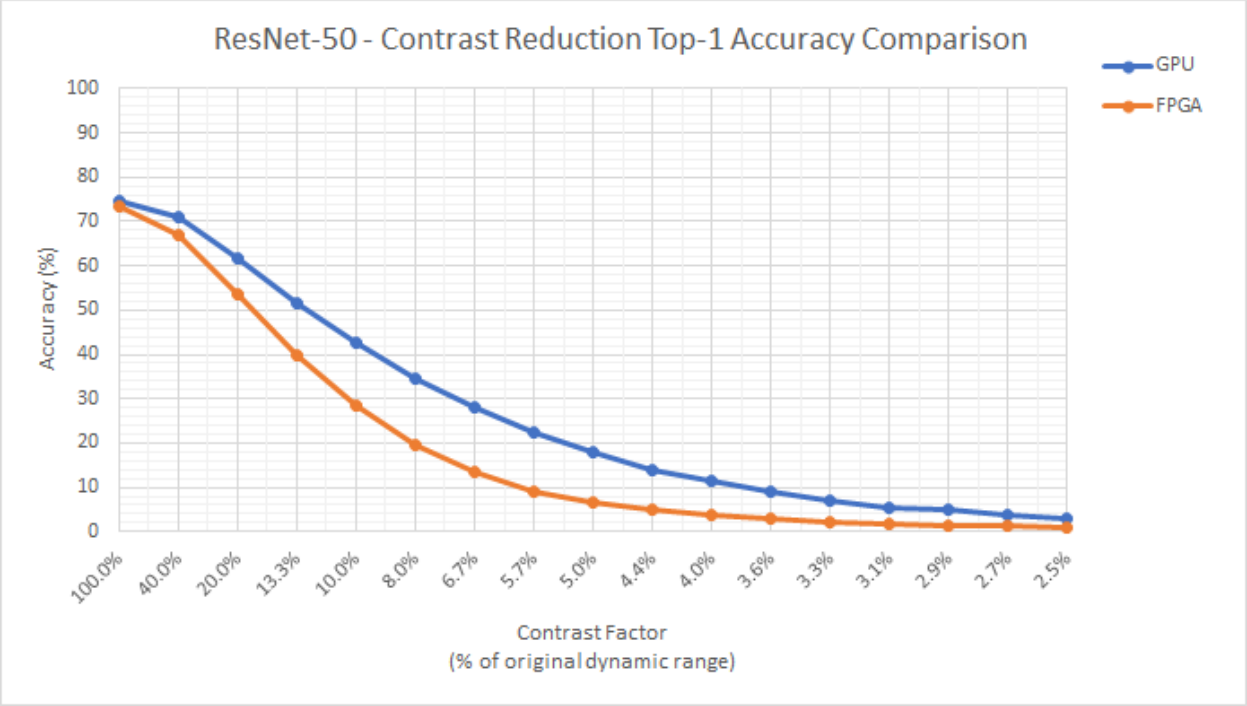


Figure 6.3: Comparison of the results of contrast reduction degradation on top-1 accuracy for ResNet-50 on GPU versus FPGA implementation for the ImageNet testing set (5,000 images). *With no contrast reduction (a contrast parameter of 100.0%) the top-1 accuracy for the GPU model (blue line) is observed to be 74.44% and the top-1 accuracy for the FPGA model (orange line) achieves 73.34% (a difference of 1.1%) which reflects baseline performance for reference. As previously noted, the decrease in accuracy for both models is minimal up to a contrast parameter of 40.0%. Between the parameters of 40.0% and 3.3% we see a sharper rate of decrease in inference accuracy for both hardware implementations, however the FPGA model is shown to be more impacted by the contrast reduction degradation. This higher rate of accuracy degradation for the FPGA implementation is due to the less precision of the 8-bit integer weights it leverages as compared to the 32-bit floating point weights of the GPU model. However, this increase in weight precision and thus accuracy comes at the cost of much greater SWaP requirements for deployment. For both hardware models, beyond a contrast reduction parameter of 2.5% the accuracy performance plateaus and is not reported here.*

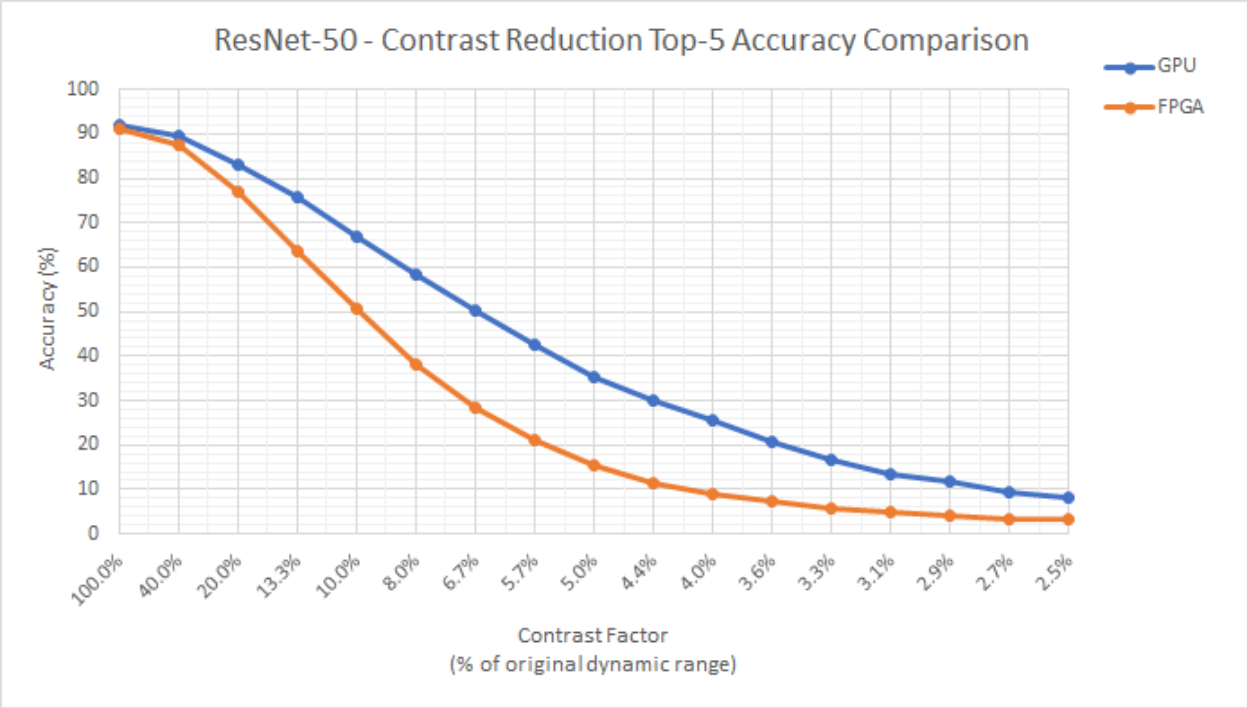


Figure 6.4: Comparison of the results of contrast reduction degradation on top-5 accuracy for ResNet-50 on GPU versus FPGA implementation for the ImageNet testing set (5,000 images). *With no contrast reduction (a contrast parameter of 100.0%) the top-5 accuracy for the GPU model (blue line) is observed to be 91.85% and the top-5 accuracy for the FPGA model (orange line) achieves 91.31% (a difference of 0.54%) which reflects baseline performance for reference. As previously noted, the decrease in accuracy for both models is minimal up to a contrast parameter of 40.0%. Between the parameters of 40.0% and 3.3% we see a sharper rate of decrease in inference accuracy for both hardware implementations, however the FPGA model is shown to be more impacted by the contrast reduction degradation. This higher rate of accuracy degradation for the FPGA implementation is due to the less precision of the 8-bit integer weights it leverages as compared to the 32-bit floating point weights of the GPU model. However, this increase in weight precision and thus accuracy comes at the cost of much greater SWaP requirements for deployment. For both hardware models, beyond a contrast reduction parameter of 2.5% the accuracy performance plateaus and is not reported here.*

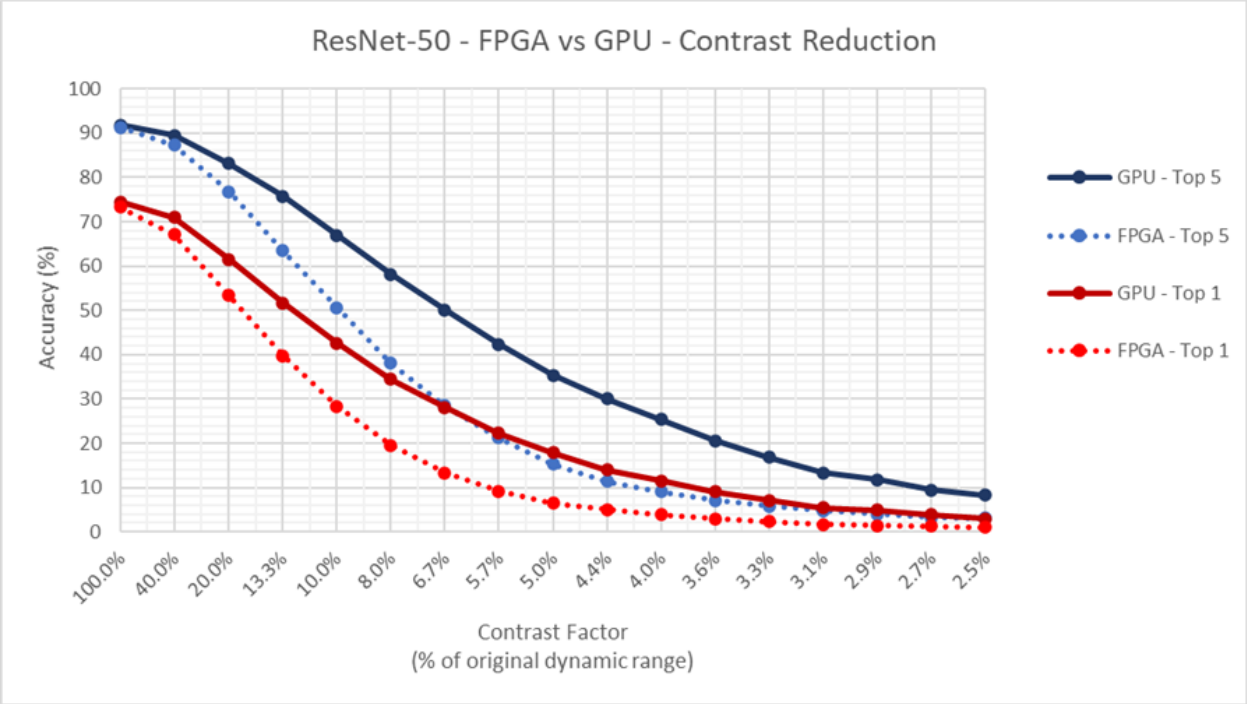


Figure 6.5: Side-by-side comparison of the results of contrast reduction degradation on both top-1 and top-5 accuracy for ResNet-50 on GPU versus FPGA implementation. *The set of blue lines (solid for GPU and dashed for FPGA) plot the top-5 accuracies when contrast reduction is applied. The set of red lines (solid for GPU and dashed for FPGA) plot the top-1 accuracies when contrast reduction is applied. With no contrast reduction (a contrast parameter of 100.0%) the top-1 and top-5 accuracies for both the FPGA and GPU models have an approximate 1% difference and are shown here for a baseline reference. For both cases, the FPGA model accuracy is shown to be more impacted by contrast reduction between the 40.0% and 3.3% parameters as compared to the GPU model. However, this degradation can be corrected with optimally applied methods.*



(a) *Original.*



(b) *GPU Inference.*



(c) *FPGA Inference.*

Figure 6.6: ResNet-50 example testing image with contrast reduction degradation at 20%. *The original image from the ImageNet testing set appears in the top. Both GPU and FPGA models make the correct inference (lens cap). For this particular instance the degradation level was not severe enough to cause a mislabeling for either hardware model.*



(a) *Original.*



(b) *GPU Inference.*



(c) *FPGA Inference.*

Figure 6.7: ResNet-50 example testing image with contrast reduction degradation at 20%. *The original image from the ImageNet testing set appears in the top. The GPU model provides correct inference (bee) but the FPGA model is incorrect (tarantula). In this instance the higher precision of the GPU model weights appear to be more robust to this degree of contrast reduction.*



(a) *Original.*



(b) *GPU Inference.*



(c) *FPGA Inference.*

Figure 6.8: ResNet-50 example testing image with contrast reduction degradation at 20%. *The original image from the ImageNet testing set appears in the top. Both the GPU and FPGA models provide incorrect inference for the class lighthouse. Both models provide the correct level before degradation, however, at 20% contrast reduction neither model can successfully classify the image.*

6.1.2 Comparison of Salt-and-Pepper Degradation on GPU and FPGA

The results of the salt-and-pepper degradation upon the GPU implementation of ResNet-50 as assessed on the ImageNet testing set can be seen in Figure 6.9. With no salt-and-pepper degradation (a parameter of 0) the top-1 accuracy is observed to be 74.44% and the top-5 accuracy achieves 91.85%. These reported accuracies with no salt-and-pepper introduced to the testing dataset provide baseline inference accuracy for reference. Unlike with contrast reduction, the impact of salt-and-pepper degradation upon accuracy is significant at even small parameters. With a probability of noise parameter of 0.005 the top-1 accuracy goes from 74.44% to 62.84% (a drop of 11.6%) and the top-5 accuracy goes from 91.85% to 83.58% (a drop of 8.27%). Therefore the CNN architecture appears to have little inherent robustness to salt-and-pepper degradation. For our experiments we cut off the probability of noise parameter to 0.1, where for the GPU the top-1 accuracy achieves 19.9% and the top-5 accuracy achieves 40.48%. Additionally, one can observe that the top-1 and top-5 accuracies drop off with a near identical rate as the parameter increases.

The results of the salt-and-pepper degradation upon the FPGA implementation of ResNet-50 as assessed on the ImageNet testing set can be seen in Figure 6.10. With no salt-and-pepper degradation (a parameter of 0) the top-1 accuracy is observed to be 73.34% and the top-5 accuracy achieves 91.31%. These reported accuracies with no salt-and-pepper introduced to the testing dataset provide baseline inference accuracy for reference. Unlike with contrast reduction, the impact of salt-and-pepper degradation upon accuracy is significant at even small parameters. With a probability of noise parameter of 0.005 the top-1 accuracy goes from 73.34% to 62.7% (a drop of 10.64%) and the top-5 accuracy goes from 91.31% to 83.5% (a drop of 7.81%). Therefore the CNN architecture appears to have little inherent robustness to salt-and-pepper degradation. For our experiments we cut off the probability of noise parameter to 0.1, where for the FPGA the top-1 accuracy achieves 19.14% and the top-5 accuracy achieves 39.42%. Additionally, one can observe that the top-1 and top-5 accuracies drop off with a near identical rate as the parameter increases - exactly as with the GPU model. Therefore, both the GPU model and FPGA model appear to be impacted in an identical manner to this sort of noise. The most likely explanation for this is that since salt-and-pepper degradation is a discreet operation instead of continuous (as is contrast reduction) the integer based weights of the FPGA model react nearly identically to the floating point based weights of the GPU model.

Figure 6.11 compares the top-1 accuracy of the GPU implementation of ResNet-50 versus the FPGA counterpart when degrading via the salt-and-pepper noise. The GPU implementation has a top-1 accuracy of 74.44% as compared to the FPGA implementation of 73.34% when no salt-and-pepper degradation is present (parameter of 0) showing a mere drop of 1.1% in FPGA accuracy for the baseline performance. With a probability of noise parameter of 0.005 the top-1 accuracy for GPU 74.44% to 62.84% (a drop of 11.6%) and the top-1 accuracy for FPGA goes from 73.34% to 62.7% (a drop of 10.64%) showing that the impact of salt-and-pepper degradation upon accuracy is significant at even small parameters. Unlike with the contrast reduction degradation, the FPGA implementation appears to have a similar degree of robustness to this degradation type as compared to the GPU implementation. The rate of accuracy decrease is almost identical between the two

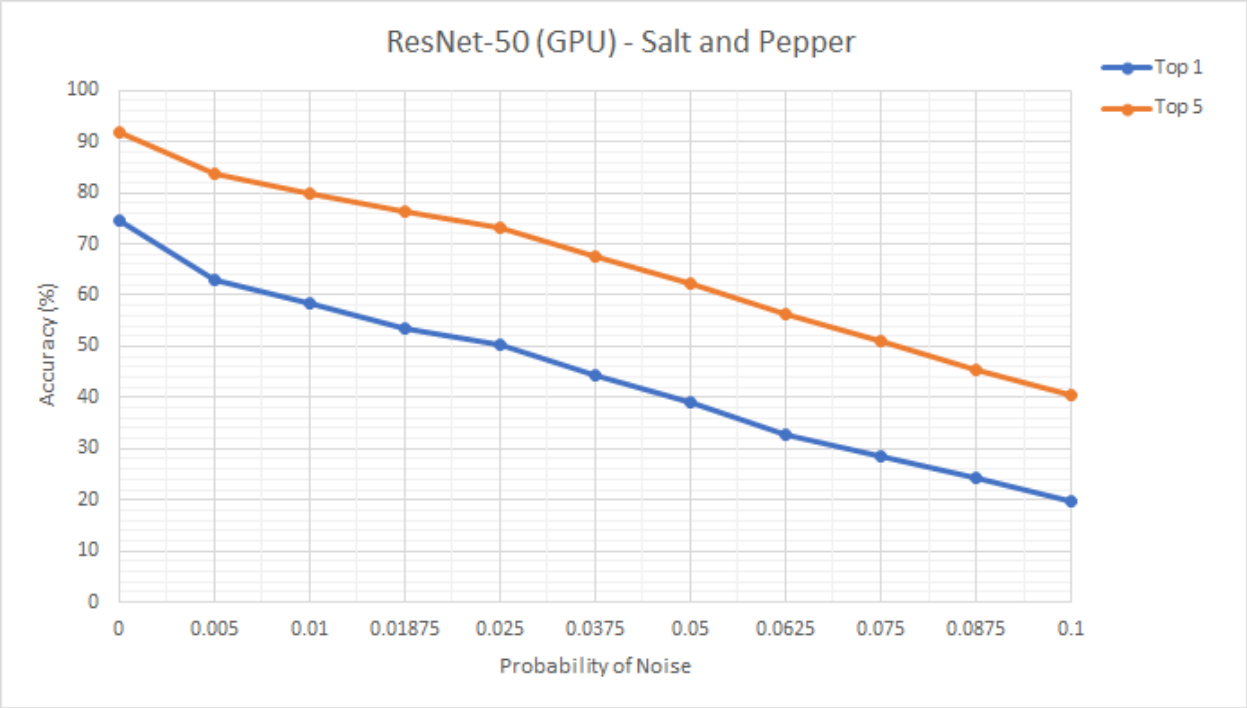


Figure 6.9: Results of salt-and-pepper degradation on ResNet-50 GPU implementation accuracy for the ImageNet testing set (5,000 images). *With no salt-and-pepper degradation (a parameter of 0) the top-1 accuracy is observed to be 74.44% and the top-5 accuracy achieves 91.85% which reflects baseline performance for reference. The impact of salt-and-pepper degradation upon accuracy is significant at even small parameters. With a probability of noise parameter of 0.005 the top-1 accuracy goes from 74.44% to 62.84% (a drop of 11.6%) and the top-5 accuracy goes from 91.85% to 83.58% (a drop of 8.27%). For our experiments we cut off the probability of noise parameter to 0.1, where the GPU top-1 accuracy achieves 19.9% and the top-5 accuracy achieves 40.48%.*

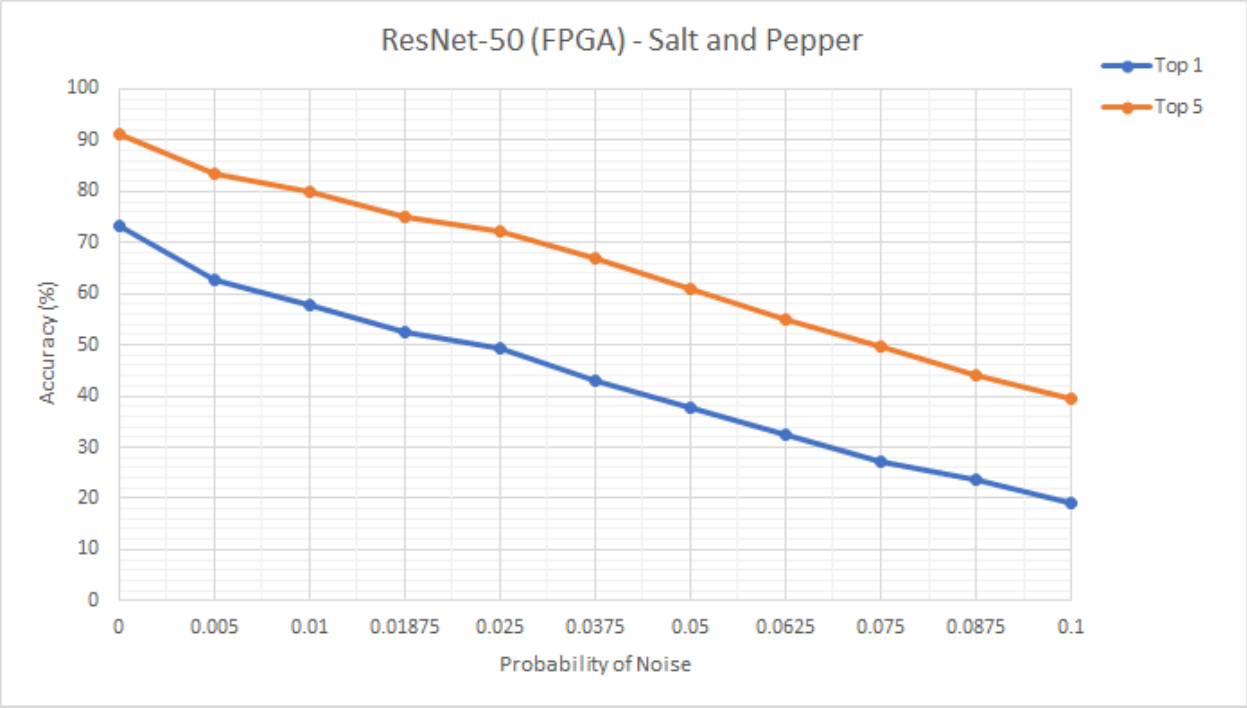


Figure 6.10: Results of salt-and-pepper degradation on ResNet-50 FPGA implementation accuracy for the ImageNet testing set (5,000 images). *With no salt-and-pepper degradation (a parameter of 0) the top-1 accuracy is observed to be 73.34% and the top-5 accuracy achieves 91.31% which reflects baseline performance for reference. The impact of salt-and-pepper degradation upon accuracy is significant at even small parameters. With a probability of noise parameter of 0.005 the top-1 accuracy goes from 73.34% to 62.7% (a drop of 10.64%) and the top-5 accuracy goes from 91.31% to 83.5% (a drop of 7.81%). For our experiments we cut off the probability of noise parameter to 0.1, where the FPGA top-1 accuracy achieves 19.14% and the top-5 accuracy achieves 39.42%.*

implementations with the FPGA accuracy maintaining an approximate 1% drop at all salt-and-pepper parameters. Thus, the FPGA model has similar robustness to salt-and-pepper degradation model as the GPU will being delivered with superior SWaP characteristics.

Figure 6.12 compares the top-5 accuracy of the GPU implementation of ResNet-50 versus the FPGA counterpart when degrading via the salt-and-pepper noise. The GPU implementation has a top-5 accuracy of 91.85% as compared to the FPGA implementation of 91.31% when no salt-and-pepper degradation is present (parameter of 0) showing a mere drop of 0.54% in FPGA accuracy for the baseline performance. With a probability of noise parameter of 0.005 the top-5 accuracy for GPU 91.85% to 83.58% (a drop of 8.27%) and the top-5 accuracy for FPGA goes from 91.31% to 83.5% (a drop of 7.81%) showing that the impact of salt-and-pepper degradation upon accuracy is significant at even small parameters. Once again when considering the top-5 accuracy, the FPGA implementation appears to have a similar degree of robustness to this degradation type as compared to the GPU implementation. The rate of accuracy decrease is almost identical between the two implementations with the FPGA accuracy maintaining an approximate 1% drop at all salt-and-pepper parameters.

Figure 6.13 displays a side-by-side comparison of both the top-1 accuracies and the top-5 accuracies for the FPGA and GPU models when degraded by salt-and-pepper. The set of blue lines (solid for GPU and dashed for FPGA) plot the top-5 accuracies when salt-and-pepper degradation is applied. The set of red lines (solid for GPU and dashed for FPGA) plot the top-1 accuracies when salt-and-pepper degradation is applied. With no salt-and-pepper degradation (a probability of noise parameter of 0) the top-1 and top-5 accuracies for both the FPGA and GPU models have an approximate 1% difference and are shown here for a baseline reference. The rate of accuracy decrease is almost identical between the two implementations with the FPGA accuracy maintaining an approximate 1% drop at all salt-and-pepper parameters. However, just as with the contrast reduction, this degradation can be corrected with optimally applied methods which are detailed in a subsequent section.

Finally, in this section comparing the results of salt-and-pepper degradation between the GPU and FPGA models, we provide visual examples from the ImageNet testing set to illustrate our statistical findings. We chose a probability parameter of 0.05 to accommodate easy human readability in these examples images. Figure 6.14 provides an example testing image where both the GPU and FPGA models still made the proper prediction (bee). Figure 6.15 has the GPU predicting the correct class (tent) with the salt-and-pepper degraded image but the FPGA model predicting incorrectly (pinwheel). Figure 6.16 shows an instance where both the GPU and FPGA provided incorrect inferences to the class of snowplow - tow truck and jeep, respectively. To the best of our knowledge, there were no instances where the FPGA model provided a correct inference for a degraded image that the GPU model predicted incorrectly. As with our contrast reduction experiment on ResNet-50 hardware models, this finding further provides evidence that differences between the inference accuracies of the two models arise from the lower precision weights leveraged by the FPGA model. However, as detailed in Chapter 4 and reviewed at the end of this section, the lower precision weights result in a slightly less accurate model but provide the hardware platform for significant gains in SWaP parameters to maximize Edge performance.

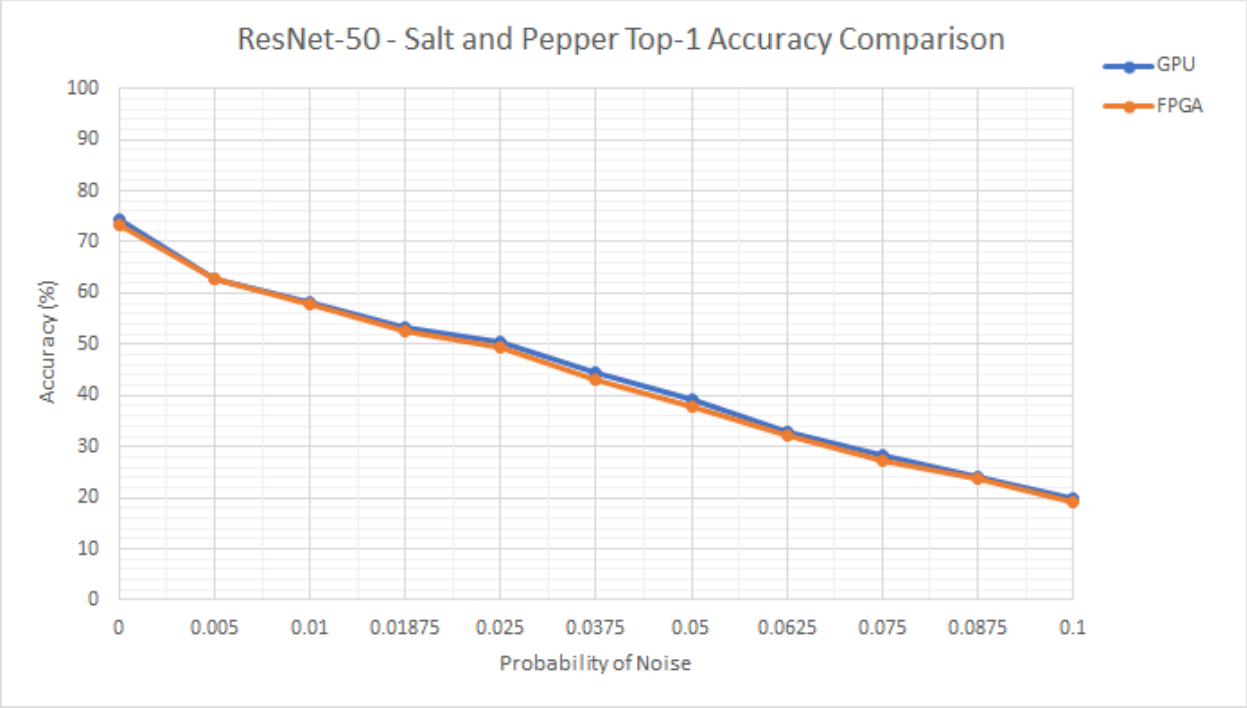


Figure 6.11: Comparison of the results of salt-and-pepper degradation on top-1 accuracy for ResNet-50 on GPU versus FPGA implementation for the ImageNet testing set (5,000 images). *With no salt-and-pepper degradation (a probability of noise parameter of 0) the top-1 accuracy for the GPU model (blue line) is observed to be 74.44% and the top-1 accuracy for the FPGA model (orange line) achieves 73.34% (a difference of 1.1%) which reflects baseline performance for reference. With a probability of noise parameter of 0.005 the top-1 accuracy for GPU 74.44% to 62.84% (a drop of 11.6%) and the top-1 accuracy for FPGA goes from 73.34% to 62.7% (a drop of 10.64%) showing that the impact of salt-and-pepper degradation upon accuracy is significant at even small parameters. Unlike with the contrast reduction degradation, the FPGA implementation appears to have a similar degree of robustness to salt-and-pepper degradation as compared to the GPU implementation. The rate of accuracy decrease is almost identical between the two implementations with the FPGA accuracy maintaining an approximate 1% drop at all salt-and-pepper parameters.*

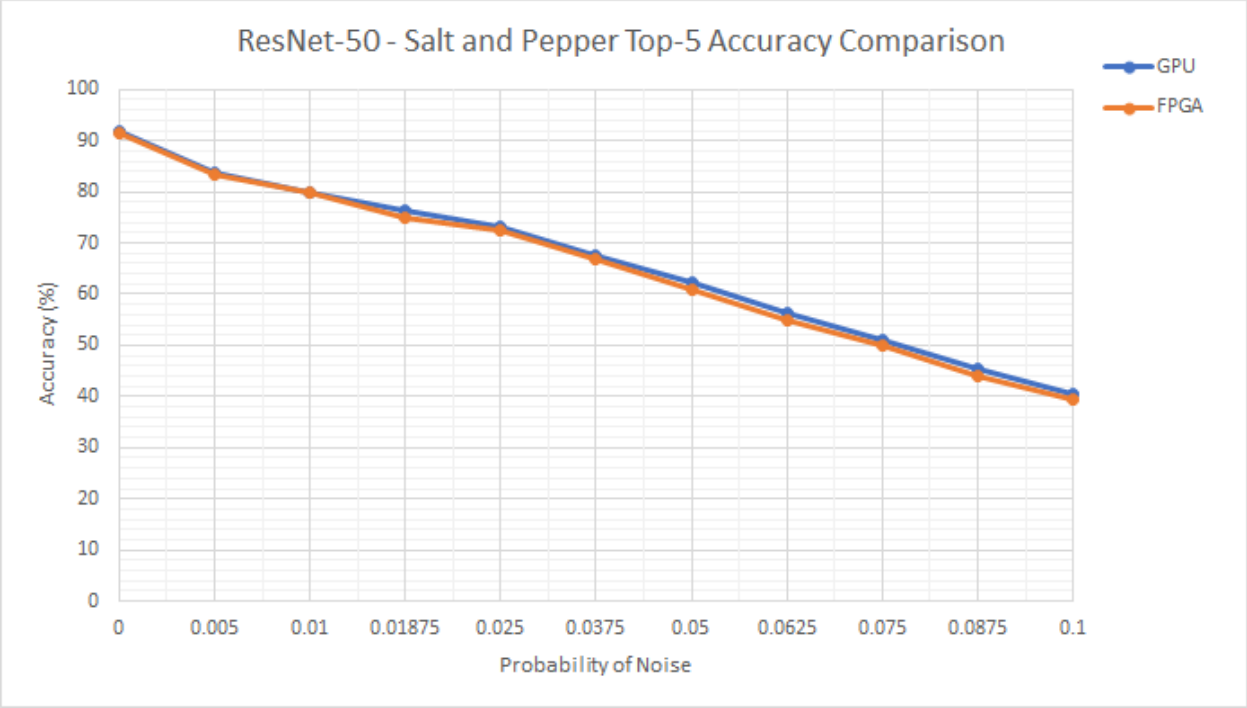


Figure 6.12: Comparison of the results of salt-and-pepper degradation on top-5 accuracy for ResNet-50 on GPU versus FPGA implementation for the ImageNet testing set (5,000 images). *With no salt-and-pepper degradation (a probability of noise parameter of 0) the top-5 accuracy for the GPU model (blue line) is observed to be 91.85% and the top-5 accuracy for the FPGA model (orange line) achieves 91.31% (a difference of 0.54%) which reflects baseline performance for reference. With a probability of noise parameter of 0.005 the top-5 accuracy for GPU 91.85% to 83.58% (a drop of 8.27%) and the top-5 accuracy for FPGA goes from 91.31% to 83.5% (a drop of 7.81%) showing that the impact of salt-and-pepper degradation upon accuracy is significant at even small parameters. Unlike with the contrast reduction degradation, the FPGA implementation appears to have a similar degree of robustness to salt-and-pepper degradation as compared to the GPU implementation. The rate of accuracy decrease is almost identical between the two implementations with the FPGA accuracy maintaining an approximate 1% drop at all salt-and-pepper parameters.*

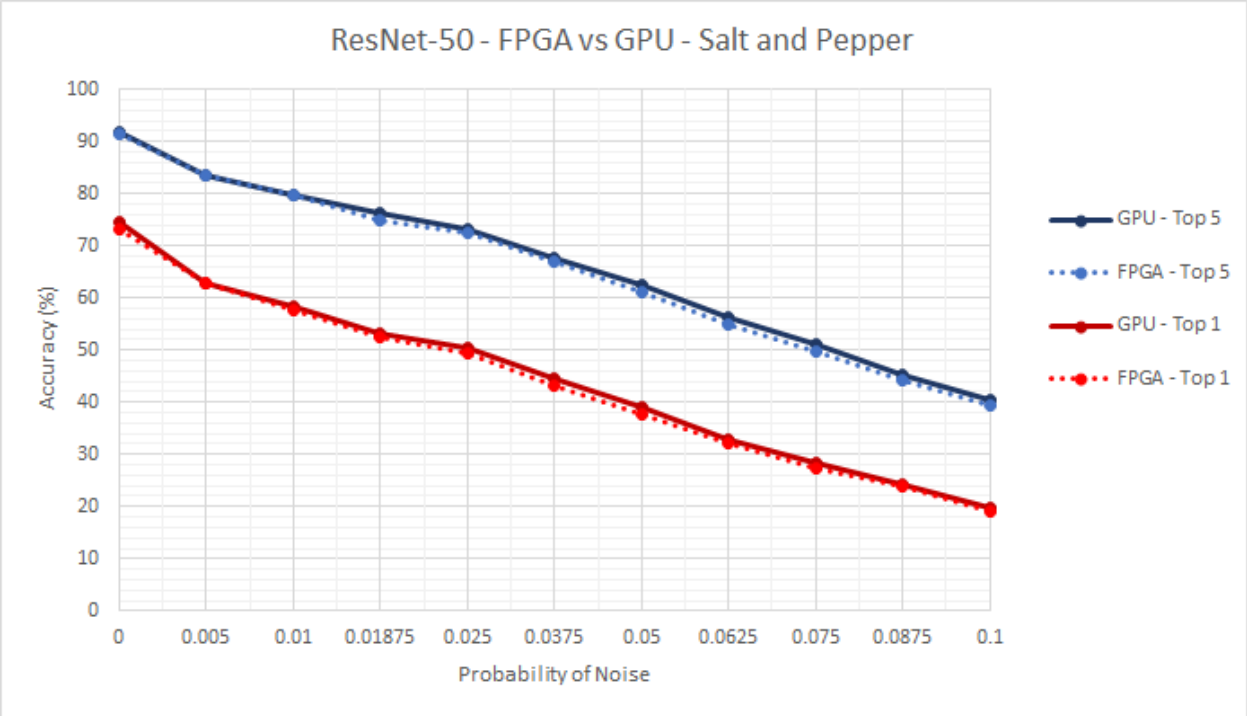
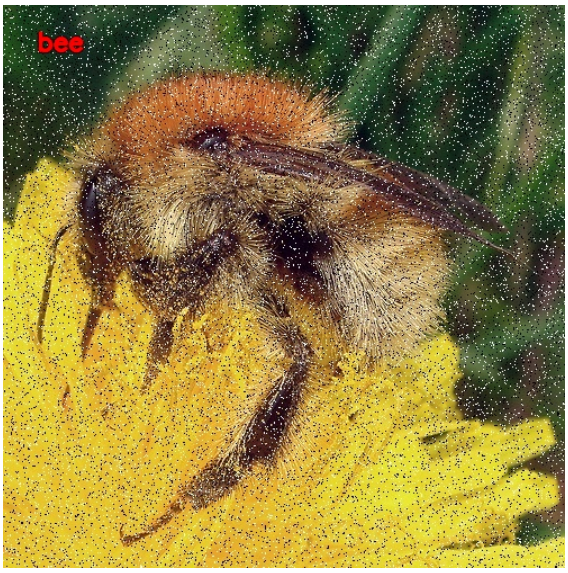


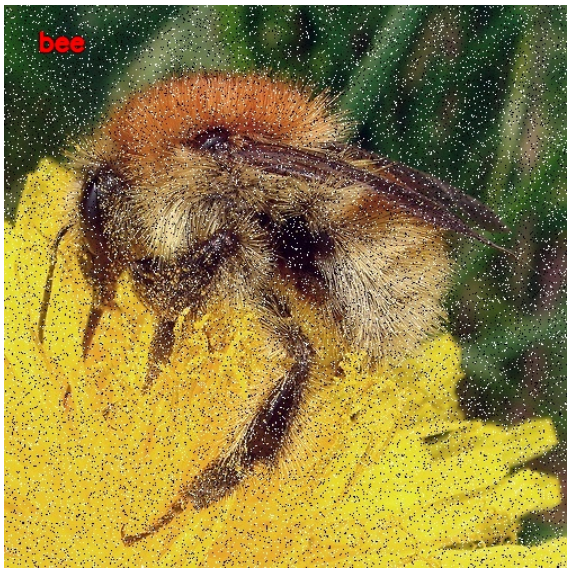
Figure 6.13: Side-by-side comparison of the results of salt-and-pepper degradation on both top-1 and top-5 accuracy for ResNet-50 on GPU versus FPGA implementation. *The set of blue lines (solid for GPU and dashed for FPGA) plot the top-5 accuracies when salt-and-pepper degradation is applied. The set of red lines (solid for GPU and dashed for FPGA) plot the top-1 accuracies when salt-and-pepper degradation is applied. With no salt-and-pepper degradation (a probability of noise parameter of 0) the top-1 and top-5 accuracies for both the FPGA and GPU models have an approximate 1% difference and are shown here for a baseline reference. The rate of accuracy decrease is almost identical between the two implementations with the FPGA accuracy maintaining an approximate 1% drop at all salt-and-pepper parameters.*



(a) *Original.*



(b) *GPU Inference.*



(c) *FPGA Inference.*

Figure 6.14: ResNet-50 example testing image with salt-and-pepper degradation at 0.05. *The original image from the ImageNet testing set appears in the top. Both GPU and FPGA models make the correct inference (bee). For this particular instance the degradation level was not severe enough to cause a mislabeling for either hardware model.*



(a) *Original.*



(b) *GPU Inference.*



(c) *FPGA Inference.*

Figure 6.15: ResNet-50 example testing image with salt-and-pepper degradation at 0.05. *The original image from the ImageNet testing set appears in the top. The GPU model provides correct inference (tent) but the FPGA model is incorrect (pinwheel). In this instance the higher precision of the GPU model weights appear to be more robust to this degree of salt-and-pepper degradation.*



(a) *Original.*



(b) *GPU Inference.*



(c) *FPGA Inference.*

Figure 6.16: ResNet-50 example testing image with salt-and-pepper degradation at 0.05. *The original image from the ImageNet testing set appears in the top. Both the GPU model and FPGA provides incorrect inference for the class snowplow. Both models provide the correct level before degradation, however, at salt-and-pepper degradation with a noise parameter of 0.05 neither model can successfully classify the image.*

6.1.3 Improvement of Accuracy with Gamma Correction Applied to Contrast Reduction Degradation on GPU and FPGA

Figure 6.17 explores the degree to which gamma correction can reduce the effects of the contrast reduction based degradation to the top-1 accuracy for our GPU implementation of ResNet-50. Each line in the graph represents the response of a particular contrast reduction factor across the range of gamma correction parameters of 1 to 2. For example, with a contrast reduction factor of 20.0% and a gamma correction factor of 1 (no correction) the top-1 accuracy is 61.48%. The optimal gamma factor for this contrast reduction parameter is observed at 1.5 returning the top-1 accuracy to 70.09% (an increase of 8.61%). The other corrections in Figure 6.17 show a similar response. We choose to display only contrast reduction parameters up to 3.3% since beyond this the degradation is so severe that barely any increase in accuracy can be had no matter the extent of correction of this sort. We observe that contrast reduced images and videos can have their quality significantly improved with optimal correction methods which also increases ResNet-50 inference accuracy. Given that images and videos with reduced contrast are common for in-the-wild imaging, this significant increase in accuracy is advantageous for Edge systems.

Figure 6.18 explores the degree to which gamma correction can reduce the effects of the contrast reduction based degradation to the top-1 accuracy for our FPGA implementation of ResNet-50. Each line in the graph represents the response of a particular contrast reduction factor across the range of gamma correction parameters of 1 to 2. For example, with a contrast reduction parameter of 20.0% and a gamma correction factor of 1 (no correction) the top-1 accuracy is 53.42%. The optimal gamma factor for this contrast reduction parameter is observed at 1.5 returning the top-1 accuracy to 66.25% (an increase of 12.83%). The other corrections in Figure 6.18 show a similar response. We choose to display only contrast reduction parameters up to 3.3% since beyond this the degradation is so severe that barely any increase in accuracy can be had no matter the extent of correction of this sort. Similar to the GPU model experiment, we observe that contrast reduced images and videos can have their quality significantly improved with optimal correction methods which also increases the FPGA ResNet-50 model inference accuracy. Given that images and videos with reduced contrast are common for in-the-wild imaging, this significant increase in accuracy is highly advantageous for Edge systems.

Figure 6.19 compares the top-1 accuracy of the GPU implementation of ResNet-50 versus the FPGA counterpart when mitigating the effects of the contrast reduction degradation with gamma correction. For reference, the GPU implementation has a top-1 accuracy of 74.44% when no degradation is present compared to that of 73.34% for the FPGA implementation. We chose to use three illustrative contrast reduction parameters to reflect the overall response of GPU versus FPGA correction (namely, 20.0%, 6.7% and 3.3%). For example, when the contrast reduction parameter is 20.0% and no correction is applied (gamma parameter is 1) the GPU top-1 accuracy (blue line) is 61.48% whereas the FPGA top-1 accuracy (orange line) is 53.42%. With an optimal gamma correction of 1.5 the GPU implementation at contrast reduction parameter of 20.0% raises to 70.08% (an 8.6% increase). Similarly, with an optimal gamma correction of 1.5 the FPGA implementation at a contrast reduction parameter of 20.0% raises to 66.25% (a 12.83%). For this example, after optimal correction the FPGA accuracy returns to within 3.83% of the GPU accuracy. This trend of a higher

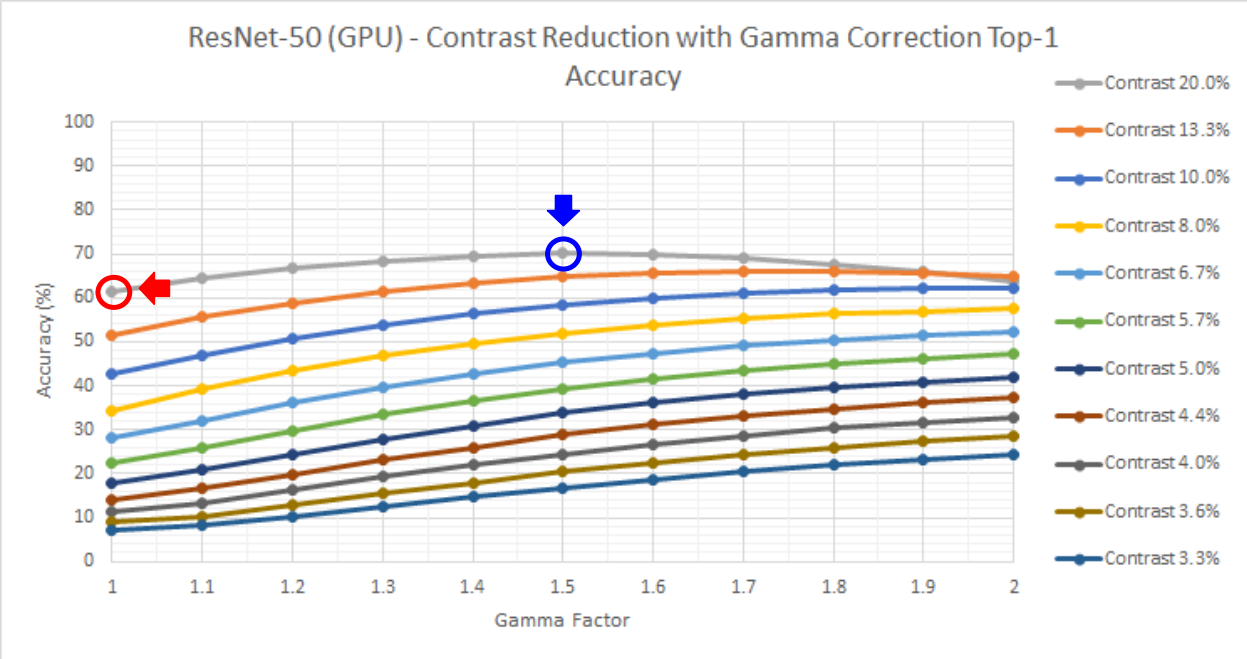


Figure 6.17: The results of gamma correction at various parameters when applied to contrast reduction based degradation for the GPU implementation of ResNet-50 for the ImageNet testing set (5,000 images). *Each line in the graph represents the response of a particular contrast reduction factor across the range of gamma correction parameters of 1 to 2. For example, with a contrast reduction factor of 20.0% (the top grey line) and a gamma correction factor of 1 (no correction) the top-1 accuracy is 61.48% (see red arrow). The optimal gamma factor for this contrast reduction parameter is observed at 1.5 returning the top-1 accuracy to 70.09% (see blue arrow) - an increase of 8.61%. Therefore, we observe that contrast reduced images and videos can have their quality significantly improved with optimal correction methods which also increases ResNet-50 inference accuracy. Given that images and videos with reduced contrast are common for in-the-wild imaging, this significant increase in accuracy is highly advantageous for Edge systems.*

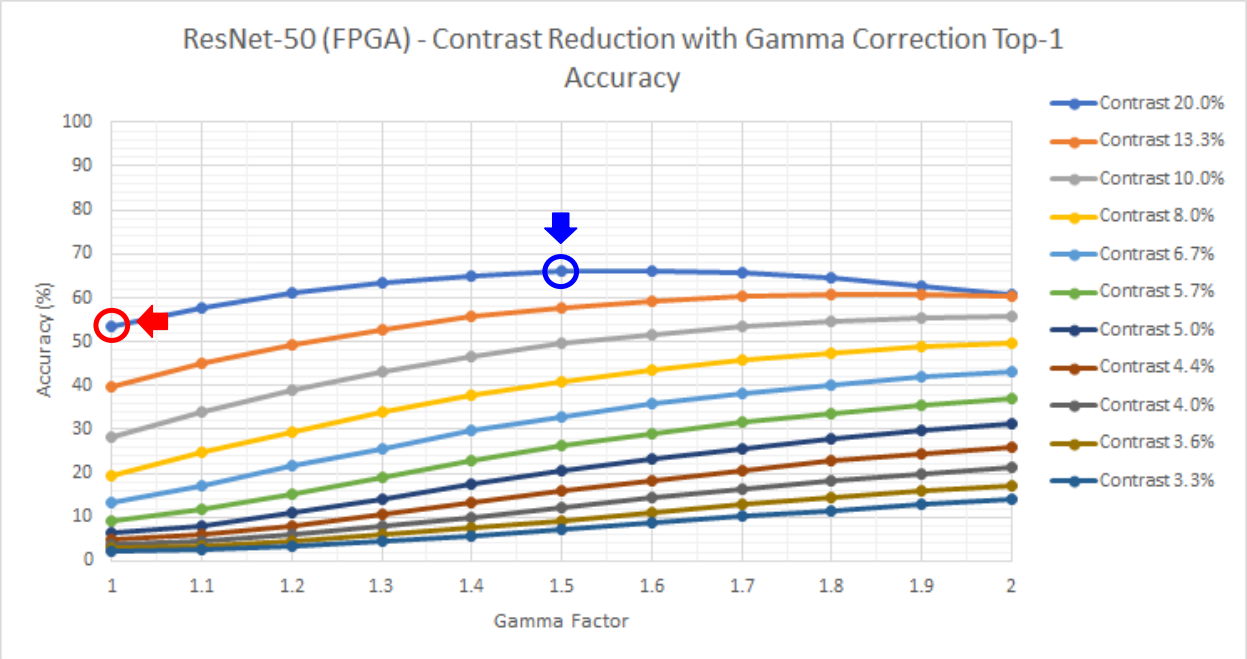


Figure 6.18: The results of gamma correction at various parameters when applied to contrast reduction based degradation for the FPGA implementation of ResNet-50. for the ImageNet testing set (5,000 images) *Each line in the graph represents the response of a particular contrast reduction factor across the range of gamma correction parameters of 1 to 2. For example, with a contrast reduction parameter of 20.0% (the top blue line) and a gamma correction factor of 1 (no correction) the top-1 accuracy is 53.42% (see red arrow). The optimal gamma factor for this contrast reduction parameter is observed at 1.5 returning the top-1 accuracy to 66.25% (see blue arrow) - an increase of 12.83%. As with the GPU model, we observe that contrast reduced images and videos can have their quality significantly improved with optimal correction methods which also increases the FPGA ResNet-50 model inference accuracy. Given that images and videos with reduced contrast are common for in-the-wild imaging, this significant increase in accuracy is highly advantageous for Edge systems.*

starting accuracy for the GPU but a higher accuracy correction for the FPGA appears to be consistent for the gamma correction of contrast reduction degradation for the ResNet-50 model. Therefore, the FPGA model can provide comparable robustness as the GPU model to contrast reduction noise while being delivered with far superior SWaP characters better suited for Edge applications.

Figure 6.20 displays an example of model inference as compared between the GPU and FPGA implementation for the case when an image from the ImageNet testing dataset is degraded by contrast reduction noise with a parameter of 20.0% and then corrected with an optimal gamma correction parameter of 1.5. The original image appears in the top left. The top right image displays the results after contrast reduction is applied with a parameter of 20.0%. The bottom two images display optimal gamma correction of 1.5 applied to the contrast reduced image and the classification labels after correction is applied (GPU result on bottom left and FPGA result on bottom right). The images in this figure represent a single sample test image that failed for both GPU and FPGA when degraded by contrast reduction parameter 20.0% but are correctly labeled after optimal correction.

6.1.4 Improvement of Accuracy with Median Filter Correction Applied to Salt-and-Pepper Degradation on GPU and FPGA

Figure 6.21 explores the degree to which median filter correction can reduce the effects of the salt-and-pepper based degradation to the top-1 accuracy for our GPU implementation of ResNet-50. Each line in the graph represents the response of a particular salt-and-pepper probability parameter corrected with various median filter kernel sizes. The kernel size of "None" indicates no correction and is used to illustrate the large increase in top-1 accuracy that the median filter provides. For a reference, the Prob 0.0 line in Figure 6.21 at a "None" kernel size provides a top-1 accuracy of 74.44%, this is reflective of no degradation and no corresponding correction. As an example, the line (bottom blue line) with the most severe salt-and-pepper degradation occurs with the probability parameter of 0.1. With no correction it has a top-1 accuracy of 19.9%. However, with an optimal correction kernel size of 5x5 the top-1 accuracy is increased to 63.74%, which represents a significant increase of 43.84%. The other corrections show a similar response of significant improvement from salt-and-pepper degradation when optimal median filter correction is applied. We observe that salt-and-pepper degraded images and videos can have their quality significantly improved when optimal correction is applied which in turn increases ResNet-50 inference accuracy. Given that images and videos with salt-and-pepper noise are common for in-the-wild imaging, this significant increase in accuracy is highly advantageous for Edge systems.

Figure 6.22 explores the degree to which median filter correction can reduce the effects of the salt-and-pepper based degradation to the top-1 accuracy for our FPGA implementation of ResNet-50. Each line in the graph represents the response of a particular salt-and-pepper probability parameter corrected with various median filter kernel sizes. The kernel size of "None" indicates no correction and is used to illustrate the large increase in top-1 accuracy that the median filter provides. For a reference, the Prob 0.0 line in Figure 6.22 at a "None" kernel size provides a top-1 accuracy of 73.34% (approximately 1% lower than the GPU implementation), this is reflective of no degradation and no corresponding correction. As

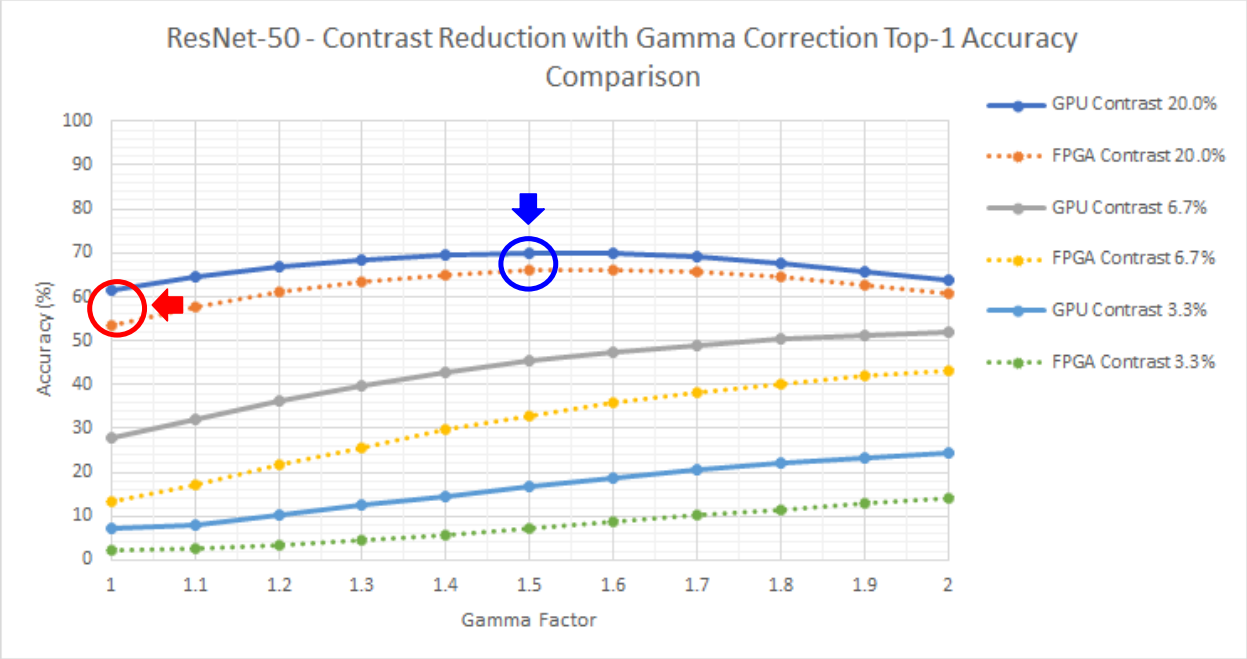


Figure 6.19: Side-by-side comparison of the results of gamma correction on contrast reduction based degradation for ResNet-50 on GPU versus FPGA implementations. *Three illustrative contrast reduction parameters were selected to compare implementation performance: 20.0%, 6.7%, and 3.3% (solid lines for GPU and dashed lines for FPGA). For example, when the contrast reduction parameter is 20.0% and no correction is applied (gamma parameter is 1) the GPU top-1 accuracy (blue line) is 61.48% whereas the FPGA top-1 accuracy (orange line) is 53.42% (see red arrow). With an optimal gamma correction of 1.5 the GPU implementation at contrast reduction parameter of 20.0% raises to 70.08% (see blue arrow) - an 8.6% increase. Similarly, with an optimal gamma correction of 1.5 the FPGA implementation at a contrast reduction parameter of 20.0% raises to 66.25% (see blue arrow) - a 12.83% increase. For this example, after optimal correction the FPGA accuracy returns to within 3.83% of the GPU accuracy. Therefore, even though the contrast reduction degradation impacts the FPGA model more severely, the gamma correction method improves the FPGA model at a higher rate than the GPU model. After optimal correction, the FPGA model shows similar robustness to contrast reduction as compared to the GPU model. This similar robustness coupled with the FPGA's advantage of far superior SWaP characteristics make it the superior implementation for deployment in Edge applications.*



(a) *Original.*



(b) *Contrast at 20.0%.*



(c) *GPU Inference - Gamma Corrected.*



(d) *FPGA Inference - Gamma corrected.*

Figure 6.20: ResNet-50 example testing image with contrast reduction degradation (20.0%) and gamma correction (1.5). *The original image from the ImageNet testing set appears in the top left. The top right image displays this image after contrast reduction is applied with a parameter of 20.0%. Both the GPU and FPGA failed to properly classify this image at contrast reduction degradation at 20.0% but both provide the correct inference class label (lighthouse) when an optimal gamma correction of 1.5 is applied (GPU result in bottom left and FPGA result in bottom right).*

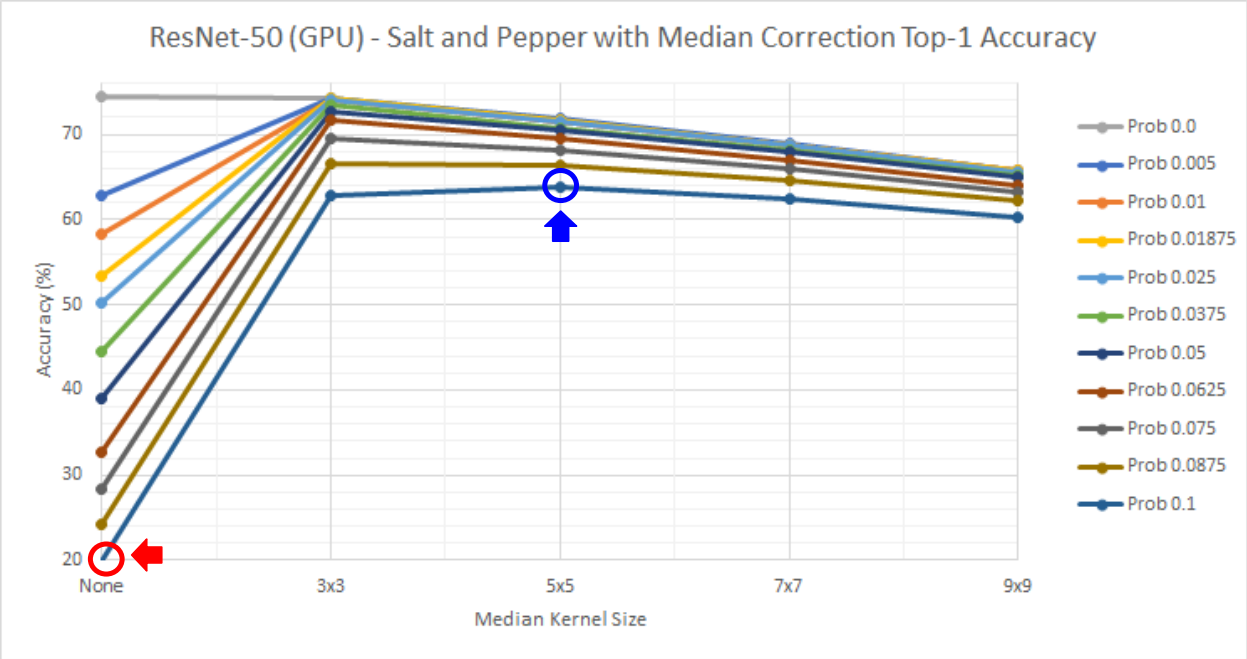


Figure 6.21: The results of median filter correction at various kernel sizes when applied to salt-and-pepper based degradation for the GPU implementation of ResNet-50 for the ImageNet testing set (5,000 images). *Each line in the graph represents the response of a particular salt-and-pepper degradation probability of noise parameter when corrected by several median filters of various kernel sizes. The kernel size of "None" is used to indicate no correction is applied. The top line reflects the case where no salt-and-pepper degradation is introduced (probability of noise parameter of 0.0) and is used to illustrate that the median filtering correction can have a negative impact upon image quality by causing a blurring effect if the kernel size is set too large. For example, the bottom blue line represents the most severe salt-and-pepper degradation with the probability parameter of 0.1. With no correction it has a top-1 accuracy of 19.9% (see red arrow). However, with an optimal correction kernel size of 5x5 the top-1 accuracy is increased to 63.74% (see blue arrow), which represents a significant increase of 43.84%. Of additional note is that simply applying a median filter kernel of size 3x3 can greatly increase accuracy regardless of the salt-and-pepper degradation parameter. Therefore, we observe that salt-and-pepper degraded images and videos can have their quality significantly improved when optimal correction is applied which in turn increases ResNet-50 inference accuracy. Given that images and videos with salt-and-pepper noise are common for in-the-wild imaging, this significant increase in accuracy is highly advantageous for Edge systems.*

an example, the line (bottom blue line) with the most severe salt-and-pepper degradation occurs with the probability parameter of 0.1. With no correction it has a top-1 accuracy of 19.14%. However, with an optimal correction kernel size of 5x5 the top-1 accuracy is increased to 63.29%, which represents a significant increase of 44.15%. The other corrections show a similar response of significant improvement from salt-and-pepper degradation when optimal median filter correction is applied. Similar to the GPU model experiment, we observe that salt-and-pepper degraded images and videos can have their quality significantly improved when optimal correction is applied which also increases the FPGA ResNet-50 inference accuracy. Given that images and videos with salt-and-pepper noise are common for in-the-wild imaging, this significant increase in accuracy is highly advantageous for Edge systems.

Figure 6.23 compares the top-1 accuracy of the GPU implementation of ResNet-50 versus the FPGA counterpart when mitigating the effects of the salt-and-pepper degradation with median filter correction. For reference, the GPU implementation has a top-1 accuracy of 74.44% when no degradation is present compared to that of 73.34% for the FPGA implementation. We chose to use three illustrative salt-and-pepper probability of noise parameters to reflect the overall response of GPU versus FPGA correction (namely, 0.005, 0.05 and 0.1). For example, when the probability of noise parameter is 0.05 and no correction is applied (no median filter applied, i.e. "None" in Figure 6.23) the GPU top-1 accuracy is 39.06% whereas the FPGA top-1 accuracy of 37.8%. With an optimal median filter size of 3x3 the GPU implementation at salt-and-pepper probability of 0.05 raises to 72.74% (a 33.68% increase). Similarly, with an optimal median filter size of 3x3 the FPGA implementation at salt-and-pepper probability of 0.05 raises to 71.86% (a 34.06% increase). Unlike with the contrast reduction based degradation, for the salt-and-pepper noise both the GPU and FPGA implementations respond with similar drops in accuracy for degradation and similar improvements in accuracy for correction. Therefore, the FPGA model can provide identical robustness as the GPU model to salt-and-pepper degradation while being delivered with far superior SWaP characters better suited for Edge applications.

Figure 6.24 displays an example of model inference as compared between the GPU and FPGA implementation for the case when an image from the ImageNet testing dataset is degraded by salt-and-pepper noise with a parameter of 0.05 and then corrected with an optimal median kernel filter of size 3x3. The original image appears in the top left. The top right image displays the results after salt-and-pepper degradation is applied with a parameter of 0.05. The bottom two images display optimal median filter correction with a kernel of size 3x3 applied to the salt-and-pepper degraded image and the classification labels after correction is applied (GPU result on bottom left and FPGA result on bottom right). The images in this figure represent a single sample test image that failed for both GPU and FPGA when degraded by salt-and-pepper parameter 0.05 but are correctly labeled after correction.

6.1.5 Experimental Results of ResNet-50 Benchmarking

Table 6.1 provides a series of metrics summarizing the contrast reduction degradation and gamma correction evaluation of the ResNet-50 model in our experiments. This table outlines seven experiments to provide an overview of both the GPU and FPGA pipeline performances. For each of these experiments we also display the associated average SSIM score for the

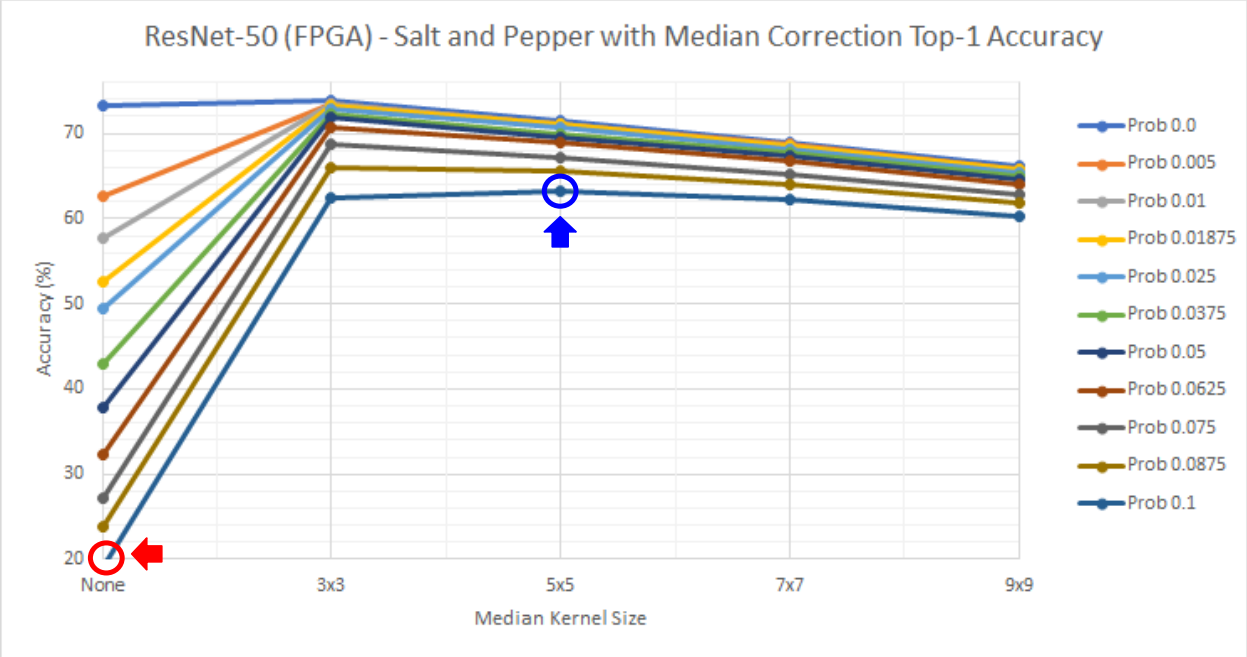


Figure 6.22: The results of median filter correction at various kernel sizes when applied to salt-and-pepper based degradation for the FPGA implementation of ResNet-50 for the ImageNet testing set (5,000 images). *Each line in the graph represents the response of a particular salt-and-pepper degradation probability of noise parameter when corrected by several median filters of various kernel sizes. The kernel size of "None" is used to indicate no correction is applied. The top line reflects the case where no salt-and-pepper degradation is introduced (probability of noise parameter of 0.0) and is used to illustrate that the median filtering correction can have a negative impact upon image quality by causing a blurring effect if the kernel size is set too large. For example, the bottom blue line represents the most severe salt-and-pepper degradation with the probability parameter of 0.1. With no correction it has a top-1 accuracy of 19.14% (see red arrow). However, with an optimal correction kernel size of 5x5 the top-1 accuracy is increased to 63.29% (see blue arrow), which represents a significant increase of 44.15%. Of additional note is that simply applying a median filter kernel of size 3x3 can greatly increase accuracy regardless of the salt-and-pepper degradation parameter. As with the GPU model, we observe that salt-and-pepper degraded images and videos can have their quality significantly improved when optimal correction is applied which in turn increases the FPGA ResNet-50 inference accuracy. Given that images and videos with salt-and-pepper noise are common for in-the-wild imaging, this significant increase in accuracy is highly advantageous for Edge systems.*

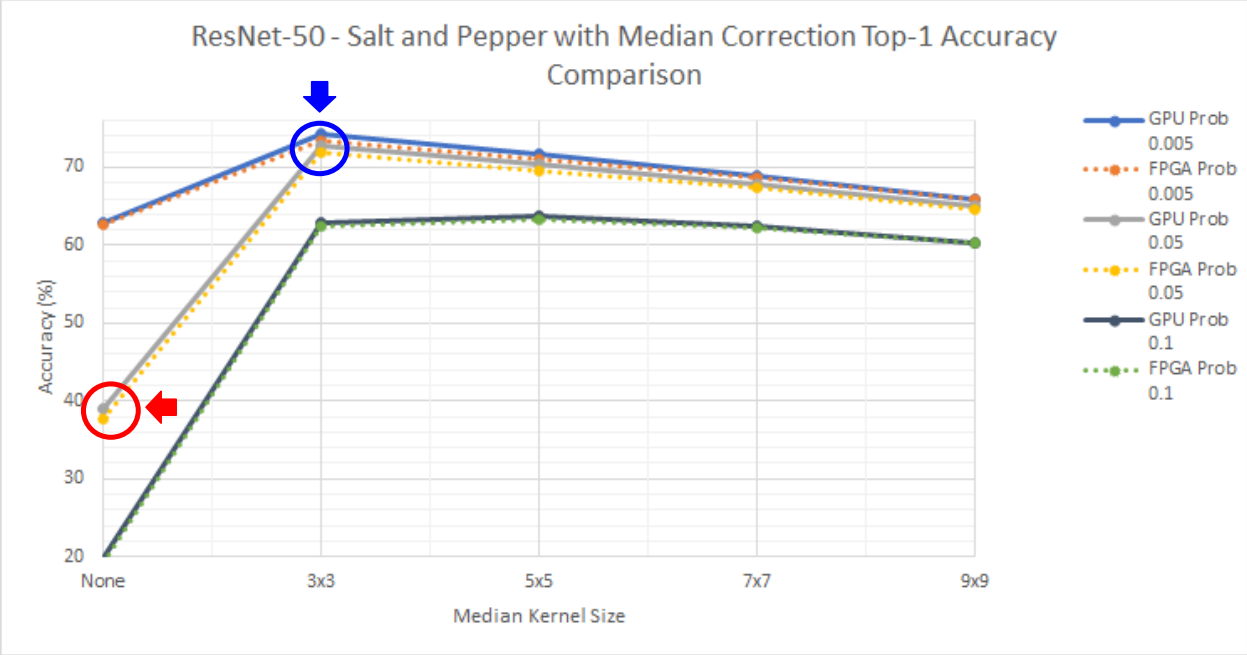


Figure 6.23: Side-by-side comparison of the results of median filter correction on salt-and-pepper based degradation for ResNet-50 on GPU versus FPGA implementations. *Three illustrative salt-and-pepper probability of noise parameters were selected to compare implementation performance: 0.005, 0.05, and 0.1 (solid lines for GPU and dashed lines for FPGA). For example, when the probability of noise parameter is 0.05 and no correction is applied (no median filter applied, i.e. "None") the GPU top-1 accuracy is 39.06% whereas the FPGA top-1 accuracy of 37.8% (see red arrow). With an optimal median filter size of 3x3 the GPU implementation at salt-and-pepper probability of 0.05 raises to 72.74% (see blue arrow) - a 33.68% increase. Similarly, with an optimal median filter size of 3x3 the FPGA implementation at salt-and-pepper probability of 0.05 raises to 71.86% (see blue arrow) - a 34.06% increase. For this example, after optimal correction the FPGA returns to within 0.88% of the GPU accuracy. Unlike with the gamma corrected contrast reduced experiments, the FPGA has near identical robustness as the GPU to salt-and-pepper degradation when optimal median filter correction is applied. This near identical robustness coupled with the FPGA's advantage of far superior SWaP characteristics make it the superior implementation for deployment in Edge applications.*



(a) *Original.*



(b) *Salt-and-Pepper at 0.05.*



(c) *GPU Inference - Median Filter Corrected.*



(d) *FPGA Inference - Median Filter corrected.*

Figure 6.24: ResNet-50 example testing image with salt-and-pepper degradation (0.05) and median filter correction (3x3). *The original image from the ImageNet testing set appears in the top left. The top right image displays this image after salt-and-pepper degradation is applied with a parameter of 0.05. Both the GPU and FPGA failed at salt-and-pepper degradation at 0.05 but both provide the correct inference class label (snowplow) when an optimal median filter correction of 3x3 is applied (GPU result in bottom left and FPGA result in bottom right.*

quality of the degraded or corrected images being evaluated in the ImageNet dataset. As theorized in Chapter 5, the lower the SSIM score the lower the inference performance.

As demonstrated in Chapter 4, the GPU and FPGA implementations have near identical performance at the base experiment where no contrast reduction degradation is introduced (see table 6.1). However, for the experiments with contrast reduction degradation the FPGA model is consistently impacted more severely than the GPU equivalent implementation. We theorize this is due to the nature of the weights in the two model types. The GPU model using 32-bit floating point weights provides greater precision and better performance to the contrast reduction based degradation as opposed to the less precise 8-bit quantized weights in the FPGA model. However, as summarized in Chapter 4, this increase in precision comes at the cost of much greater SWaP requirements for deployment.

We observe that correction methods can offer significant improvements in inference for both GPU and FPGA (see table 6.1). Of particular note is the impact of degradation and correction to the FPGA-based systems. Our experiments seem to indicate the the FPGA-based model is more impacted by degradation and simultaneously realizes greater improvement by correction. Once again, our intuition is that this is due to the quantized weights leveraged by the FPGA-based model versus the more robust floating point weights in the GPU model equivalent. This would lead us to believe that an FPGA Edge system with the potential for low light imaging would benefit greatly from such a correction functionality.

Table 6.2 provides a series of metrics summarizing the salt-and-pepper and median filter correction evaluation of the ResNet-50 model in our experiments. This table outlines seven experiments to provide an overview of both the GPU and FPGA pipeline performances. For each of these experiments we also display the associated average SSIM score for the quality of the degraded or corrected images being evaluated in the ImageNet dataset. As theorized in Chapter 5, the lower the SSIM score the lower the inference performance.

Once again we note that at the base experiment (with no degradation introduced) the performance of the GPU and FPGA models is near identical (approximately 1% - see table 6.2). However, unlike with the contrast reduction based degradation, both the GPU and FPGA models respond in a similar fashion by keeping this nearly identical inference performance at each of the degradation experiments. Therefore, the less precise quantized FPGA model appears to have an equal robustness to the salt-and-pepper based noise as compared to the GPU model.

We observe that correction methods can offer significant improvements in inference on the salt-and-pepper degradation for both GPU and FPGA (see table 6.2). During median filter correction the GPU and FPGA models receive very similar improvement showing that both are equally impacted by correction measures. Our intuition is that this robustness to salt-and-pepper noise is inherent to the architecture of deep learning models themselves since they leverage averaging and pooling at multiple layers.

6.2 Pipeline 2: YOLO Version 3 Evaluation

The YOLO version 3 pipeline is evaluated and compared on both our GPU and FPGA implementations. As with the base model evaluation in Chapter 4, we use the VOC2007 testing set for the YOLO version 3 experiments. As with the ResNet-50 experiments in the

Table 6.1: Comparison of contrast reduction degradation and gamma correction on ImageNet testing dataset of GPU (32-bit floating point model) vs. FPGA (8-bit integer quantized model) performance on ResNet-50. *This table outlines seven experiments to provide an overview of both the GPU and FPGA pipeline performances. For each of these experiments we also display the associated average SSIM score for the quality of the degraded or corrected images being evaluated in the ImageNet dataset. As demonstrated in Chapter 4, the GPU and FPGA implementations have near identical performance at the base experiment where no contrast reduction degradation is introduced (Base experiment in the table). For example, with contrast reduction degradation of 20% applied the GPU accuracy drops from 74.44% to 61.48% (a 12.96% decrease) and the associated SSIM scores drops from 1 to 0.2563 (a decrease of 0.7437). For the same contrast degradation with the FPGA, accuracy drops from 73.34% to 53.42% (a 19.92% decrease) with the same SSIM score to reflect testing dataset quality. However, after optimal gamma correction is applied the GPU accuracy returns to 70.08% (an 8.6% increase) and the FPGA returns to 66.25% (a 12.83% increase) and the SSIM score for the corrected dataset becomes 0.8901 (an increase of 0.6338). As theorized, the SSIM score can be used as a predictor for inference performance. These experiments demonstrate that the FPGA model can achieve comparable accuracy to the GPU model when optimal gamma correction is applied to contrast reduction degradation. Although the FPGA model provides slightly lower accuracy for this type of degradation it comes with the advantage of superior SWaP characteristics better suited for Edge applications.*

Experiment	**SSIM Score**	**GPU Top-1 Accuracy**	**GPU Improvement**	**FPGA Top-1 Accuracy**	**FPGA Improvement**
Base	1	74.44%	N/A	73.34%	N/A
Contrast 20.0%	0.2563	61.48%	N/A	53.42%	N/A
Contrast 20.0% optimal gamma	0.8901	70.08%	8.6%	66.25%	12.83%
Contrast 6.7%	0.0753	28.04%	N/A	13.30%	N/A
Contrast 6.7% optimal gamma	0.6527	52.09%	24.05%	43.09%	29.79%
Contrast 3.3%	0.0399	7.16%	N/A	2.28%	N/A
Contrast 3.3% optimal gamma	0.1691	24.49%	17.33%	14.06%	11.78%

Table 6.2: Comparison of salt-and-pepper degradation and median filter correction on ImageNet testing dataset of GPU (32-bit floating point model) vs. FPGA (8-bit integer quantized model) performance on ResNet-50. *This table outlines seven experiments to provide an overview of both the GPU and FPGA pipeline performances. For each of these experiments we also display the associated average SSIM score for the quality of the degraded or corrected images being evaluated in the ImageNet dataset. As demonstrated in Chapter 4, the GPU and FPGA implementations have near identical performance at the base experiment where no salt-and-pepper degradation is introduced (Base experiment in the table). For example, with salt-and-pepper degradation of 0.05 the GPU accuracy drops from 74.44% to 39.06% (a 35.38% decrease) and the associated SSIM scores drops from 1 to 0.2545 (a decrease of 0.7455). For the same salt-and-pepper degradation with FPGA, accuracy drops from 73.34% to 37.8% (a 35.53% decrease) with the same SSIM score to reflect testing dataset quality. However, after optimal median filter correction is applied the GPU accuracy returns to 72.74% (a 33.68% increase) and the FPGA returns to 71.86% (a 34.06% increase) and the SSIM score for the corrected dataset becomes 0.8381 (an increase of 0.5836). As theorized, the SSIM score can be used as a predictor for inference performance. These experiments demonstrate that the FPGA model can achieve near identical accuracy to the GPU model when optimal median filter correction is applied to salt-and-pepper degradation. Not only does the FPGA model provides near identical robustness for this type of degradation it comes with the advantage of superior SWaP characteristics better suited for Edge applications.*

Experiment	SSIM Score	GPU Top-1 Accuracy	GPU Improvement	FPGA Top-1 Accuracy	FPGA Improvement
Base	1	74.44%	N/A	73.34%	N/A
Prob. 0.005	0.8109	62.84%	N/A	62.7%	N/A
Prob. 0.005 optimal median	0.8519	74.28%	11.44%	73.44%	10.74%
Prob. 0.05	0.2545	39.06%	N/A	37.8%	N/A
Prob. 0.05 optimal median	0.8381	72.74%	33.68%	71.86%	34.06%
Prob. 0.1	0.1431	19.9%	N/A	19.14%	N/A
Prob 0.1 optimal median	0.7942	63.74%	43.84%	63.29%	44.15%

previous section, we first explore the impact of two degradations: contrast reduction and salt-and-pepper noise upon both GPU and FPGA models. We analyze how these noises impact the separate hardware implementations both individually as well as a side-by-side comparison. As in Chapter 4, we use both mean average precision for complete model performance (all classes) and average precision for individual class performance for our metrics for this object detection evaluation. Next, we evaluate how the associated correction technique for each degradation impacts performance. As before, we explore the GPU and FPGA implementations separately and then perform a comparison.

6.2.1 Comparison of Contrast Reduction Degradation on GPU and FPGA

The results of the contrast reduction degradation upon the GPU implementation of YOLO version 3 as assessed on the VOC2007 testing set can be seen in Figure 6.25. For reference, the bold green line reflects the performance of the mean average precision (mAP) for all classes, the bold blue line reflects the performance of the best performing class (car) and the bold red line the the performance of the worst performing class (potting plant). With no contrast reduction (a contrast parameter of 100.0%) the mAP for all classes is observed to be 78.46%. For no contrast reduction the best performing class (car) has an average precision of 86.45% and the worst performing class (potted plant) has an average precision of 51.68%. As expected, the greater the contrast reduction parameter the lower the calculated average precision. Additionally, one can observe that the degradation decreases for all classes at a similar rate as the contrast reduction parameter increases. After a contrast reduction of 2.5% the decrease in mAP plateaus and is not reported here. Images and video frames degraded beyond 2.5% are shown to be visually improvable with optimal correction but not to a degree such that desirable CNN mAP can be achieved.

The results of the contrast reduction degradation upon the FPGA implementation of YOLO version 3 can be seen in Figure 6.26. For reference, the bold green line reflects the performance of the mean average precision (mAP) for all classes, the bold blue line reflects the performance of the best performing class (car) and the bold red line the the performance of the worst performing class (potted plant). With no contrast reduction (a contrast parameter of 100.0%) the mAP for all classes is observed to be 77.44%. For no contrast reduction the best performing class (car) has an average precision of 85.47% and the worst performing class (potted plant) has an average precision of 50.46%. As expected, the greater the contrast reduction parameter the lower the calculated average precision. One can observe that the degradation decreases for all classes at a similar rate as the contrast reduction parameter increases. After a contrast reduction of 2.5% the decrease in mAP plateaus and is not reported here. Images and video frames degraded beyond 2.5% are shown to be visually improvable with optimal correction but not to a degree such that desirable CNN mAP can be achieved. Additionally, as with the GPU versus FPGA performance for ResNet-50, the contrast reduction degradation appears to have a stronger impact upon the quantized model used in the FPGA implementation.

Figure 6.27 compares the average precision of the GPU implementation of YOLO version 3 versus the FPGA counterpart when degrading via the contrast reduction noise. The

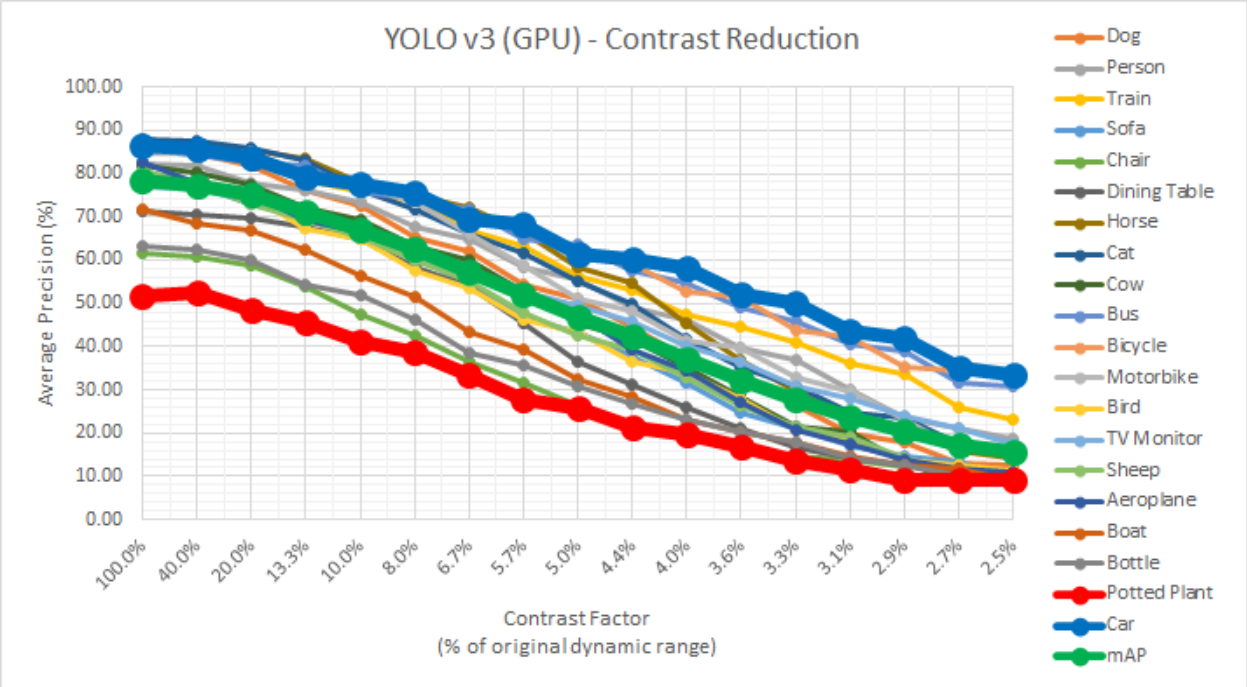


Figure 6.25: Results of contrast reduction degradation on YOLO version 3 GPU implementation average precision as evaluated on the VOC2007 testing set (4,952 images). *With no contrast reduction (a contrast parameter of 100.0%) the mAP for all classes (bold green line) is observed to be 78.46%. For no contrast reduction the best performing class ("car" - bold blue line) has an average precision of 86.45% and the worst performing class ("potted plant" - bold red line) has an average precision of 51.68%. After a contrast reduction of 2.5% the decrease in mAP plateaus and is not reported here. Images and video frames degraded beyond 2.5% are shown to be visually improvable with optimal correction but not to a degree such that desirable CNN mAP can be achieved.*

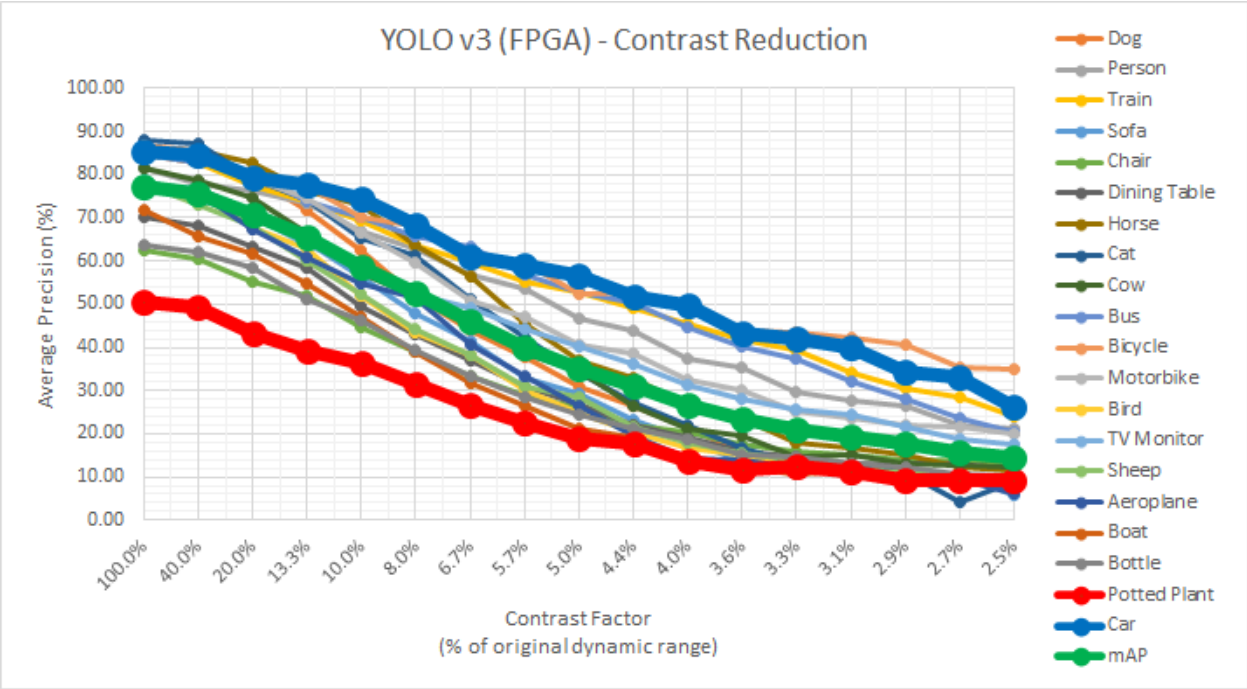


Figure 6.26: Results of contrast reduction degradation on YOLO version 3 FPGA implementation average precision as evaluated on the VOC2007 testing set (4,952 images). *With no contrast reduction (a contrast parameter of 100.0%) the mAP for all classes (bold green line) is observed to be 77.44%. For no contrast reduction the best performing class ("car" - bold blue line) has an average precision of 85.47% and the worst performing class ("potted plant" - bold red line) has an average precision of 50.46%. After a contrast reduction of 2.5% the decrease in mAP plateaus and is not reported here. Images and video frames degraded beyond 2.5% are shown to be visually improvable with optimal correction but not to a degree such that desirable CNN mAP can be achieved.*

GPU implementation has a mAP of 78.46% as compared to the FPGA implementation of 77.44% (a difference of 1.02%) when no contrast reduction degradation is present (contrast parameter of 100.0%). Although there is only an approximate one percent point difference with unaltered images, we observe that the FPGA mAP falls off at a quicker rate than the GPU implementation. The most likely reason for this difference in degradation impact is the nature of the two model weights. Although the 32-bit floating point weights in the GPU model require higher SWaP characteristics to calculate they are more robust to the contrast reduction degradation than the 8-bit integer quantized weights of the FPGA model. However, as detailed later, the FPGA model's robustness to contrast degradation can be greatly improved with optimal correction measures.

Finally, in this section comparing the results of contrast reduction degradation between the GPU and FPGA models, we provide visual examples from the VOC2007 testing set to illustrate our statistical findings. We chose a contrast reduction parameter of 20.0% to accommodate easy human readability in these examples images. Figure 6.28 provides an example testing image where both the GPU and FPGA models made the proper prediction (two instances of class potted plant) even with contrast reduction of 20%. Figure 6.29 has the GPU predicting the correct class (single instance of a potted plant) but the FPGA model predicting incorrectly (zero detections of any class). Figure 6.30 shows an instance where both the GPU and FPGA provided incorrect inferences to the testing image ground truth of two instance of class horse. To the best of our knowledge, there were no instances where the FPGA model provided a correct inference for a degraded image that the GPU model predicted incorrectly. This finding further provides evidence that differences between the inference accuracies of the two models arise from the lower precision weights leveraged by the FPGA model. However, as detailed in Chapter 4 and reviewed at the end of this section, the lower precision weights result in a slightly less accurate model but provide the hardware platform for significant gains in SWaP parameters to maximize Edge performance.

6.2.2 Comparison of Salt-and-Pepper Degradation on GPU and FPGA

The impact of the salt-and-pepper degradation upon the GPU implementation of YOLO version 3 as assessed on the VOC2007 testing set can be seen in Figure 6.31. For reference, the bold green line reflects the performance of the mean average precision (mAP) for all classes, the bold blue line reflects the performance of the best performing class (bicycle) and the bold red line the the performance of the worst performing class (potted plant). With no noise (a parameter of 0) the mAP for all classes is observed to be 78.46%. For no salt-and-pepper noise the best performing class (bicycle) has an average precision of 86.47% and the worst performing class (potted plant) has an average precision of 51.68%. As expected, the greater the salt-and-pepper parameter the lower the calculated average precision. As with the ResNet-50 architecture, the YOLO version 3 architecture appears to have little inherent robustness to salt-and-pepper degradation. For our experiments we cut off the probability of noise parameter to 0.1.

The impact of the salt-and-pepper degradation upon the FPGA implementation of YOLO version 3 as assessed on the VOC2007 testing set can be seen in Figure 6.32. For reference,

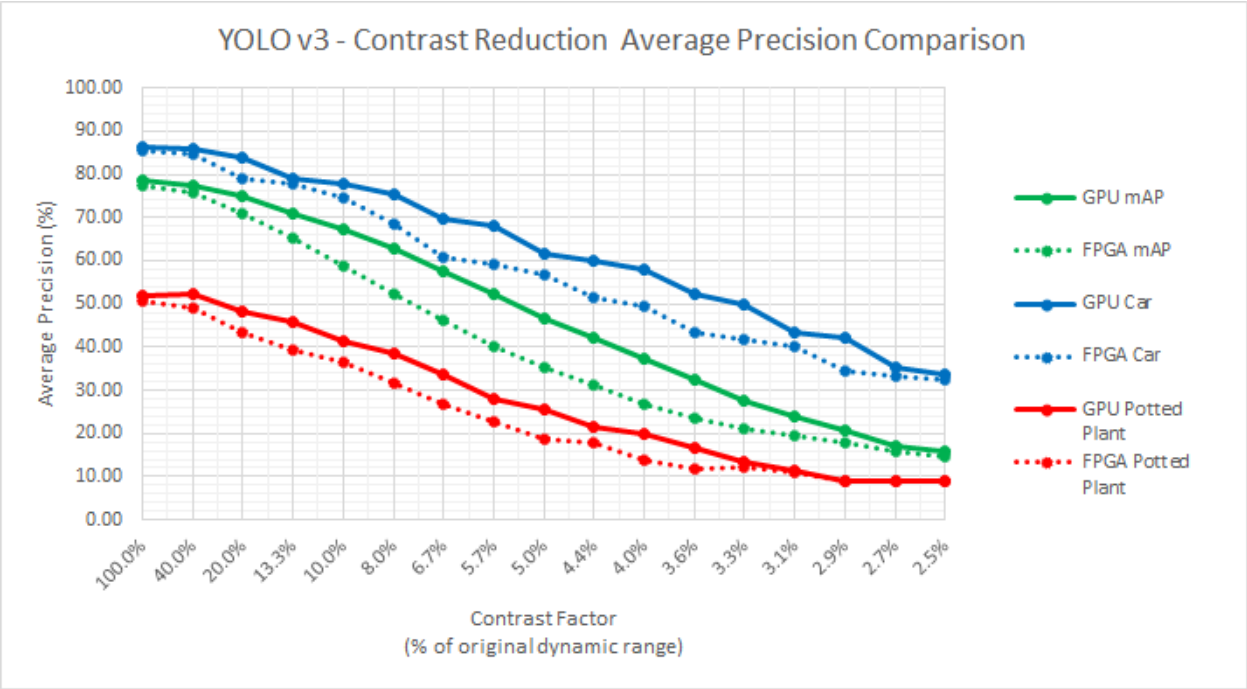
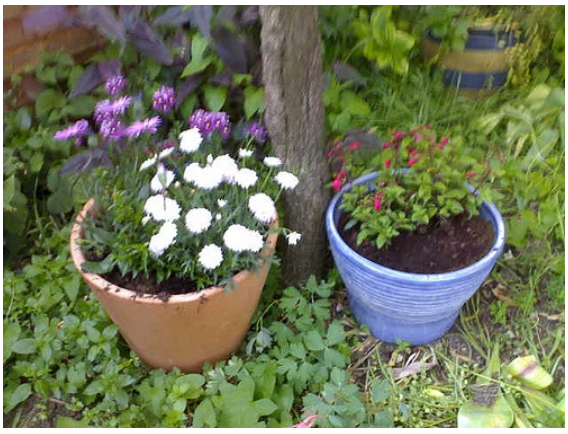
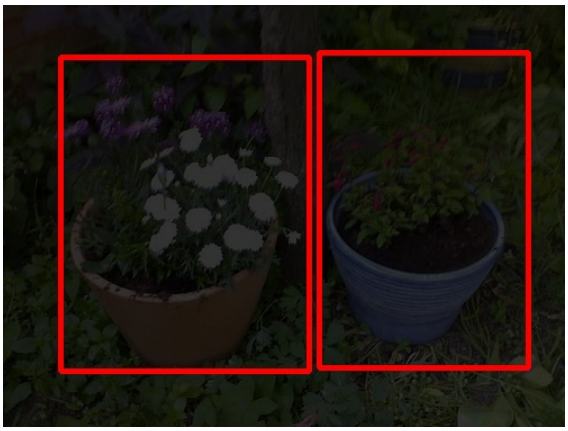


Figure 6.27: Comparison of the results of contrast reduction degradation on average precision for YOLO version 3 on GPU versus FPGA implementation for the VOC2007 testing set (4,952 images). *The GPU implementation has a mAP of 78.46% as compared to the FPGA implementation of 77.44% (a difference of 1.02%) when no contrast reduction degradation is present (contrast parameter of 100.0%). The FPGA implementation average precision decreases at a quicker rate than the GPU implementation indicating a greater impact of degradation upon model inference. The most likely reason for this difference in degradation impact is the nature of the two model weights. Although the 32-bit floating point weights in the GPU model require higher SWaP characteristics to calculate they are more robust to the contrast reduction degradation than the 8-bit integer quantized weights of the FPGA model.*



(a) *Original.*

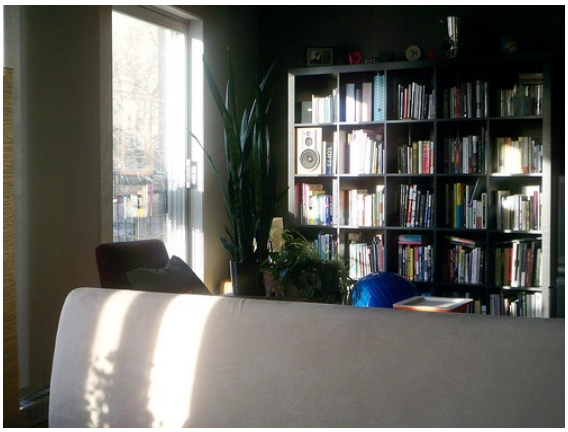


(b) *GPU Inference.*



(c) *FPGA Inference.*

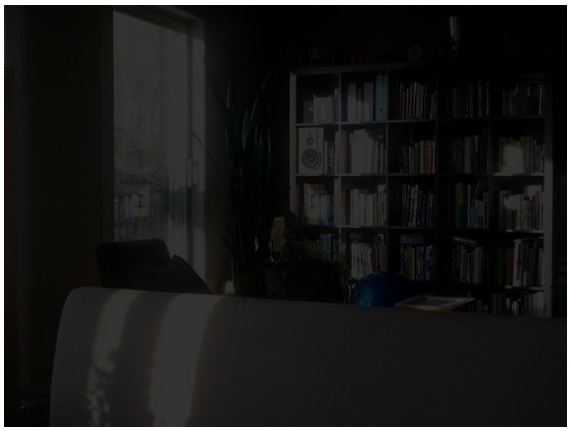
Figure 6.28: YOLO version 3 example testing image with contrast reduction degradation at 20.0%. *The original image from the VOC2007 testing set appears in the top. Both GPU and FPGA models make the correct inference (two instances of class potted plant). For this particular instance the degradation level was not severe enough to cause a mislabeling for either hardware model.*



(a) *Original.*



(b) *GPU Inference.*



(c) *FPGA Inference.*

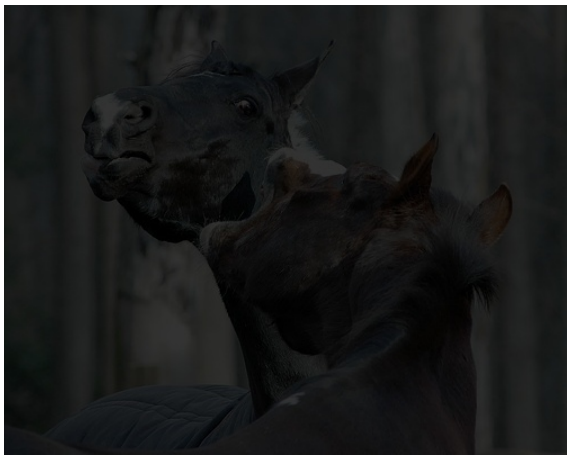
Figure 6.29: YOLO version 3 example testing image with contrast reduction degradation at 20.0%. *The original image from the VOC2007 testing set appears in the top. The GPU model provides correct inference (single instance of potted plant) but the FPGA model is incorrect (no detections). In this instance the higher precision of the GPU model weights appear to be more robust to this degree of contrast reduction.*



(a) *Original.*



(b) *GPU Inference.*



(c) *FPGA Inference.*

Figure 6.30: YOLO version 3 example testing image with contrast reduction degradation at 20.0%. *The original image from the VOC2007 testing set appears in the top. Both the GPU model and FPGA provides incorrect inference (ground truth is two instances of class horse). Both models provide the correct level before degradation, however, at 20% contrast reduction neither model can successfully classify the image.*

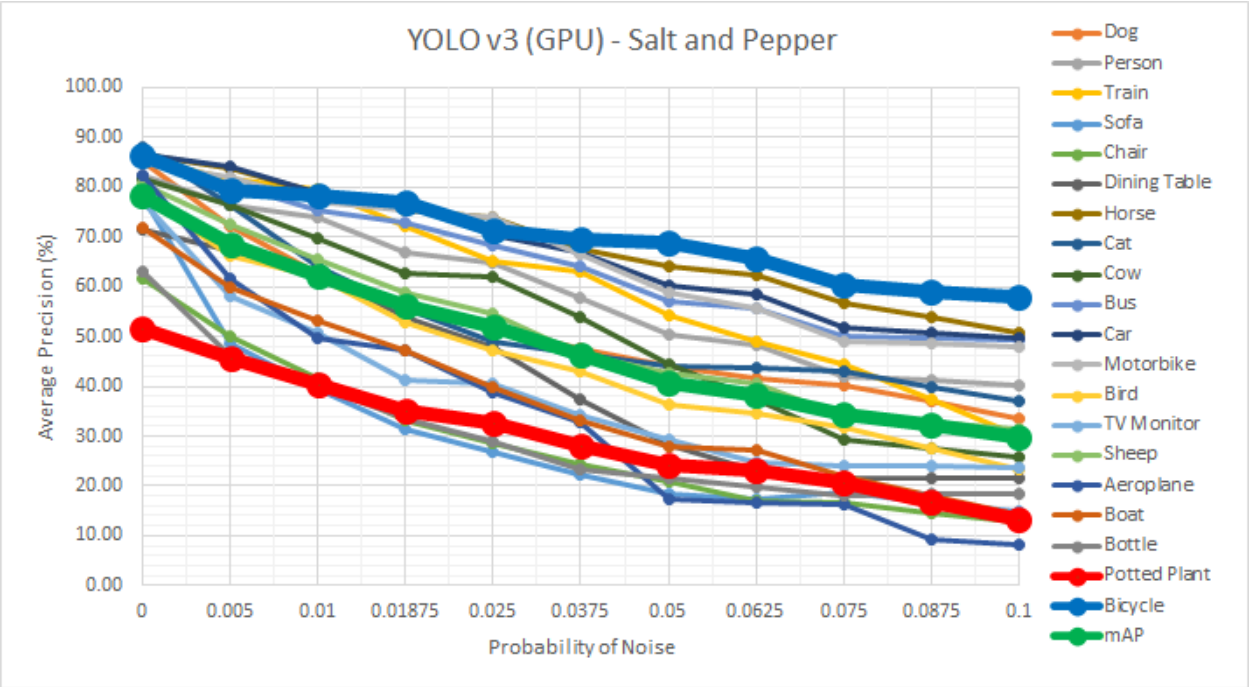


Figure 6.31: Results of salt-and-pepper degradation on YOLO version 3 GPU implementation average precision as evaluated on the VOC2007 testing set (4,952 images). *With no noise (a parameter of 0) the mAP for all classes (bold green line) is observed to be 78.46%. For no salt-and-pepper noise the best performing class ("bicycle" - bold blue line) has an average precision of 86.47% and the worst performing class ("potted plant" - bold red line) has an average precision of 51.68%. For our experiments we cut off the probability of noise parameter to 0.1.*

the bold green line reflects the performance of the mean average precision (mAP) for all classes, the bold blue line reflects the performance of the best performing class (bicycle) and the bold red line the the performance of the worst performing class (potted plant). With no noise (a parameter of 0) the mAP for all classes is observed to be 77.44%. For no salt-and-pepper noise the best performing class (bicycle) has an average precision of 86.92% and the worst performing class (potted plant) has an average precision of 50.46%. As expected, the greater the salt-and-pepper parameter the lower the calculated average precision. For our experiments we cut off the probability of noise parameter to 0.1.

Figure 6.33 compares the average precision of the GPU implementation of YOLO version 3 versus the FPGA counterpart when degrading via the salt-and-pepper noise. The GPU implementation has a mAP of 78.46% as compared to the FPGA implementation of 77.44% (a difference of 1.02%) when no salt-and-pepper degradation is present (probability of noise is 0). Unlike with the contrast reduction degradation, the FPGA implementation appears to have a similar degree of robustness to this degradation type as compared to the GPU implementation. This similar robustness to salt-and-pepper noise between the GPU and FPGA models was also observed with the ResNet-50 experiments. The most likely explanation for this is that since salt-and-pepper degradation is a discreet operation instead of continuous (as is contrast reduction) the integer based weights of the FPGA model react nearly identically to the floating point based weights of the GPU model.

Finally, in this section comparing the results of salt-and-pepper degradation between the GPU and FPGA models, we provide visual examples from the VOC2007 testing set to illustrate our statistical findings. We chose a probability parameter of 0.05 to accommodate easy human readability in these examples images. Figure 6.34 provides an example testing image where both the GPU and FPGA models made the proper prediction (two instances of class bicycle). Figure 6.35 has the GPU predicting the correct class (a single instance of class bicycle) but the FPGA model predicting incorrectly (no detections of class bicycle). Figure 6.36 shows an instance where both the GPU and FPGA provided incorrect inferences to the ground truth of 9 instances of the class aeroplane - GPU labeled 2 correct aeroplanes and FPGA labeled 1 correct aeroplane. To the best of our knowledge, there were no instances where the FPGA model provided a correct inference for a degraded image that the GPU model predicted incorrectly. This finding further provides evidence that differences between the inference accuracies of the two models arise from the lower precision weights leveraged by the FPGA model. However, as detailed in Chapter 4 and reviewed at the end of this section, the lower precision weights result in a slightly less accurate model but provide the hardware platform for significant gains in SWaP parameters to maximize Edge performance.

6.2.3 Improvement of Precision with Gamma Correction Applied to Contrast Reduction Degradation on GPU and FPGA

Figure 6.37 explores the degree to which gamma correction can reduce the effects of the contrast reduction based degradation to the mean average precision (mAP) for all classes on our GPU implementation of YOLO version 3. Each line in the graph represents the response of a particular contrast reduction parameter across the range of gamma correction parameters of 1 to 2. For example, with a contrast reduction parameter of 13.3% and a

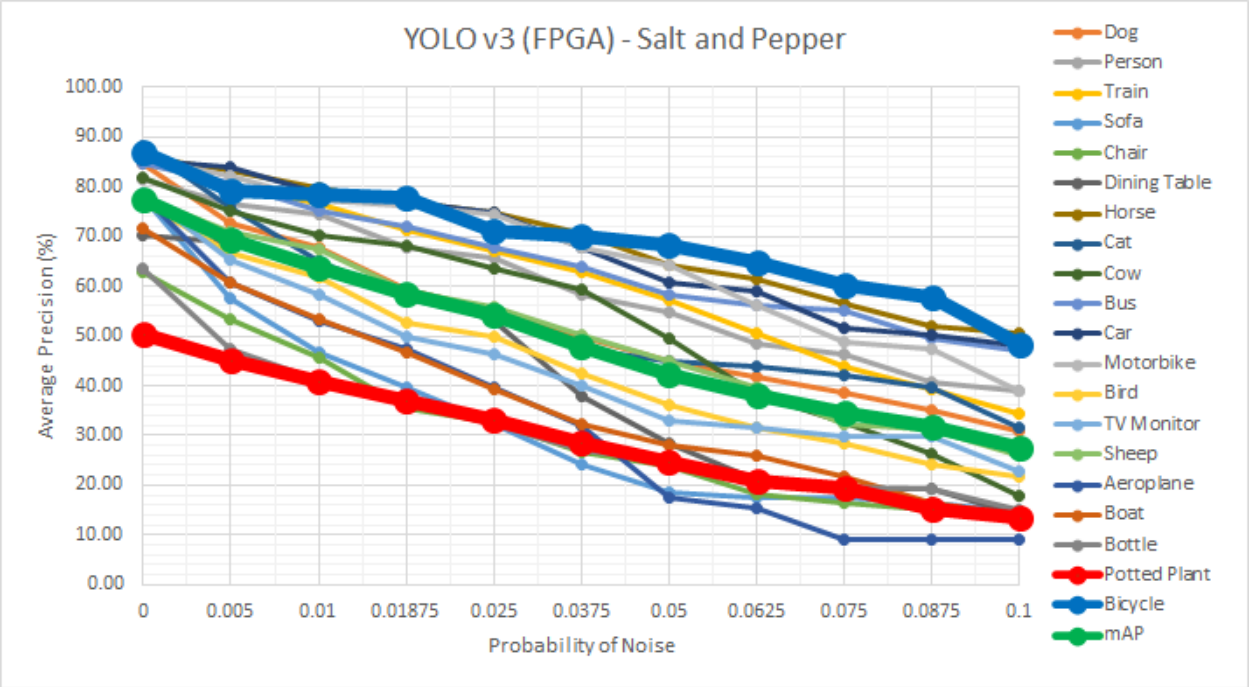


Figure 6.32: Results of salt-and-pepper degradation on YOLO version 3 FPGA implementation average precision as evaluated on the VOC2007 testing set (4,952 images). *With no noise (a parameter of 0) the mAP for all classes (bold green line) is observed to be 77.44%. For no salt-and-pepper noise the best performing class ("bicycle" - bold blue line) has an average precision of 86.92% and the worst performing class ("potted plant" - bold red line) has an average precision of 50.46%. As expected, the greater the salt-and-pepper parameter the lower the calculated average precision. For our experiments we cut off the probability of noise parameter to 0.1.*

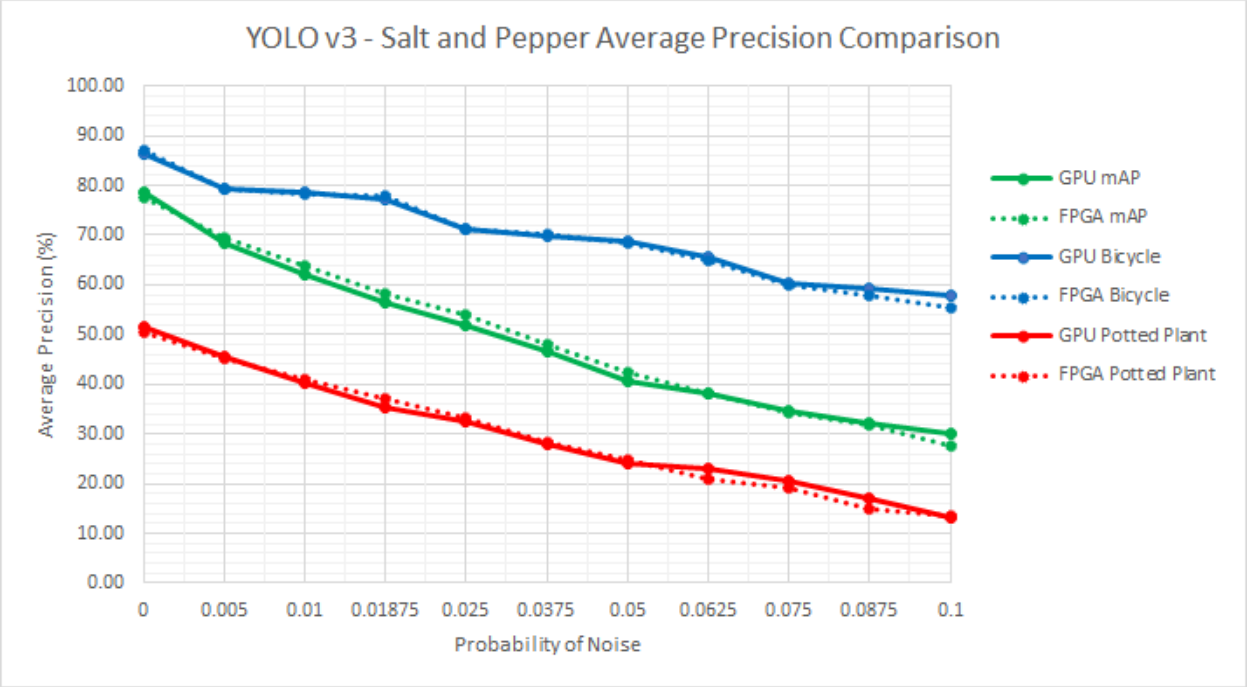
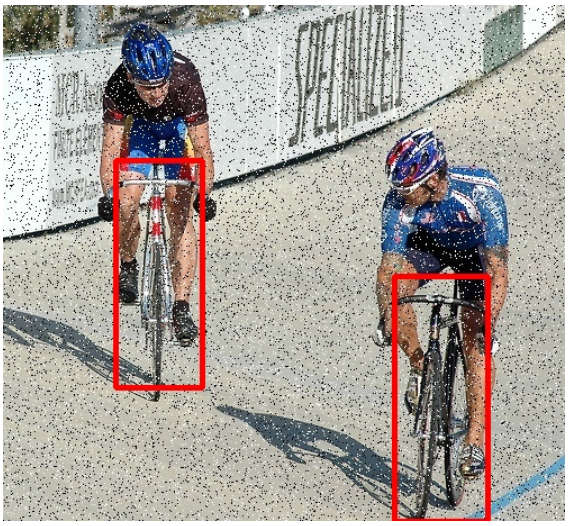


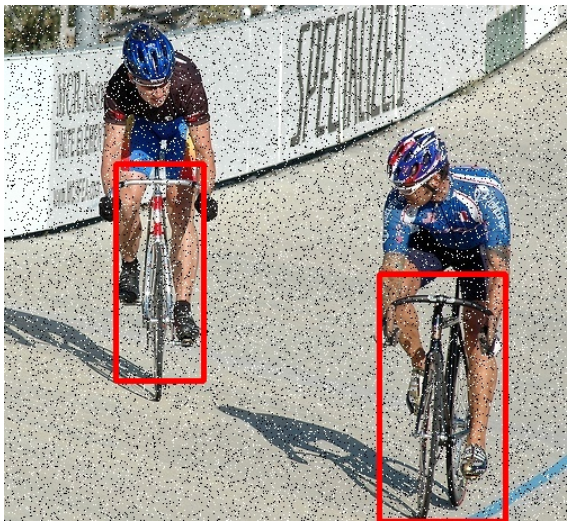
Figure 6.33: Comparison of the results of salt-and-pepper degradation on mean average precision (mAP) for YOLO version 3 on GPU versus FPGA implementation for the VOC2007 testing set (4,952 images). *GPU results appear in solid lines and FPGA results in dashed lines. We report the mAP for all classes (green lines), the best class ("bicycle" - blue lines), and the worst class ("potted plant" - red lines) in this figure. The GPU implementation has a mAP of 78.46% as compared to the FPGA implementation of 77.44% (a difference of 1.02%) when no salt-and-pepper degradation is present (probability of noise is 0). As was the case for the ResNet-50 architecture, the FPGA implementation appears to have a similar degree of robustness to this degradation type as compared to the GPU implementation. The most likely explanation for this is that since salt-and-pepper degradation is a discreet operation instead of continuous (as is contrast reduction) the integer based weights of the FPGA model react nearly identically to the floating point based weights of the GPU model.*



(a) *Original.*



(b) *GPU Inference.*



(c) *FPGA Inference.*

Figure 6.34: YOLO version 3 example testing image with salt-and-pepper degradation at 0.05. *The original image from the VOC2007 testing set appears in the top. Both GPU and FPGA models make the correct inference (two instances of class bicycle). For this particular instance the degradation level was not severe enough to cause a mislabeling for either hardware model.*



(a) *Original.*



(b) *GPU Inference.*

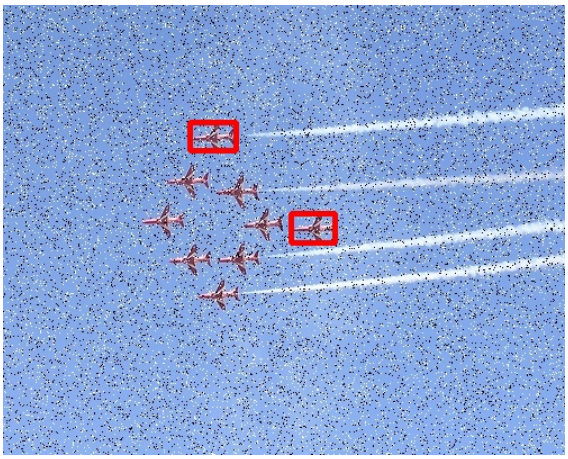


(c) *FPGA Inference.*

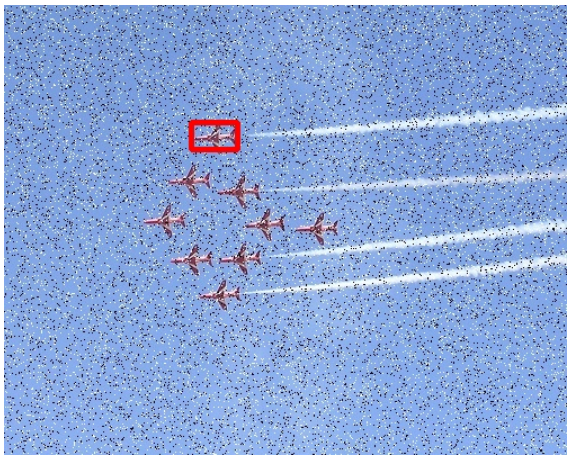
Figure 6.35: YOLO version 3 example testing image with salt-and-pepper degradation at 0.05. *The original image from the VOC2007 testing set appears in the top. The GPU model provides correct inference (single instance of class bicycle) but the FPGA model is incorrect (no instance of class bicycle). In this instance the higher precision of the GPU model weights appear to be more robust to this degree of salt-and-pepper degradation.*



(a) *Original.*



(b) *GPU Inference.*



(c) *FPGA Inference.*

Figure 6.36: YOLO version 3 example testing image with salt-and-pepper degradation at 0.05. *The original image from the VOC2007 testing set appears in the top. Both the GPU model and FPGA provide incorrect inference for the ground truth of 9 instances of class aeroplane. Both models provide the correct level before degradation, however, at salt-and-pepper degradation with a noise parameter of 0.05 neither model can successfully classify the image.*

gamma correction factor of 1 (no correction) the mAP is 70.94%. The optimal gamma factor for this contrast reduction parameter is observed at 1.5 returning the mAP to 75.90% (an increase of 4.96%). The other corrections in Figure 6.37 show a similar response. We choose to display only contrast reduction parameters up to 2.5% since beyond this the degradation is so severe that barely any increase in mAP can be had no matter the extent of correction of this sort. We observe that contrast reduced images and videos can have their quality significantly improved with optimal correction methods which also increases YOLO version 3 mAP. Given that images and videos with reduced contrast are common for in-the-wild imaging, this significant increase in mAP is highly advantageous for Edge systems.

Figure 6.38 explores the degree to which gamma correction can reduce the effects of the contrast reduction based degradation to the mean average precision (mAP) for all classes on our FPGA implementation of YOLO version 3. Each line in the graph represents the response of a particular contrast reduction parameter across the range of gamma correction parameters of 1 to 2. For example, with a contrast reduction parameter of 13.3% and a gamma correction factor of 1 (no correction) the mAP is 65.34%. The optimal gamma factor for this contrast reduction parameter is observed at 1.5 returning the mAP to 74.92% (an increase of 9.58%). The other corrections in Figure 6.38 show a similar response. We choose to display only contrast reduction parameters up to 2.5% since beyond this the degradation is so severe that barely any increase in mAP can be had no matter the extent of correction of this sort. Similar to the GPU model experiment, we observe that contrast reduced images and videos can have their quality significantly improved with optimal correction methods which also increases the FPGA YOLO version 3 mAP. Given that images and videos with reduced contrast are common for in-the-wild imaging, this significant increase in mAP is highly advantageous for Edge systems.

Figure 6.39 compares the mean average precision (mAP) of the GPU implementation of YOLO version 3 versus the FPGA counterpart when mitigating the effects of the contrast reduction degradation with gamma correction. For reference, the GPU implementation has a mAP of 78.46% when no degradation is present compared to that of 77.44% for the FPGA implementation. We chose to use three illustrative contrast reduction parameters to reflect the overall response of GPU versus FPGA correction (namely, 13.3%, 4.0% and 2.5%). When the contrast reduction parameter is 13.3% and no correction is applied (gamma parameter is 1) the GPU mAP is 70.94% whereas the FPGA mAP is 65.34%. With an optimal gamma correction of 1.5 the GPU implementation at contrast reduction parameter of 13.3% raises to 75.90% (a 5.04% increase). Similarly, with an optimal gamma correction of 1.5 the FPGA implementation at a contrast reduction parameter of 13.3% raises to 74.92% (a 9.58%). This trend of a higher starting mAP for the GPU but a higher mAP correction for the FPGA appears to be the trend for the gamma correction of contrast reduction degradation for the YOLO version 3 model. This same trend was also observed in our ResNet-50 experiments. Therefore, the FPGA model can provide comparable robustness as the GPU model to contrast reduction noise while being delivered with far superior SWaP characters better suited for Edge applications.

Figure 6.40 displays an example of model inference as compared between the GPU and FPGA implementation for the case when an image from the VOC2007 testing dataset is degraded by contrast reduction noise with a parameter of 20.0% and then corrected with an optimal gamma correction parameter. The original image appears in the top left. The

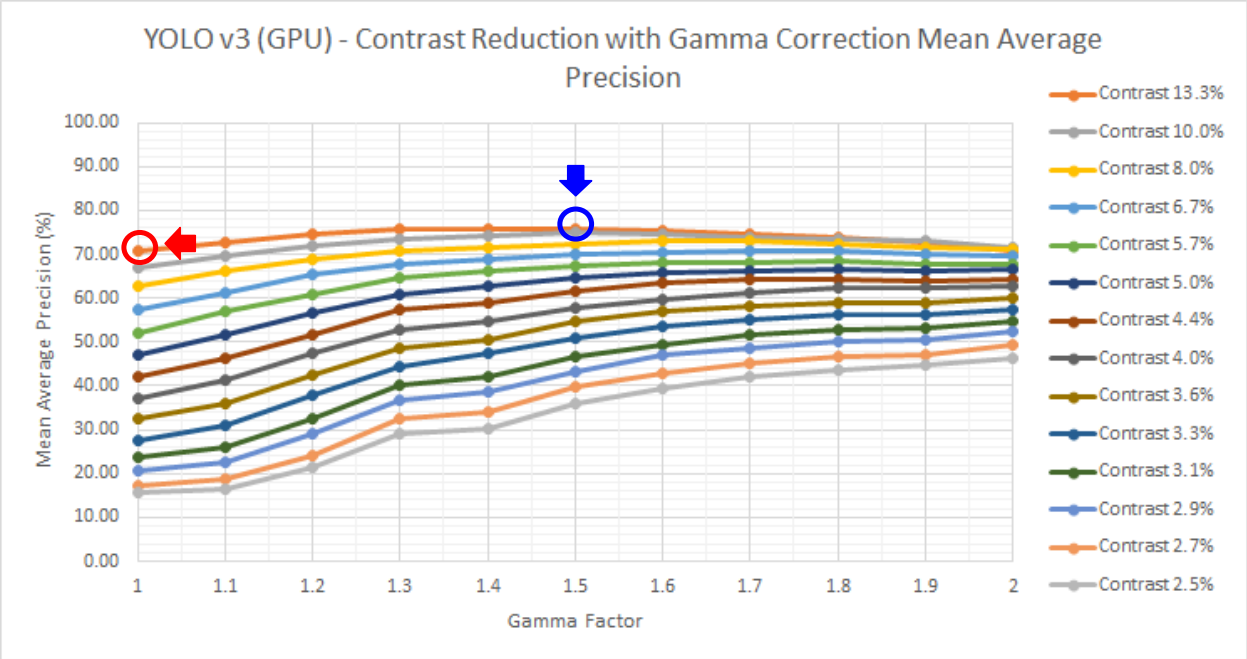


Figure 6.37: The results of gamma correction at various parameters when applied to contrast reduction based degradation for the GPU implementation of YOLO version 3 for the VOC2007 testing set (4,952 images). *Each line in the graph represents the response of a particular contrast reduction factor across the range of gamma correction parameters of 1 to 2. For example, with a contrast reduction parameter of 13.3% (top orange line) and a gamma correction factor of 1 (no correction) the mAP is 70.94% (see red arrow). The optimal gamma factor for this contrast reduction parameter is observed at 1.5 returning the mAP to 75.90% (see blue arrow) - an increase of 4.96%. The other corrections in this graph show similar response with some achieving significant improvements in mAP. Therefore, we observe that contrast reduced images and videos can have their quality significantly improved with optimal correction methods which also increases YOLO version 3 mAP. Given that images and videos with reduced contrast are common for in-the-wild imaging, this significant increase in mAP is highly advantageous for Edge systems.*

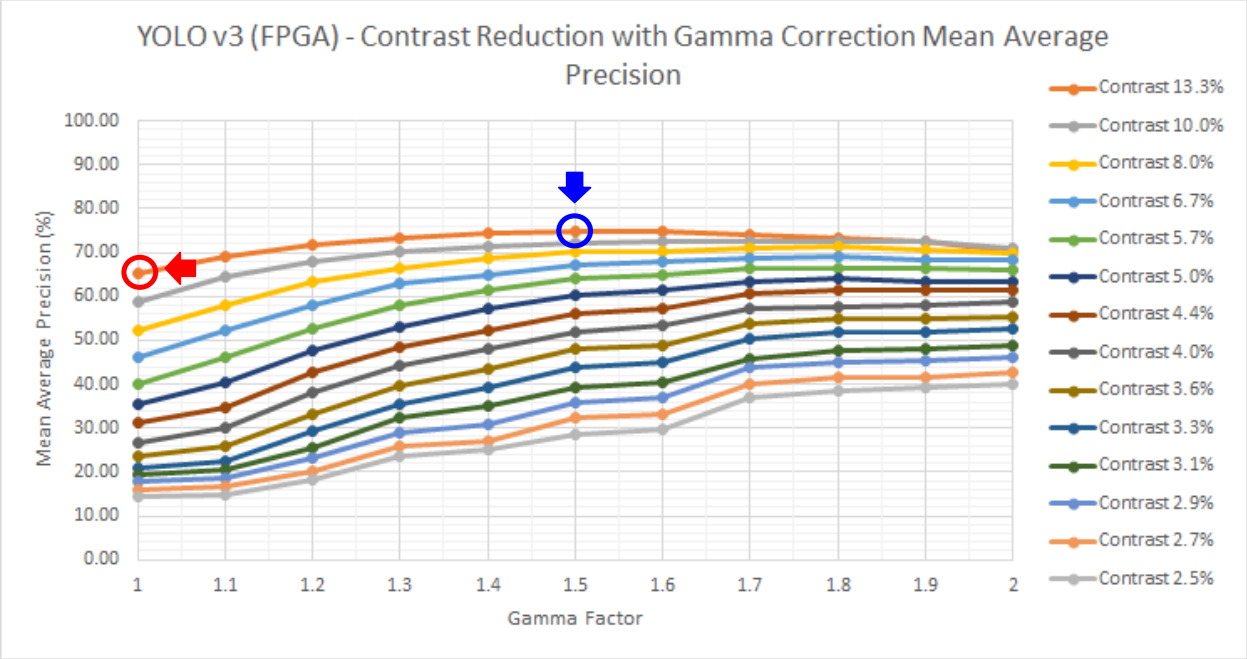


Figure 6.38: The results of gamma correction at various parameters when applied to contrast reduction based degradation for the FPGA implementation of YOLO version 3 for the VOC2007 testing set (4,952 images). *Each line in the graph represents the response of a particular contrast reduction factor across the range of gamma correction parameters of 1 to 2. For example, with a contrast reduction parameter of 13.3% (top orange line) and a gamma correction factor of 1 (no correction) the mAP is 65.34% (see red arrow). The optimal gamma factor for this contrast reduction parameter is observed at 1.5 returning the mAP to 74.92% (see blue arrow) - an increase of 9.58%. The other corrections in this graph show similar response with some achieving significant improvements in mAP. As with the GPU model, we observe that contrast reduced images and videos can have their quality significantly improved with optimal correction methods which also increases the FPGA YOLO version 3 model mAP. Given that images and videos with reduced contrast are common for in-the-wild imaging, this significant increase in accuracy is highly advantageous for Edge systems.*

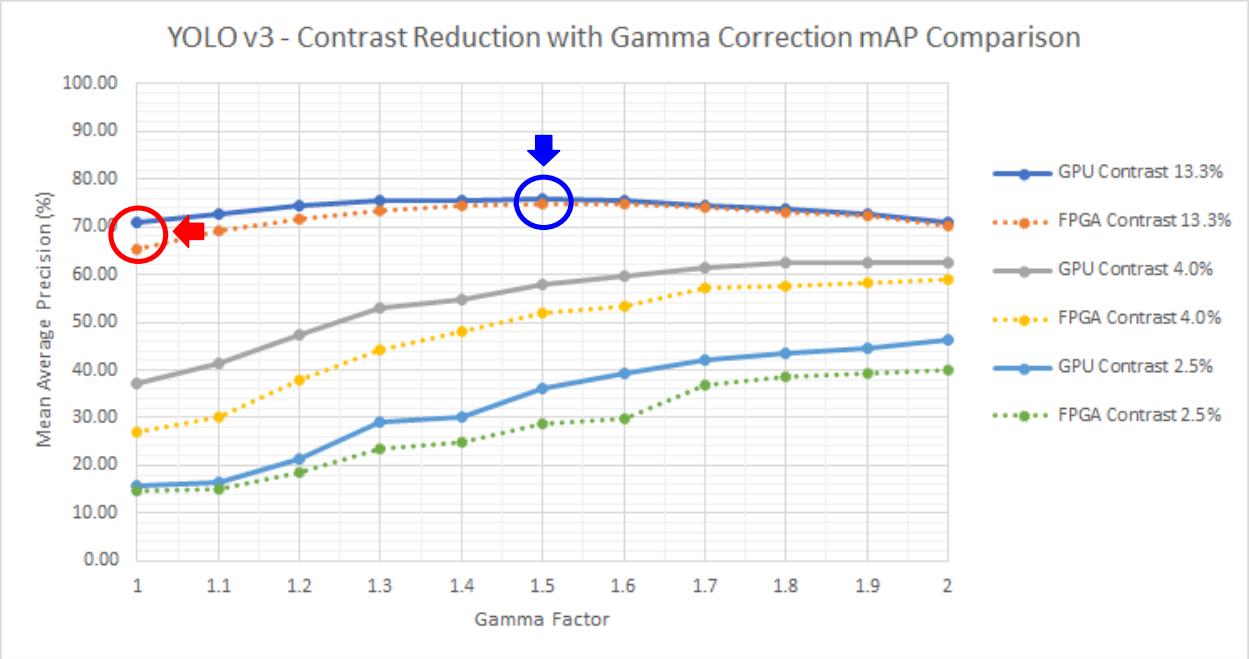


Figure 6.39: Side-by-side comparison of the results of gamma correction on contrast reduction based degradation for YOLO version 3 on GPU versus FPGA implementations. *Three illustrative low contrast parameters were selected to compare implementation performance: 13.3%, 4.0%, and 2.5% (solid lines for GPU and dashed lines for FPGA). For example, when the contrast reduction parameter is 13.3% and no correction is applied (gamma parameter is 1) the GPU mAP is 70.94% whereas the FPGA mAP is 65.34% (see red arrow). With an optimal gamma correction of 1.5 the GPU implementation at contrast reduction parameter of 13.3% raises to 75.90% (see blue arrow) - a 5.04% increase. Similarly, with an optimal gamma correction of 1.5 the FPGA implementation at a contrast reduction parameter of 13.3% raises to 74.92% (see blue arrow) - an increase of 9.58%. For this example, after optimal correction the FPGA mAP returns to within 0.98% of the GPU mAP. Therefore, even though the contrast reduction degradation impacts the FPGA model more severely, the gamma correction method improves the FPGA model at a higher rate than the GPU model. After optimal correction, the FPGA model shows similar robustness to contrast reduction as compared to the GPU model. This similar robustness coupled with the FPGA's advantage of far superior SWaP characteristics make it the superior implementation for deployment in Edge applications.*

top right image displays the results after contrast reduction is applied with a parameter of 20.0%. The bottom two images display optimal gamma correction of 1.5 applied to the contrast reduced image and the classification labels after correction is applied (GPU result on bottom left and FPGA result on bottom right). The images in this figure represent a single sample test image that failed for both GPU and FPGA when degraded by contrast reduction parameter 20.0% but are correctly labeled after correction.

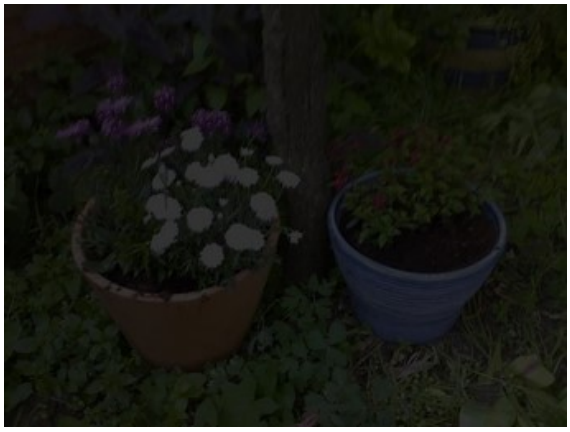
6.2.4 Improvement of Precision with Median Filter Correction Applied to Salt-and-Pepper Degradation on GPU and FPGA

Figure 6.41 explores the degree to which median filter correction can reduce the effects of the salt-and-pepper based degradation to the mean average precision (mAP) for our GPU implementation of YOLO version 3. Each line in the graph represents the response of a particular salt-and-pepper probability parameter corrected with various median filter kernel sizes. The kernel size of "None" indicates no correction and is used to illustrate the large increase in mAP that the median filter provides. For a reference, the Prob 0.0 line in Figure 6.41 at a "None" kernel size provides a mAP of 78.46%, this is reflective of no degradation and no corresponding correction. As an example, the line (bottom blue line) with the most severe salt-and-pepper degradation occurs with the probability parameter of 0.1. With no correction it has a mAP of 29.9%. However, with an optimal correction kernel size of 5x5 the mAP is increased to 66.79%, which represents a significant increase of 36.89%. Of additional note is that simply applying a median filter kernel of size 3x3 can greatly increase accuracy regardless of the salt-and-pepper degradation parameter. Therefore, we observe that salt-and-pepper degraded images and videos can have their quality significantly improved when optimal correction is applied which in turn increases YOLO version 3 maP. Given that images and videos with salt-and-pepper noise are common for in-the-wild imaging, this significant increase in accuracy is highly advantageous for Edge systems.

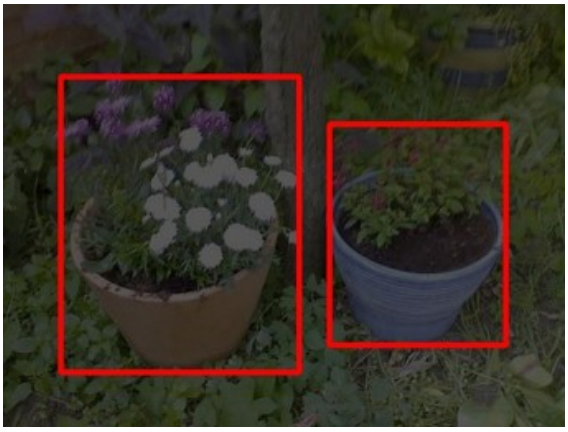
Figure 6.42 explores the degree to which median filter correction can reduce the effects of the salt-and-pepper based degradation to the mean average precision (mAP) for our FPGA implementation of YOLO version 3. Each line in the graph represents the response of a particular salt-and-pepper probability parameter corrected with various median filter kernel sizes. The kernel size of "None" indicates no correction and is used to illustrate the large increase in mAP that the median filter provides. For a reference, the Prob 0.0 line in Figure 6.42 at a "None" kernel size provides a mAP of 77.44% (approximately 1% lower than the GPU implementation), this is reflective of no degradation and no corresponding correction. As an example, the line (bottom blue line) with the most severe salt-and-pepper degradation occurs with the probability parameter of 0.1. With no correction it has a mAP of 27.60%. However, with an optimal correction kernel size of 5x5 the mAP is increased to 67.56%, which represents an increase of 39.96%. The other corrections show a similar response of significant improvement from salt-and-pepper degradation when optimal median filter correction is applied. Similar to the GPU model experiment, we observe that salt-and-pepper degraded images and videos can have their quality significantly improved when optimal correction is applied which also increases the FPGA YOLO version 3 mAP. Given



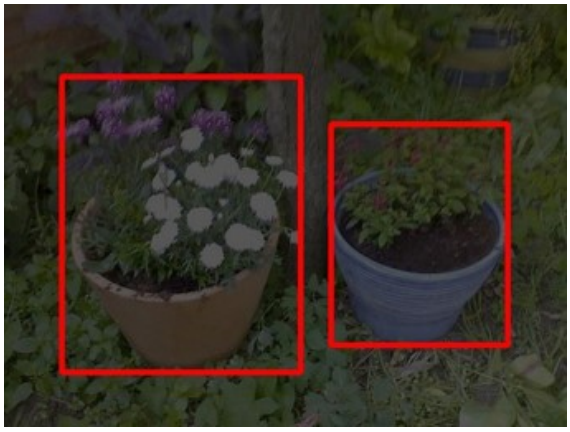
(a) *Original.*



(b) *Contrast at 20.0%.*



(c) *GPU Inference - Gamma Corrected.*



(d) *FPGA Inference - Gamma corrected.*

Figure 6.40: YOLO version 3 example testing image with contrast reduction degradation (20.0%) and gamma correction (1.5). *The original image from the ImageNet testing set appears in the top left. The top right image displays this image after contrast reduction is applied with a parameter of 20.0%. Both the GPU and FPGA failed to properly classify this image at contrast reduction degradation at 20.0% but both provide the correct inference class label (two instance of class potted plant) when an optimal gamma correction of 1.5 is applied (GPU result in bottom left and FPGA result in bottom right).*

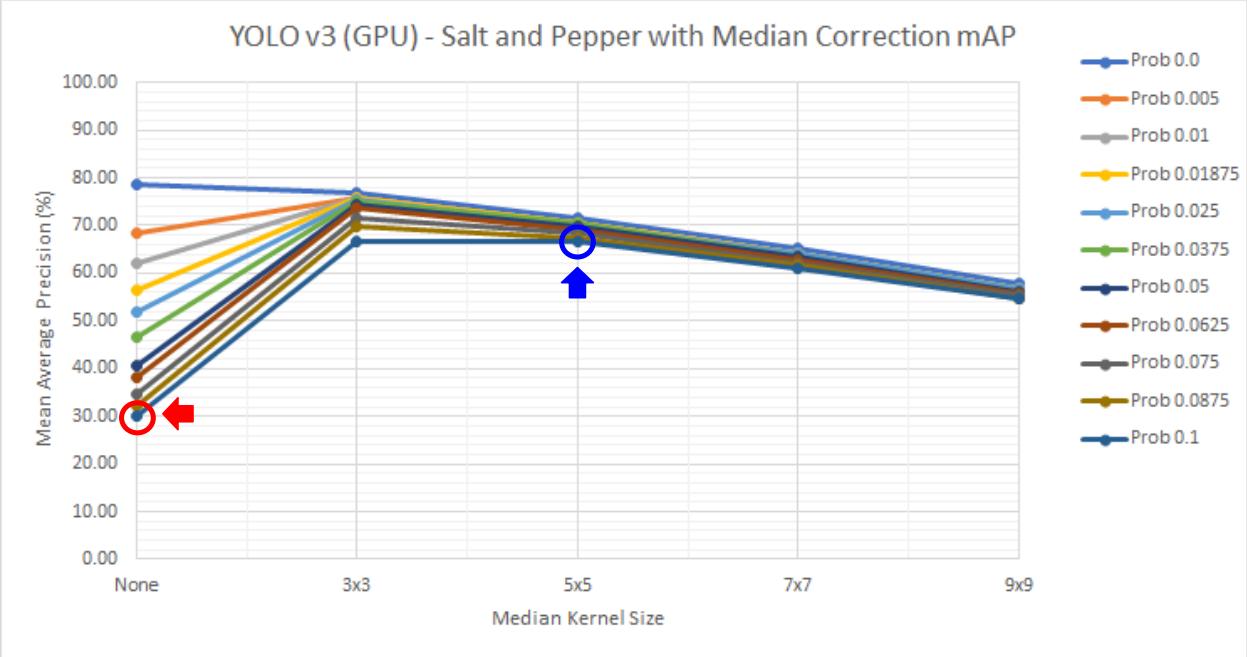


Figure 6.41: The results of median filter correction at various kernel sizes when applied to salt-and-pepper based degradation for the GPU implementation of YOLO version 3 for the VOC2007 testing set (4,952 images). *Each line in the graph represents the response of a particular salt-and-pepper degradation probability of noise parameter when corrected by several median filters of various kernel sizes. The kernel size of "None" is used to indicate no correction is applied. The top line reflects the case where no salt-and-pepper degradation is introduced (probability of noise parameter of 0.0) and is used to illustrate that the median filtering correction can have a negative impact upon image quality by causing a blurring effect if the kernel size is set too large. For example, the bottom blue line represents the most severe salt-and-pepper degradation with the probability parameter of 0.1. With no correction it has a mAP of 29.9% (see red arrow). However, with an optimal correction kernel size of 5x5 the mAP is increased to 66.79% (see blue arrow), which represents a significant increase of 36.89%. Of additional note is that simply applying a median filter kernel of size 3x3 can greatly increase accuracy regardless of the salt-and-pepper degradation parameter. Therefore, we observe that salt-and-pepper degraded images and videos can have their quality significantly improved when optimal correction is applied which in turn increases YOLO version 3 mAP. Given that images and videos with salt-and-pepper noise are common for in-the-wild imaging, this significant increase in accuracy is highly advantageous for Edge systems.*

that images and videos with salt-and-pepper noise are common for in-the-wild imaging, this significant increase in accuracy is highly advantageous for Edge systems.

Figure 6.43 compares the mAP of the GPU implementation of YOLO version 3 versus the FPGA counterpart when mitigating the effects of the salt-and-pepper degradation with median filter correction. For reference, the GPU implementation has a mAP of 78.46% when no degradation is present compared to that of 77.44% for the FPGA implementation. We chose to use three illustrative salt-and-pepper probability of noise parameters to reflect the overall response of GPU versus FPGA correction (namely, 0.005, 0.05 and 0.1). For example, when the probability of noise parameter is 0.05 and no correction is applied (no median filter applied, i.e. "None" in Figure 6.43) the GPU mAP is 40.69% whereas the FPGA mAP of 42.37%. With an optimal median filter size of 3x3 the GPU implementation at salt-and-pepper probability of 0.05 raises to 74.52% (a 33.83% increase). Similarly, with an optimal median filter size of 3x3 the FPGA implementation at salt-and-pepper probability of 0.05 raises to 74.13% (a 31.76% increase). Unlike with the contrast reduction based degradation, with the salt-and-pepper noise both the GPU and FPGA implementations respond with similar drops in mAP for degradation and similar improvements in mAP for correction. This similar response to correction of salt-and-pepper degradation was also observed in our ResNet-50 experiments. Therefore, the FPGA model can provide identical robustness as the GPU model to salt-and-pepper degradation while being delivered with far superior SWaP characters better suited for Edge applications.

Figure 6.44 displays an example of model inference as compared between the GPU and FPGA implementation for the case when an image from the VOC2007 testing dataset is degraded by salt-and-pepper noise with a parameter of 0.05 and then corrected with an optimal median kernel filter of size 3x3. The original image appears in the top left. The top right image displays the results after salt-and-pepper degradation is applied with a parameter of 0.05. The bottom two images display optimal median filter correction with a kernel of size 3x3 applied to the salt-and-pepper degraded image and the classification labels after correction is applied (GPU result on bottom left and FPGA result on bottom right). The images in this figure represent a single sample test image that failed for both GPU and FPGA when degraded by salt-and-pepper parameter 0.05 but are correctly labeled after correction.

6.2.5 Experimental Results of YOLO Version 3 Benchmarking

Table 6.3 provides a series of metrics summarizing the contrast reduction degradation and gamma correction evaluation of the YOLO version 3 model in our experiments. This table outlines seven experiments to provide an overview of both the GPU and FPGA pipeline performances. For each of these experiments we also display the associated average SSIM score for the quality of the degraded or corrected images being evaluated in the VOC2007 dataset. As theorized in Chapter 5, the lower the SSIM score the lower the inference performance.

Take note in Table 6.3, that as with ResNet-50, the GPU and FPGA implementation of YOLO version 3 have near identical performance at the base experiment where no degradation is introduced. Also as with ResNet-50, the FPGA implementation appears to be consistently more impacted by the contrast reduction degradation than its GPU counterpart.

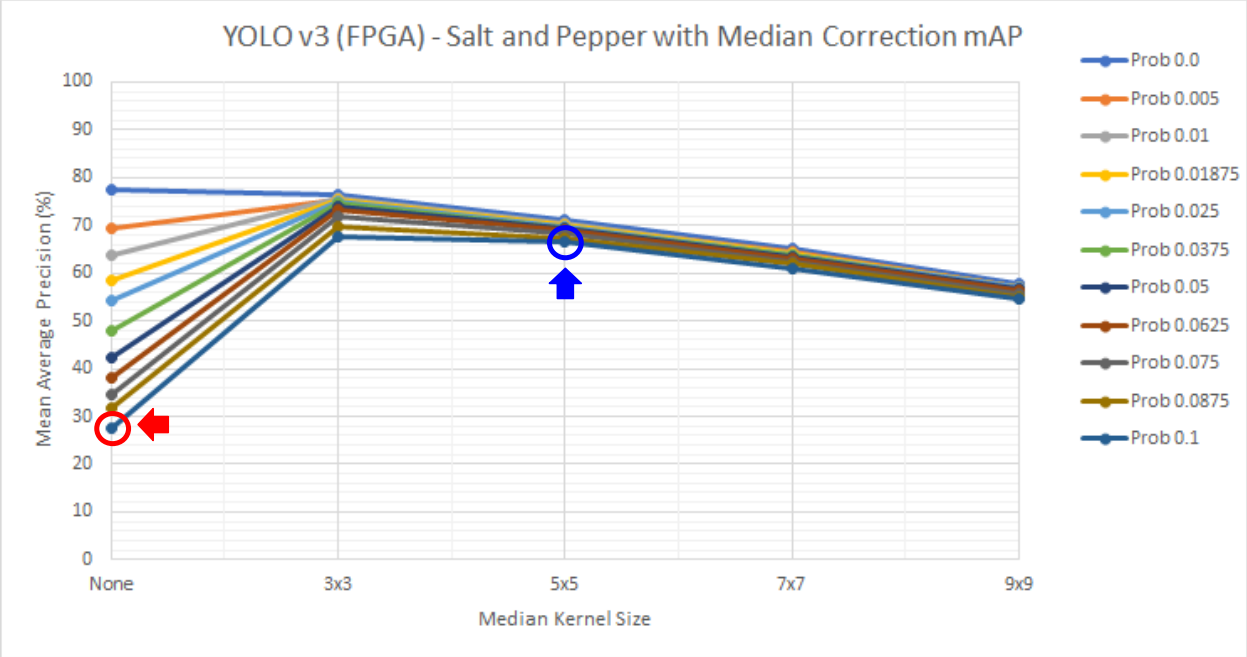


Figure 6.42: The results of median filter correction at various kernel sizes when applied to salt-and-pepper based degradation for the FPGA implementation of YOLO version 3 for the VOC2007 testing set (4,952 images). *Each line in the graph represents the response of a particular salt-and-pepper degradation probability of noise parameter when corrected by several median filters of various kernel sizes. The kernel size of "None" is used to indicate no correction is applied. The top line reflects the case where no salt-and-pepper degradation is introduced (probability of noise parameter of 0.0) and is used to illustrate that the median filtering correction can have a negative impact upon image quality by causing a blurring effect if the kernel size is set too large. For example, the bottom blue line represents the most severe salt-and-pepper degradation with the probability parameter of 0.1. With no correction it has a mAP of 27.60% (see red arrow). However, with an optimal correction kernel size of 5x5 the mAP is increased to 67.56% (see blue arrow), which represents a significant increase of 39.96%. Of additional note is that simply applying a median filter kernel of size 3x3 can greatly increase accuracy regardless of the salt-and-pepper degradation parameter. As with the GPU model, we observe that salt-and-pepper degraded images and videos can have their quality significantly improved when optimal correction is applied which in turn increases the FPGA YOLO version 3 mAP. Given that images and videos with salt-and-pepper noise are common for in-the-wild imaging, this significant increase in accuracy is highly advantageous for Edge systems.*

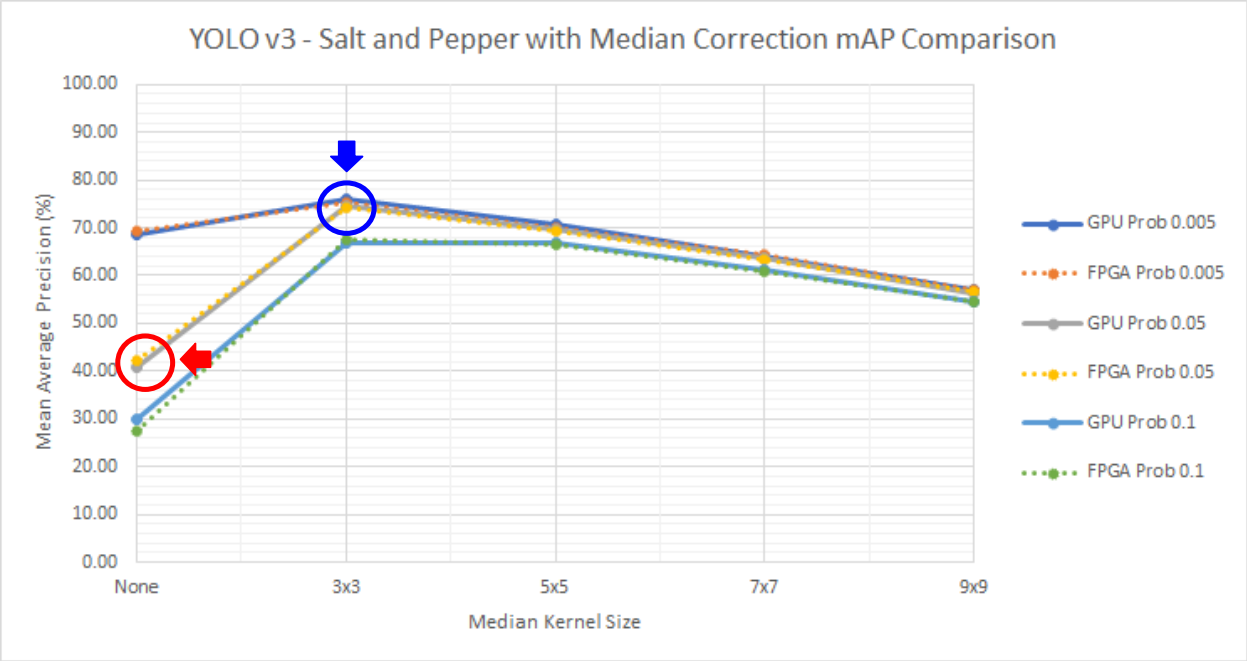


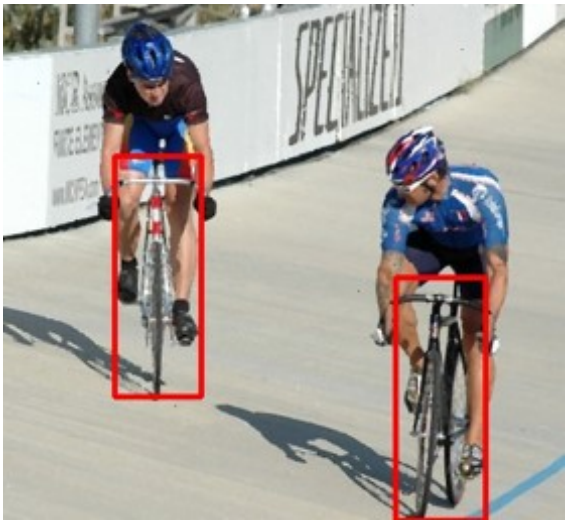
Figure 6.43: Side-by-side comparison of the results of median filter correction on salt-and-pepper based degradation for YOLO version 3 on GPU versus FPGA implementations. *Three illustrative salt-and-pepper probability of noise parameters were selected to compare implementation performance: 0.005, 0.05, and 0.1 (solid lines for GPU and dashed lines for FPGA). For example, when the probability of noise parameter is 0.05 and no correction is applied (no median filter applied, i.e. "None") the GPU mAP is 40.69% whereas the FPGA mAP of 42.37% (see red arrow). With an optimal median filter size of 3x3 the GPU implementation at salt-and-pepper probability of 0.05 raises to 74.52% (see blue arrow) - a 33.83% increase. Similarly, with an optimal median filter size of 3x3 the FPGA implementation at salt-and-pepper probability of 0.05 raises to 74.13% (see blue arrow) - a 31.76% increase. For this example, after optimal correction the FPGA returns to within 0.39% of the GPU mAP. Unlike with the gamma corrected contrast reduced experiments, the FPGA has near identical robustness as the GPU to salt-and-pepper degradation when optimal median filter correction is applied. This near identical robustness coupled with the FPGA's advantage of far superior SWaP characteristics make it the superior implementation for deployment in Edge applications.*



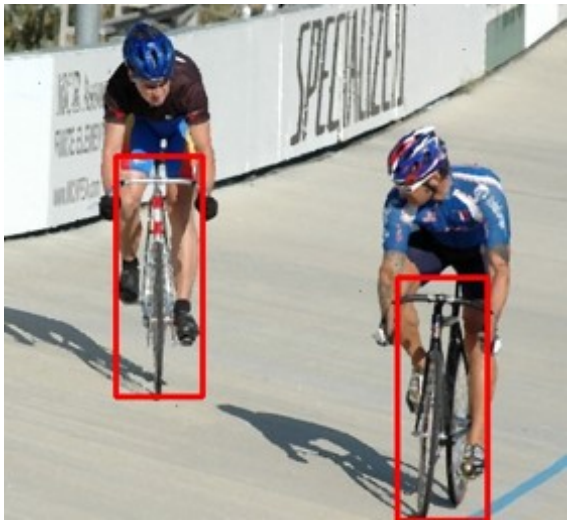
(a) *Original.*



(b) *Salt-and-Pepper at 0.05.*



(c) *GPU Inference - Median Filter Corrected.*



(d) *FPGA Inference - Median Filter corrected.*

Figure 6.44: YOLO version 3 example testing image with salt-and-pepper degradation (0.05) and median filter correction (3x3). *he original image from the ImageNet testing set appears in the top left. The top right image displays this image after salt-and-pepper degradation is applied with a parameter of 0.05. Both the GPU and FPGA failed at salt-and-pepper degradation at 0.05 but both provide the correct inference class label (two instances of class bicycle) when an optimal median filter correction of 3x3 is applied (GPU result in bottom left and FPGA result in bottom right.)*

Our intuition once again points to the more precise weights leverage by the GPU model which requires higher SWaP characteristics to deploy operationally.

However, we observe once more that correction methods can offer significant improvements in inference for both GPU and FPGA (see table 6.3). Our experiments seem to indicate the the FPGA-based model of YOLO version 3 is more impacted by degradation and improved by correction as compared to the GPU model. Therefore, an FPGA-based Edge system would ideally be outfitted with correction methods for low light conditions to maintain as high as inference performance as possible.

Table 6.4 provides a series of metrics summarizing the salt-and-pepper degradation and median filter correction evaluation of the YOLO version 3 model in our experiments. This table outlines seven experiments to provide an overview of both the GPU and FPGA pipeline performances. For each of these experiments we also display the associated average SSIM score for the quality of the degraded or corrected images being evaluated in the VOC2007 dataset. As theorized in Chapter 5, the lower the SSIM score the lower the inference performance.

As with ResNet-50, the GPU and FPGA models of YOLO version 3 have nearly identical inference performance when degraded with the salt-and-pepper noise. Given that this same response was seen for ResNet-50 this lends evidence to our assumption that floating point and quantized weights respond similarly to this noise type. With the much lower SWaP profile of the FPGA system this allows for GPU performance with a much smaller footprint.

As with the ResNet-50 experiment, the correction method for the salt-and-pepper degradation provides equivalent performance increase for both the GPU and FPGA at the various experiments. Both the GPU and FPGA models can be greatly corrected by the application of the optimal median filter size to offset the impact of the salt-and-pepper noise. This response shows the general robustness of both floating point and quantized model types to this degradation type.

6.3 Findings From Pipeline Experimentation Results

In this chapter we evaluated the performance of two models (ResNet-50 and YOLO version 3) implemented in both GPU and FPGA pipelines to determine their robustness to input degradation and performance improvement for post-degradation image correction. Both models were tested with two degradation types: contrast reduction and salt-and-pepper noise. Contrast reduction was used to emulate low light conditions often seen in deployed Edge systems. The salt-and-pepper noise was used to emulate sensor noise caused by overheating or mechanical vibration also common in deployed Edge systems. We chose gamma correction as a means to correct images degraded by the contrast reduction type noise and median kernel filtering as the correction for images degraded by the salt-and-pepper noise. This chapter therefore integrates the GPU vs FPGA base implementations in Chapter 4 with the quality degradation/correction metrics in Chapter 5 to present experimental results in complete end-to-end pipelines for our two models to illustrate their appropriateness for Edge deployed computer vision systems. These experiments demonstrate that the FPGA-based models have similar robustness as GPU-based models when image degradation is reduced by correction. Considering that our experimental results leverage two different

Table 6.3: Comparison of contrast reduction degradation and gamma correction on VOC2007 testing dataset of GPU (32-bit floating point model) vs. FPGA (8-bit integer quantized model) performance on YOLO version 3. *This table outlines seven experiments to provide an overview of both the GPU and FPGA pipeline performances. For each of these experiments we also display the associated average SSIM score for the quality of the degraded or corrected images being evaluated in the ImageNet dataset. As demonstrated in Chapter 4, the GPU and FPGA implementations have near identical performance at the base experiment where no contrast reduction degradation is introduced (Base experiment in the table). For example, with contrast reduction degradation of 13.3% applied the GPU mAP drops from 78.46% to 70.94% (a 7.52% decrease) and the associated SSIM scores drops from 1 to 0.1512 (a decrease of 0.8488). For the same contrast degradation with the FPGA, mAP drops from 77.44% to 65.34% (a 12.1% decrease) with the same SSIM score to reflect testing dataset quality. However, after optimal gamma correction is applied the GPU mAP returns to 75.90% (an 5.04% increase) and the FPGA returns to 74.92% (a 9.58% increase) and the SSIM score for the corrected dataset becomes 0.6996 (an increase of 0.5484). As theorized, the SSIM score can be used as a predictor for inference performance. These experiments demonstrate that the FPGA model can achieve comparable mAP to the GPU model when optimal gamma correction is applied to contrast reduction degradation. Although the FPGA model provides slightly lower mAP for this type of degradation it comes with the advantage of superior SWaP characteristics better suited for Edge applications.*

Experiment	**SSIM Score**	**GPU mAP**	**GPU Improvement**	**FPGA mAP**	**FPGA Improvement**
Base	1	78.46%	N/A	77.44%	N/A
Contrast 13.3%	0.1512	70.94%	N/A	65.34%	N/A
Contrast 13.3% optimal gamma	0.6996	75.90%	5.04%	74.92%	9.58%
Contrast 4.0%	0.0403	37.19%	N/A	26.81%	N/A
Contrast 4.0% optimal gamma	0.2398	62.61%	25.42%	58.94%	32.18%
Contrast 2.5%	0.0263	15.69%	N/A	14.55%	N/A
Contrast 2.5% optimal gamma	0.0742	46.47%	30.78%	40.00%	25.45%

Table 6.4: Comparison of salt-and-pepper degradation and median filter correction on VOC2007 testing dataset of GPU (32-bit floating point model) vs. FPGA (8-bit integer quantized model) performance on YOLO version 3. *This table outlines seven experiments to provide an overview of both the GPU and FPGA pipeline performances. For each of these experiments we also display the associated average SSIM score for the quality of the degraded or corrected images being evaluated in the ImageNet dataset. As demonstrated in Chapter 4, the GPU and FPGA implementations have near identical performance at the base experiment where no salt-and-pepper degradation is introduced (Base experiment in the table). For example, with salt-and-pepper degradation of 0.05 the GPU mAP drops from 78.46% to 40.69% (a 37.77% decrease) and the associated SSIM scores drops from 1 to 0.2601 (a decrease of 0.7399). For the same salt-and-pepper degradation with FPGA, mAP drops from 77.44% to 42.37% (a 35.07% decrease) with the same SSIM score to reflect testing dataset quality. However, after optimal median filter correction is applied the GPU mAP returns to 74.52% (a 33.83% increase) and the FPGA returns to 74.13% (a 31.76% increase) and the SSIM score for the corrected dataset becomes 0.8075 (an increase of 0.5474). As theorized, the SSIM score can be used as a predictor for inference performance. These experiments demonstrate that the FPGA model can achieve near identical mAP to the GPU model when optimal median filter correction is applied to salt-and-pepper degradation. Not only does the FPGA model provides near identical robustness for this type of degradation it comes with the advantage of superior SWaP characteristics better suited for Edge applications.*

Experiment	SSIM Score	GPU mAP	GPU Improvement	FPGA mAP	FPGA Improvement
Base	1	78.46%	N/A	77.44%	N/A
Prob. 0.005	0.8141	68.47%	N/A	69.34%	N/A
Prob. 0.005 optimal median	0.8219	75.96%	7.49%	75.46%	6.12%
Prob. 0.05	0.2601	40.69%	N/A	42.37%	N/A
Prob. 0.05 optimal median	0.8075	74.52%	33.83%	74.13%	31.76%
Prob. 0.1	0.1468	29.90%	N/A	27.60%	N/A
Prob 0.1 optimal median	0.7637	66.79%	36.89%	67.56%	39.96%

CNN architectures targeting two different computer vision domains, two different datasets, and a combination of two degradation and two correction types, we believe our results are statistically significant and can be generalized to other CNNs and input perturbations. Therefore, we have proven the FPGA-based system as the superior Edge platform given that it provides similar inference performance to the GPU-based system while being implemented in a smaller SWaP platform making it an ideal candidate for Edge AI deployment.

Chapter 7

Findings and Recommendations

The research in this thesis has proven that an FPGA-based solution is an ideal platform for an Edge deployed computer vision system. The feasibility of the FPGA-based solution being demonstrated by the various experiments laid out in the previous chapters outlining desirable SWaP characteristics, baseline inference comparison and noise robustness of the FPGA as compared to the conventional GPU counterpart. This chapter summarizes the key points of our experiments conducted in this research, maps these key points to the thesis contributions, and recommends potential future extensions to this thesis.

7.1 Summary

The work in this thesis leverages the pipeline in Figure 1.2 to provide a side-by-side performance analysis of deep learning vision models implemented in FGPA and GPU. The overarching goal of this analysis being to determine if FPGA-based solutions provide an ideal alternative to Edge deployment over their GPU counterparts. As seen in Figure 1.2, the pipeline can be configured to perform the various experiments laid out in previous chapters which demonstrate the validity of our thesis work. In this section we provide an overview of the key results for each experimental chapter in our work.

In our final experiments to prove our principal contribution we evaluate the complete FPGA and GPU computer vision inference pipelines when degradation and/or correction is introduced to the input datasets proving that our FPGA solution is a more ideal platform for Edge deployment. Both models were evaluated with our two degradation methods (contrast reduction and salt-and-pepper noise) as well as evaluated when the associated correction method (gamma correction or median filter) is applied. Contrast reduction degradation was used to emulate low light conditions often seen in deployed Edge systems. The salt-and-pepper noise was used to emulate sensor noise caused by overheating or mechanical vibration also common in deployed Edge systems. For both the ResNet-50 and YOLO version 3 models it was observed that inference performance dropped off at a quicker rate for the FPGA implementation as the contrast reduction degradation parameter was increased demonstrating that the quantized weights of the hardware model were less robust to this sort of noise. However, both the FPGA and GPU models had a near identical drop in the rate of inference performance for the same parameters when the salt-and-pepper degradation type

was introduced. Therefore, for this degradation, the FPGA and GPU demonstrated similar robustness. When gamma correction was applied to contrast reduction both of the FPGA and GPU models recovered at the same rate. The same near identical recovery of inference performance is seen in both FPGA and GPU models when median filtering is applied to the salt-and-pepper noise. Thus, when appropriate correction methods are applied, the FPGA implementation provides similar robustness to noise as well as inference performance to its conventional GPU counterpart while maintaining a much smaller SWaP profile. This result is of great significance to Edge applications such as autonomous vehicles, smart security cameras and UAVs which all would benefit from robust visions systems that can be deployed with smaller SWaP requirements.

The first minor contribution leveraged by our principal contribution was the development of an FPGA-based video processing pipeline. We achieved this using various Xilinx design tools creating first a video processing architecture which could receive a HDMI input source and output an altered signal to a HDMI output sink. Additional to this base design was the implementation of a library of video processing IP cores that could be placed within the FPGA data stream to alter the video signal as desired. This base FPGA system and library of video processing cores were implemented in several Xilinx FPGA boards and provide the experimental environment in which to conduct our later experiments. Of particular note is that our system was able to achieve 30 frames-per-second processing of videos with 1920x1080 resolution. Such high speed video processing is ideal for the real-time needs in Edge deployment.

After implementing the FPGA video processing system we extended this architecture with another minor contribution to include deep learning computer vision IP cores in the hardware processing pipeline. The CNN hardware IP cores themselves were created leveraging Xilinx's DNNDK tool. This tool takes a pre-trained model from several popular deep learning frameworks and quantizes the model weights from 32-bit floating point to 8-bit integer representation for FPGA implementation. For our side-by-side analysis of FPGA versus GPU performance we decided to target two computer vision tasks: (1) image classification via the ResNet-50 CNN architecture; and, (2) object detection via the YOLO version 3 CNN architecture. The ResNet-50 model was trained on the ImageNet-2012 training dataset (approximately 1.2 million images) and tested on the a subset of the ImageNet-2012 testing dataset (5,000 images) in which the images are assigned to 1,000 classes. The YOLO version 3 model was trained on the VOC2007 training dataset consisting of 9,963 images with 24,640 annotated objects and tested on the VOC2007 testing dataset (4,952 images) in which images can fall under 20 classes. Our FPGA models were evaluated using a Xilinx ZCU104 UltraScale+ evaluation board and the original floating point models on a Nvidia RTX 2070 Super. Model performance was measured via accuracy for the image classification task with ResNet-50 and the FPGA model was shown to have an approximate 1% drop in both top-1 and top-5 accuracy as compared to the GPU equivalent as shown in Table 4.2 while providing far superior SWaP characteristics to the GPU implementation (the FPGA achieving 2.4 fps/W whereas the GPU based-solution achieves 0.65 fps/W). Performance for the object detection task with the YOLO version 3 model was measured with mean average precision and the FPGA again delivered an approximate 1% drop in performance as compared to the original floating point model as shown in Table 4.5 while still delivering more ideal characteristics for SWaP deployment as compared to the GPU (the

FPGA achieving 0.45 fps/W whereas the GPU based-solution achieves 0.13 fps/W). With these experiments we establish that an FPGA-based solution incurs only a minimal drop in inference performance while providing far superior SWaP characteristics better suited for Edge deployment.

After establishing that the FPGA system provides near identical inference performance compared to the GPU counterpart we next wanted to explore its robustness to noisy signals. Signals in unconstrained scenarios where Edge systems are deployed are often degraded due to environmental conditions, electro-mechanical interference, or other factors. Therefore, determining if the FPGA model would provide similar robustness to noise allows for one measure of explainability when an incorrect inference is made. Our experiments evaluated the impact of two degradation methods (contrast reduction and salt-and-pepper noise) as well as their correction methods (gamma correction and median filter, respectively). Before determining the impact on inference with our FPGA and GPU models, our initial experiments provide a baseline response to SSIM score when the testing datatsets (ImageNet-2012 and VOC2007) are evaluated. For each dataset we degraded with both noises as well as corrected with the associated correction methods. As predicted, these experiments shown that we could mimic real-world noises, correct the degradation as well as provide a quantitative score to each image to provide statistical significance in our final experiments. Additionally, we show that neither evaluated dataset is biased towards any selected degradation or correction method. This revelation is also important because it shows that the response from a particular degradation or correction can be expected to be consistent across different datasets.

The several experiments carried out in this thesis combine to prove that an FPGA-based computer vision system is ideal for Edge deployed applications. Specifically, the four boxes in the thesis pipeline in Figure 1.2 map directly to our contributions which demonstrate four core findings from this thesis:

1. Degradation followed by optimally selected correction restores the FPGA model to near identical performance as the GPU counterpart providing robustness while benefiting from a much smaller SWaP footprint.

2. FPGA systems are ideal for low SWaP platforms for Edge systems that need to process real-time video data.

3. The quantized FPGA deep learning model is comparatively as accurate as the floating point GPU model for both image classification and object detection tasks.

4. Certain image degradations impact the less precise weights of the FPGA model at a greater rate than the GPU model but for other methods the impact is identical.

In our thesis we have proven that our FPGA solution is an ideal platform for Edge applications such as autonomous vehicles, smart cameras and UAVs. We hypothesize that the largest current hurdle for widespread use of the FPGA in deep learning tasks is due to design complexity of solutions in FPGA environments as opposed to standard programming environments targeting GPU which have both a larger variety of programming frameworks as well as increased ease of use.

7.2 Contributions

As laid out in Chapter 1 of this thesis, there are four contributions in this work: (1) we proved that FPGA-based CNNs have comparable robustness to image degradations as their GPU counterparts when such degradations are mitigated with optimally selected correction methods attesting the potential of the FPGA as an appropriate platform for vision systems deployed to the Edge (major contribution); (2) we developed an FPGA system for video processing, along with an accompanying library of custom processing hardware cores, to serve as the base FPGA design of the thesis pipeline and is extendable for our benchmarking experiments (minor contribution); (3) we established that FPGA-based ResNet-50 and YOLO version 3 models both incur only an approximate 1% drop in performance as compared to GPU-based models, proving similar performance across computer vision problem space, while being delivered in hardware with significant SWaP improvements (minor contribution); and, (4) we show a baseline statistical analysis of the impact of two degradation methods, along with corresponding correction methods, on the evaluated datasets via a quantitative image quality metric which illustrates that neither contain an inherent bias towards our evaluated methods (minor contribution).

Contribution 1 (major)

In our principal contribution we prove that FPGA-based CNNs have comparable robustness to image degradations as their GPU counterparts when such degradations are mitigated with optimally selected correction methods attesting the potential of the FPGA as an appropriate platform for vision systems deployed to the Edge. This contribution integrates and extends our three minor contributions and is marked by the yellow box in 1.2. Both models were evaluated with our two degradation methods (contrast reduction and salt-and-pepper noise) as well as evaluated when the associated correction method (gamma correction or median filter) is applied. For both the ResNet-50 and YOLO version 3 models it was observed that as the contrast reduction degradation parameter was increased inference performance dropped off at a quicker rate for the FPGA implementation demonstrating that the quantized weights of the hardware model were less robust to this sort of noise. With contrast reduction degradation (noise parameter = 20.0%), the GPU ResNet-50 model top-1 performance dropped by 12.96% whereas the FPGA ResNet-50 model dropped by 19.92%. Likewise, with contrast reduction degradation (noise parameter = 13.3%) the GPU YOLO version 3 model mean average precision dropped by 7.52% whereas the FPGA YOLO version 3 model mean average precision dropped by 12.1%. However, both the FPGA and GPU models had a near identical drop in the rate of inference performance for the same parameters when the salt-and-pepper degradation type was introduced. For instance, with a salt-and-pepper degradation (noise probability = 0.05) the GPU ResNet-50 top-1 accuracy dropped by 35.38% and the FPGA ResNet-50 top-1 accuracy dropped by a near identical 35.54%. With a salt-and-pepper degradation (noise probability = 0.05) the GPU YOLO version 3 mean average precision fell by 37.77% and the FPGA YOLO version 3 mean average precision fell by 35.07%. Therefore, for this degradation, the FPGA and GPU demonstrated similar robustness. When gamma correction was applied to contrast reduction degradation both of the FPGA and GPU models recovered significantly. For instance, with optimal gamma

correction applied to contrast reduction degradation (noise parameter = 20.0%), the GPU ResNet-50 model top-1 performance increased by 8.6% to achieve 70.08% and the FPGA ResNet-50 model top-1 performance increased by 12.83% to achieve 66.25%. Similarly, with optimal gamma correction applied to contrast reduction degradation (noise parameter = 13.3%), the GPU YOLO version 3 model mean average precision increases by 5.04% to achieve 75.90% and the FPGA YOLO version 3 model mean average precision increases by 9.58% to achieve 74.92%. As with the salt-and-pepper degradation, the median filter correction sees near identical response from both GPU and FPGA models. With an optimal median filter correction applied to salt-and-pepper degradation (noise probability = 0.05) the GPU ResNet-50 model top-1 accuracy increases 33.68% to achieve 72.74% performance while the FPGA ResNet-50 model top-1 accuracy increases 34.06% to achieve 71.86%. Likewise, with an optimal median filter correction applied to salt-and-pepper degradation (noise probability = 0.05) the GPU YOLO version 3 model mean average precision increases 33.83% to achieve 74.52% performance while the FPGA YOLO version 3 model mean average precision increases 31.76% to achieve 74.13%. Thus, when appropriate correction methods are applied, the FPGA implementation provides similar robustness to noise as well as inference performance to its conventional GPU counterpart while maintaining a much smaller SWaP profile. We consider this demonstration of the superiority of the FPGA-based solution over the GPU-based solution for Edge vision system deployment to be a major contribution to the thesis work.

Contribution 2 (minor)

The first minor contribution of this thesis was the development of an FPGA-based video processing pipeline (see red box in 1.2). We achieved this using various Xilinx design tools creating first a video processing architecture which could receive a HDMI input source, process the signal, and then output this new signal to a HDMI output sink. In addition to this base design was the development of a library of video processing IP cores that could be placed within the FPGA data stream to alter the video signal as desired. This base FPGA system and library of video processing cores were implemented in several Xilinx FPGA boards whose performance achieved 30 frames-per-second at a resolution of 1920x1080. This base FPGA video processing system was designed to be extendable for the experiments in our other contributions. We consider this FPGA base system to be a minor contribution to the thesis work.

Contribution 3 (minor)

In our next minor contribution we established that FPGA-based models (ResNet-50 and YOLO version 3) both incur only an approximate 1% drop in performance as compared to GPU-based models, providing similar performance across both image classification and object detection problems, while benefiting from the significant SWaP characteristics of the FPGA. After developing the base FPGA video processing system we extended this architecture to include deep learning computer vision IP cores in the hardware processing pipeline (see green box in 1.2). The CNN hardware IP cores themselves were created leveraging Xilinx's DNNDK tool. This tool takes a pre-trained model from several popular

deep learning frameworks and quantizes the model weights from 32-bit floating point to 8-bit integer representation for FPGA implementation. For our side-by-side analysis of FPGA versus GPU performance we decided to target two computer vision tasks: (1) image classification via the ResNet-50 CNN architecture; and, (2) object detection via the YOLO version 3 CNN architecture. The ResNet-50 model was trained on the ImageNet-2012 training dataset (approximately 1.2 million images) and tested on a subset of the ImageNet-2012 testing dataset (5,000 images) in which the images are assigned to 1,000 classes. The YOLO version 3 model was trained on the VOC2007 training dataset consisting of 9,963 images with 24,640 annotated objects and tested on the VOC2007 testing dataset (4,952 images) in which images can fall under 20 classes. Our FPGA models were evaluated using a Xilinx ZCU104 UltraScale+ evaluation board and the original floating point models on a Nvidia RTX 2070 Super. Model performance was measured via accuracy for the image classification task with ResNet-50 and the FPGA model was shown to have an 1.1% drop in the top-1 accuracy and a 0.54% drop in top-5 accuracy as compared to the GPU solution. The FPGA had a slight boost in frames-per-second performance, 144fps for FPGA versus 140fps for GPU. Although the FPGA solution is approximately half the size and weight of the GPU solution, of particular note is the performance-to-power metrics. When consuming maximum system power, the FPGA-based system achieves 2.4 fps/W whereas the GPU based-solution achieves 0.65 fps/W. Performance for the object detection task with the YOLO version 3 model was measured with mean average precision and the FPGA had a 1.02% drop in mAP as compared to the GPU solution. In contrast to the ResNet-50 model, the GPU model had a slight boost in frames-per-second performance, 29fps for the GPU versus 27fps for the FPGA. When consuming maximum system power, the FPGA-based system achieves 0.45 fps/W whereas the GPU based-solution achieves 0.13 fps/W. We consider this side-by-side benchmarking of the two CNNs on our FPGA system against a conventional GPU system which establishes that FPGA models can achieve near identical performance to conventional GPU models to be a minor contribution to the thesis work.

Contribution 4 (minor)

In this final minor contribution we show a baseline statistical analysis of the impact of two degradation methods, along with corresponding correction methods, on the ImageNet-2012 and VOC2007 datasets via a quantitative image quality metric which illustrates that neither contain an inherent bias towards our evaluated methods. After establishing that the FPGA system provides near identical inference performance compared to the GPU counterpart we next wanted to explore its robustness to corrupted signals. Signals in unconstrained scenarios where Edge systems are deployed are often degraded due to environmental conditions, electro-mechanical interference, or other factors. Therefore, determining if the FPGA model would provide similar robustness to noise allows for one measure of explainability when an incorrect inference is made. Our experiments evaluated the impact of two degradation methods (contrast reduced images and salt-and-pepper noise) as well as their correction methods (gamma correction and median filter, respectively). Before determining the impact on inference with our FPGA and GPU models, we designed experiments to establish whether the datasets used in our evaluations were biased to any of our selected degradation methods. This contribution is shown as an extension to our pipeline in the blue box in 1.2. The

structural similarity (SSIM) score, an image quality analysis technique, was used to provide a quantitative analysis of the impact of the degradations and corrections explored in this thesis. As predicted, these experiments shown that we could mimic real-world noises and correct the degradation, as well as provide a quantitative score to each image to provide statistical significance in our final experiments. Additionally, we demonstrated that neither evaluated dataset is biased towards any selected degradation or correction method. We consider this baseline statistical analysis of our selected processing functions to our selected testing datasets to be a minor contribution to the thesis work.

7.3 Future Work

During the course of completing the research for this thesis several ideas that could be used as future improvements to our work were touched upon. In this final section we will present these ideas in the hopes that others may explore them as future contributions to the body of research within this domain. In particular we recommend higher video resolutions in video processing, additional benchmarking with more CNN architectures and domains, the use of more compelling datasets, a multi-FPGA fusion system, as well as improvements to the degradation/correction pipeline to better target real-time deployment.

The targeted resolution for video processing in our FPGA system was 1920x1080. This resolution was selected based upon its frequent use in off-the-shelf general purpose cameras, security cameras and lower-end consumer drones. Given the current trend of 4K-resolution as an option for consumer imaging systems, we recommend expanding the FPGA system to handle such a resolution as well as re-conducting the associated baseline benchmarking between the GPU and FPGA when handling such video resolutions. Additionally, such experiments would provide insight into the selection of an FPGA solution when a trade-off between SWaP characteristics and processing capability needs to be determined.

Although we attempted to provide a statistically significant analysis by considering two different CNN architectures targeting two different problem spaces, we would recommend that future work explore a wider range of architectures as well as other domain problems such as facial recognition. Such experiments could be arranged to achieve various goals: 1) provide a breadth comparison of inference speed versus robustness to noise across several architectures targeting the same problem domain, and 2) comparing deployment techniques such as pruning with GPU and FPGA implementations to test the impact on both inference speed and inference accuracy. Furthermore, our experiments could be extended by simply incorporating a wider range of more challenging datasets to investigate FPGA CNN implementation performance.

Related to the above recommendation, another potential extension would be to use multiple FPGAs running different DPU CNNs to allow for the fusion of results leading to even higher performance in Edge system inference. Such a system design could be designed with two different strategies. Firstly, as a typical deep learning fusion system where each FPGA implements a different architecture where the results from each is leveraged by a fusion method (majority vote, weighted average, etc.) to provide more confidence in the inference prediction. Secondly, a multi-modal system could be used where each FPGA leverages an architecture targeting imagery from a particular wavelength of the electromagnetic spectrum

(visible, swir, lwir, etc.) and fuses the results to provide a higher contextual understanding of a scene.

The final recommendation we offer is the use of a referenceless (aka blind) image quality metric such as the BRISQUE algorithm, so that a correction method could be invoked "on-the-fly" by FPGA partial reconfiguration during deployment allowing for a more responsive and real-time focused system. Methods such as BRISQUE provide a quality score based on information solely from the image itself providing the potential to evaluate an input frame from the video and provide direction on which correction method to use along with the degree of correction required. Such a system would enhance the performance of the entire processing pipeline and provide additional utility for Edge systems such as autonomous vehicles where quickly changing imaging scenarios are common place.

Bibliography

[1] L. K. Phimu and M. Kumar, "Design and implementation of area efficient 2-parallel filter on fpga using image system," in *2017 International Conference on Innovative Research In Electrical Sciences (IICIRES)*. IEEE, 2017, pp. 1–6. xvi, 19, 21

[2] L. H. Crockett, R. A. Elliot, M. A. Enderwitz, and R. W. Stewart, *The Zynq Book: Embedded Processing with the ARM Cortex-A9 on the Xilinx Zynq-7000 All Programmable SoC*. Strathclyde Academic Media, 2014. xvi, 19, 20, 21, 22

[3] Digilent, "Zybo zynq-7000 arm/fpga soc trainer board," July 2019. [Online]. Available: https://store.digilentinc.com/zybo-zynq-7000-arm-fpga-soc-trainer-board/ xvi, 20, 23

[4] ——, "Pynq-z1: Python productivity for zynq-7000 arm/fpga soc," July 2019. [Online]. Available: https://store.digilentinc.com/pynq-z1-python-productivity-for-zynq-7000-arm-fpga-soc/ xvi, 20, 23

[5] Xilinx, "Xilinx zynq ultrascale+ mpsoc zcu104 evaluation kit," July 2019. [Online]. Available: https://www.xilinx.com/products/boards-and-kits/zcu104.html xvi, 20, 23

[6] ——, "Axi-4 stream video ip and system design guide (ug934 version october 30, 2019)," October 2019. [Online]. Available: https://www.xilinx.com/support/documentation/ip_documentation/axi_videoip/v1_0/ug934_axi_videoIP.pdf xvii, 25, 26

[7] e. a. Alom, M.Z., "A state-of-the-art survey on deep learning theory and architectures," *Electronics*, vol. 8, no. 3, 2019. xvii, 27, 28

[8] Z. Zou, Z. Shi, Y. Guo, and J. Ye, "Object detection in 20 years: A survey," 2019. xvii, 27, 28

[9] S. Saha, "A comprehensive guide to convolutional neural networks - the eli5 way," Dec 2018. [Online]. Available: https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53 xvii, 29, 36

[10] R. E. Gonzalez, Rafel C. Woods, *Digital Image Processing, Third Edition*. Pearson Education, Inc., 2008. xviii, xxxvii, xxxviii, xxxix, xl, 46, 47, 48, 177, 179, 183, 184, 187, 189, 190, 193, 195, 199

[11] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778. xviii, 52, 53, 54

[12] Xilinx, "Dnndk user guide - ug1327 v1.5," June 2019. [Online]. Available: https://www.xilinx.com/support/documentation/user_guides/ug1327-dnndk-user-guide.pdf xviii, 32, 53, 56

[13] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 779–788. xix, 58, 60

[14] S. Ong, "At singapores test center, self-driving cars battle fake monsoons." https://spectrum.ieee.org/cars-that-think/transportation/self-driving/at-singapores-autonomous-vehicle-test-center-selfdriving-cars-battle-fake-monsoons, accessed: 2021-04-05. 2

[15] "Security cameras." https://www.secureitsecurities.com/security-cameras, accessed: 2021-04-05. 2

[16] G. S., "Winter drone flying tips: How to prepare for droning in cold weather." https://digitaltechreviewsandtips.com/winter-drone-flying-tips-how-to-prepare-for-droning-in-cold-weather/, accessed: 2021-04-05. 2

[17] S. A. Clukey, *Architecture for Real-Time, Low-SWaP Embedded Vision Using FPGAs.* Master's Thesis, University of Tennessee, 2016, available at https://trace.tennessee.edu/utk_gradthes/4281. 2, 16, 24, 37, 41

[18] Xilinx, "Vivado design suite user guide-high-level synthesis," December 2018. [Online]. Available: https://www.xilinx.com/support/documentation/sw_manuals/xilinx2018_3/ug902-vivado-high-level-synthesis.pdf 2, 24

[19] J. Bachrach, H. Vo, B. Richards, Y. Lee, A. Waterman, R. Avižienis, J. Wawrzynek, and K. Asanović, "Chisel: constructing hardware in a scala embedded language," in *DAC Design Automation Conference 2012*. IEEE, 2012, pp. 1212–1221. 2, 24

[20] M. Seals, *On the Robustness of Object Detection Based Deep Learning Models.* Master's Thesis, University of Tennessee, 2019, available at https://trace.tennessee.edu/utk_gradthes/5487. 4, 33

[21] A. Krizhevsky, I. Sutskever, and G. Hinton, "Imagenet classification with deep convolutional neural networks," *Neural Information Processing Systems*, vol. 25, 01 2012. 13, 27

[22] R. Raina, A. Madhavan, and A. Y. Ng, "Large-scale deep unsupervised learning using graphics processors," in *Proceedings of the 26th Annual International Conference on Machine Learning*, ser. ICML '09. New York, NY, USA: ACM, 2009, pp. 873–880. [Online]. Available: http://doi.acm.org/10.1145/1553374.1553486 14, 30

[23] X. Wang, Y. Han, V. C. M. Leung, D. Niyato, X. Yan, and X. Chen, "Convergence of edge computing and deep learning: A comprehensive survey," *IEEE Communications Surveys Tutorials*, vol. 22, no. 2, pp. 869–904, 2020. 14

[24] Z. Zhou, X. Chen, E. Li, L. Zeng, K. Luo, and J. Zhang, "Edge intelligence: Paving the last mile of artificial intelligence with edge computing," *Proceedings of the IEEE*, vol. 107, no. 8, pp. 1738–1762, 2019. 14

[25] H.-S. Kim, "Fog computing and the internet of things: Extend the cloud to where the things are," *International Journal of Cisco*, 2016. 14

[26] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, "Edge computing: Vision and challenges," *IEEE internet of things journal*, vol. 3, no. 5, pp. 637–646, 2016. 14

[27] C. Campolo, G. Genovese, A. Iera, and A. Molinaro, "Virtualizing ai at the distributed edge towards intelligent iot applications," *Journal of Sensor and Actuator Networks*, vol. 10, no. 1, p. 13, 2021. 14

[28] T. Zhang, Y. Li, and C. P. Chen, "Edge computing and its role in industrial internet: Methodologies, applications, and future directions," *Information Sciences*, vol. 557, pp. 34–65, 2021. 14

[29] R. Nane, V.-M. Sima, C. Pilato, J. Choi, B. Fort, A. Canis, Y. T. Chen, H. Hsiao, S. Brown, F. Ferrandi *et al.*, "A survey and evaluation of fpga high-level synthesis tools," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 35, no. 10, pp. 1591–1604, 2016. 24

[30] M. Qasaimeh, K. Denolf, J. Lo, K. Vissers, J. Zambreno, and P. H. Jones, "Comparing energy efficiency of cpu, gpu and fpga implementations for vision kernels," in *2019 IEEE International Conference on Embedded Software and Systems (ICESS)*. IEEE, 2019, pp. 1–8. 25

[31] Xilinx, "Hdmi 1.4/2.0 transmitter subsystem v3.1 product guide (pg235 v3.1)," December 2020. [Online]. Available: https://www.xilinx.com/support/documentation/ip_documentation/v_hdmi_tx_ss/v3_1/pg235-v-hdmi-tx-ss.pdf 25

[32] ——, "Implementing memory structures for video processing in the vivado hls tool (xapp793 v1.0)," September 2012. [Online]. Available: https://www.xilinx.com/support/documentation/application_notes/xapp793-memory-structures-video-vivado-hls.pdf 25

[33] "Pynq: Python productivity," http://http://www.pynq.io/, accessed: 2021-03-06. 25, 38

[34] S. S. Jadhav, N. LaMoyne, A. Chen, C. Gloster, D. Yang, S. Yun, and Y. Kim, "Building an acceleration overlay for novice students," in *2018 31st IEEE International System-on-Chip Conference (SOCC)*. IEEE, 2018, pp. 136–139. 25

[35] B. Hutchings and M. Wirthlin, "Rapid implementation of a partially reconfigurable video system with pynq," in *2017 27th International Conference on Field Programmable Logic and Applications (FPL)*. IEEE, 2017, pp. 1–8. 25

[36] F. Rosenblatt, "The perceptron: A probabilistic model for information storage and organization in the brain." *Psychological Review*, vol. 65, no. 6, 1958. 27

[37] R. Yamashita, M. Nishio, R. K. G. Do, and K. Togashi, "Convolutional neural networks: an overview and application in radiology," *Insights into imaging*, vol. 9, no. 4, pp. 611–629, 2018. 29

[38] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE conference on computer vision and pattern recognition*. Ieee, 2009, pp. 248–255. 30

[39] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman, "The PASCAL Visual Object Classes Challenge 2007 (VOC2007) Results," http://www.pascal-network.org/challenges/VOC/voc2007/workshop/index.html. 30

[40] A. Shawahna, S. M. Sait, and A. El-Maleh, "Fpga-based accelerators of deep learning networks for learning and classification: A review," *IEEE Access*, vol. 7, pp. 7823–7859, 2019. 31

[41] J. Chen and X. Ran, "Deep learning with edge computing: A review," *Proceedings of the IEEE*, vol. 107, no. 8, pp. 1655–1674, 2019. 31

[42] K. Abdelouahab, M. Pelcat, J. Serot, and F. Berry, "Accelerating cnn inference on fpgas: A survey," *arXiv preprint arXiv:1806.01683*, 2018. 31

[43] Y. Ma, Y. Cao, S. Vrudhula, and J.-s. Seo, "Performance modeling for cnn inference accelerators on fpga," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2019. 31

[44] J. Cong, Z. Fang, M. Lo, H. Wang, J. Xu, and S. Zhang, "Understanding performance differences of fpgas and gpus," in *2018 IEEE 26th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*. IEEE, 2018, pp. 93–96. 31

[45] D. L. N. Hettiarachchi, V. S. P. Davuluru, and E. J. Balster, "Integer vs. floating-point processing on modern fpga technology," in *2020 10th Annual Computing and Communication Workshop and Conference (CCWC)*, 2020, pp. 0606–0612. 31

[46] R. Zhao, W. Song, W. Zhang, T. Xing, J.-H. Lin, M. Srivastava, R. Gupta, and Z. Zhang, "Accelerating binarized convolutional neural networks with software-programmable fpgas," in *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, 2017, pp. 15–24. 31

[47] Y. Umuroglu, N. J. Fraser, G. Gambardella, M. Blott, P. Leong, M. Jahre, and K. Vissers, "Finn: A framework for fast, scalable binarized neural network inference," in *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, 2017, pp. 65–74. 31

[48] A. Prost-Boucle, A. Bourge, F. Pétrot, H. Alemdar, N. Caldwell, and V. Leroy, "Scalable high-performance architecture for convolutional ternary neural networks on fpga," in *2017 27th International Conference on Field Programmable Logic and Applications (FPL)*. IEEE, 2017, pp. 1–7. 31

[49] H. Alemdar, V. Leroy, A. Prost-Boucle, and F. Pétrot, "Ternary neural networks for resource-efficient ai applications," in *2017 international joint conference on neural networks (IJCNN)*. IEEE, 2017, pp. 2547–2554. 31

[50] S. Seo and J. Kim, "Efficient weights quantization of convolutional neural networks using kernel density estimation based non-uniform quantizer," *Applied Sciences*, vol. 9, no. 12, p. 2559, 2019. 31

[51] S. I. Venieris and C.-S. Bouganis, "fpgaconvnet: A framework for mapping convolutional neural networks on fpgas," in *2016 IEEE 24th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*. IEEE, 2016, pp. 40–47. 31

[52] Y. Guan, H. Liang, N. Xu, W. Wang, S. Shi, X. Chen, G. Sun, W. Zhang, and J. Cong, "Fp-dnn: An automated framework for mapping deep neural networks onto fpgas with rtl-hls hybrid templates," in *2017 IEEE 25th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*. IEEE, 2017, pp. 152–159. 31

[53] S. I. Venieris, A. Kouris, and C.-S. Bouganis, "Toolflows for mapping convolutional neural networks on fpgas: A survey and future directions," *arXiv preprint arXiv:1803.05900*, 2018. 32

[54] Y. Wang, J. Xu, Y. Han, H. Li, and X. Li, "Deepburning: Automatic generation of fpga-based learning accelerators for the neural network family," in *2016 53nd ACM/EDAC/IEEE Design Automation Conference (DAC)*. IEEE, 2016, pp. 1–6. 32

[55] C. Zhang, G. Sun, Z. Fang, P. Zhou, P. Pan, and J. Cong, "Caffeine: Toward uniformed representation and acceleration for deep convolutional neural networks," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 38, no. 11, pp. 2072–2085, 2018. 32

[56] V. Gokhale, A. Zaidy, A. X. M. Chang, and E. Culurciello, "Snowflake: An efficient hardware accelerator for convolutional neural networks," in *2017 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, 2017, pp. 1–4. 32

[57] V. Y. Fu, W. Ephrem, and V. Kathail, "Embedded vision with int8 optimization on xilinx devices," *Tech. Rep.*, 2017. 32

[58] Xilinx, "Xilinx ai model zoo," March 2020. [Online]. Available: https://github.com/Xilinx/AI-Model-Zoo 32, 55, 61

[59] S. Dodge and L. Karam, "Understanding how image quality affects deep neural networks," in *2016 eighth international conference on quality of multimedia experience (QoMEX)*. IEEE, 2016, pp. 1–6. 33

[60] S. Karahan, M. K. Yildirum, K. Kirtac, F. S. Rende, G. Butun, and H. K. Ekenel, "How image degradations affect deep cnn-based face recognition?" in *2016 International Conference of the Biometrics Special Interest Group (BIOSIG)*. IEEE, 2016, pp. 1–5. 33

[61] S. Dodge and L. Karam, "A study and comparison of human and deep learning recognition performance under visual distortions," in *2017 26th international conference on computer communication and networks (ICCCN)*. IEEE, 2017, pp. 1–7. 33

[62] S. Diamond, V. Sitzmann, S. Boyd, G. Wetzstein, and F. Heide, "Dirty pixels: Optimizing image classification architectures for raw sensor data," *arXiv preprint arXiv:1701.06487*, 2017. 33

[63] I. Vasiljevic, A. Chakrabarti, and G. Shakhnarovich, "Examining the impact of blur on recognition by convolutional networks," *arXiv preprint arXiv:1611.05760*, 2017. 33

[64] Y. Zhou, S. Song, and N.-M. Cheung, "On classification of distorted images with deep convolutional neural networks," in *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2017, pp. 1213–1217. 33

[65] D. Hendrycks and T. Dietterich, "Benchmarking neural network robustness to common corruptions and perturbations," *arXiv preprint arXiv:1903.12261*, 2019. 33

[66] J. Yim and K.-A. Sohn, "Enhancing the performance of convolutional neural networks on quality degraded datasets," in *2017 International Conference on Digital Image Computing: Techniques and Applications (DICTA)*. IEEE, 2017, pp. 1–8. 33

[67] E. Rusak, L. Schott, R. Zimmermann, J. Bitterwolf, O. Bringmann, M. Bethge, and W. Brendel, "Increasing the robustness of dnns against image corruptions by playing the game of noise," *arXiv preprint arXiv:2001.06057*, 2020. 33

[68] Z. Wang, E. P. Simoncelli, and A. C. Bovik, "Multiscale structural similarity for image quality assessment," in *The Thrity-Seventh Asilomar Conference on Signals, Systems & Computers, 2003*, vol. 2. IEEE, 2003, pp. 1398–1402. 34

[69] "revision getting started guide 2018.3 (ug1265)," https://github.com/Xilinx/Embedded-Reference-Platforms-User-Guide/tree/2018.3, accessed: 2021-03-06. 41

[70] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014. 52

[71] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2014, pp. 580–587. 58

[72] R. Girshick, "Fast r-cnn," in *Proceedings of the IEEE international conference on computer vision*, 2015, pp. 1440–1448. 58

[73] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," in *Advances in neural information processing systems*, 2015, pp. 91–99. 58

[74] J. Redmon and A. Farhadi, "Yolo9000: better, faster, stronger," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 7263–7271. 59

[75] ——, "Yolov3: An incremental improvement," *arXiv preprint arXiv:1804.02767*, 2018. 59

Appendix

A Library of Custom FPGA IP Cores

Sobel Edge Detection IP Core

The Sobel operator is a means of calculating the derivative of an image for sharpening/edge detection [10]. For the purposes of this implementation, two 3x3 Sobel kernels were used to calculate the horizontal and vertical derivatives of the input frame. The left kernel in figure 1 calculates the horizontal derivative and the right kernel calculates the vertical derivative. All of the weights add up to 0, which would produce a response of 0 in an area of constant intensity. The weights of 2/-2 add some smoothing to the sharpening process. For this implementation of the Sobel derivative, the horizontal and vertical derivatives are added together (with equal weighting) to produce a combined derivative.

In both the IP core code and the C++ equivalent code, each kernel is convolved with the original image and the results are added together to create an output image. The flowchart depicting this process for the IP core can been seen in figure 2.

After completing the design flow within Vivado HLS a Sobel IP core was created that could be implemented within a video pipeline architecture. The top-level block diagram of this IP core appears in figure 3. The Vivado HLS process also provides diagnostic and implementation summaries to catalog the particular implementation of a HLS created IP core. Of particular interest are the timing/latency summaries in figure 4 and the hardware utilization summary in figure 5. A comparison of hardware utilization across the different boards appears in table 1. This custom IP core was validated using a grayscale image and a color image. These results are displayed in the figures 6 and 7.

Thresholding IP Core

The Threshold function is a means of re-mapping the intensities of the original pixels based upon the value of a supplied threshold value. If the original pixel intensity is below this threshold value then it is assigned an intensity value of 0. Likewise, if the original pixel intensity is greater than or equal to the threshold value then it is assigned an intensity value of 255 (for the case of an 8-bit depth). Therefore, the result of a Threshold operation will be a binary version of the original image.

In both the IP core code and the C++ equivalent code, each pixel of an input frame is first converted to grayscale as a pre-processing step. Each pixel is then mapped to either 0 or 255 based on whether it is less or greater than or equal to a specified threshold value. This value is then assigned to each of the original R,G and B channels maintaining the grayscale value while allowing for display to a monitor. The flowchart depicting this process for the IP core can been seen below in figure 8.

After completing the design flow within Vivado HLS a Threshold IP core was created that could be implemented within a video pipeline architecture. The top-level block diagram of this IP core appears in figure 9. The Vivado HLS process also provides diagnostic and implementation summaries to catalog the particular implementation of a HLS created IP core. Of particular interest are the timing/latency summaries in figure 10 and the hardware utilization summary in figure 11. A comparison of hardware utilization across the different boards appears in table 2. This custom IP core was validated using a color image. The results are displayed in the figure 12.

-1	-2	-1
0	0	0
1	2	1

-1	0	1
-2	0	2
-1	0	1

Figure 1: Horizontal and vertical Sobel kernels.

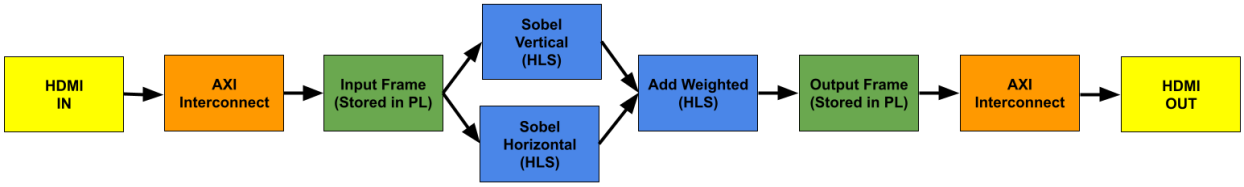


Figure 2: Sobel operation IP core flow chart. *The HDMI source signal is captured via the AXI interconnect and stored as an input frame within the Zynq PL. HLS code is then used to calculate the horizontal and vertical Sobel filters, combine them and cast the result into an output frame before sending it out of the PL via AXI interconnect and then to HDMI sink.*

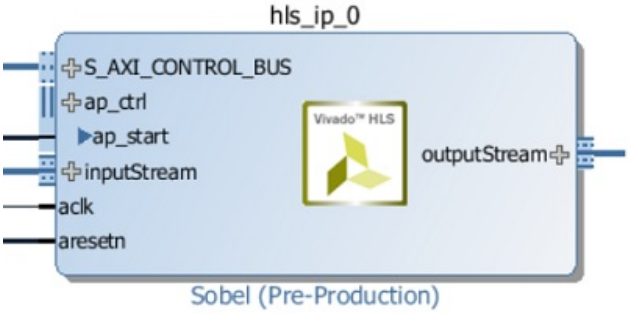


Figure 3: Sobel IP core block diagram.

Timing (ns)

Summary

Clock	Target	Estimated	Uncertainty
default	10.00	8.02	1.25

Latency (clock cycles)

Summary

Latency		Interval		
min	max	min	max	Type
155	2095147	155	2095147	dataflow

Figure 4: Sobel IP timing/latency.

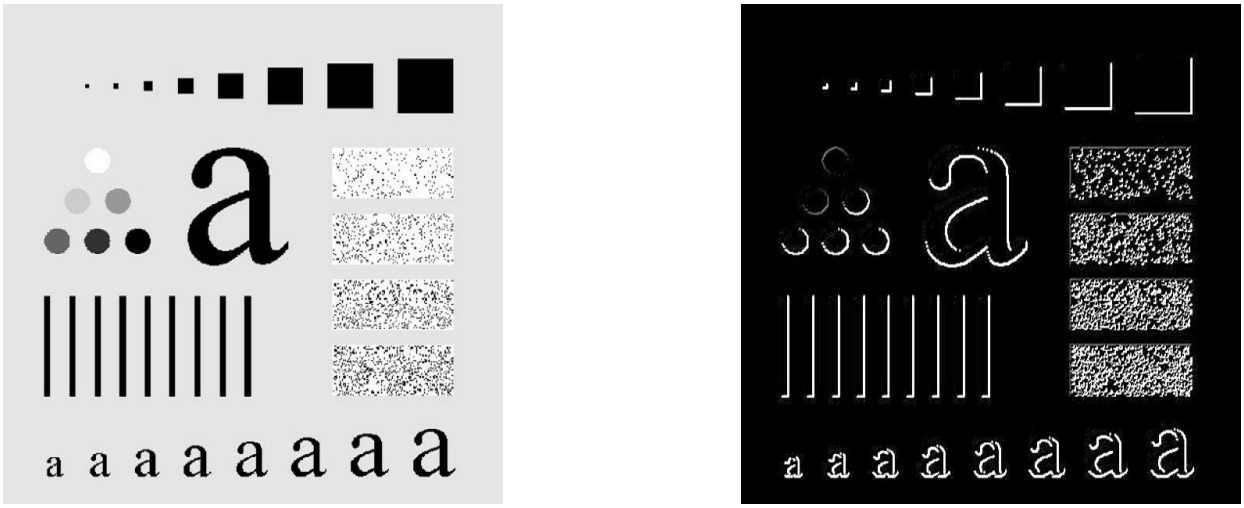
Summary

Name	BRAM_18K	DSP48E	FF	LUT
Expression	-	-	0	6
FIFO	0	-	190	840
Instance	18	36	3767	6314
Memory	-	-	-	-
Multiplexer	-	-	-	40
Register	-	-	28	-
Total	18	36	3985	7200

Figure 5: Sobel IP hardware utilization.

Table 1: Comparison of FPGA board resources utilization for Sobel IP core.

BOARD	LUTs	Flip-Flops	BRAM
Zybo	15.09%	20.45%	7.50%
Z1	7.49%	6.77%	2.86%
ZCU104 Ultrascale+	0.79%	0.18%	0.05%



(a) *Original grayscale image.* **(b)** *Sobel processed.*

Figure 6: Sobel Edge Detector - grayscale image results (image source [10]).



(a) *Original color image.* **(b)** *Sobel processed.*

Figure 7: Sobel Edge Detector - color image results (image source [10]).

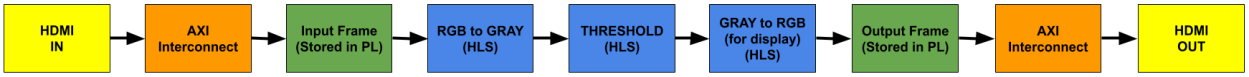


Figure 8: Threshold operation IP core flow chart. *The HDMI source signal is captured via the AXI interconnect and stored as an input frame within the Zynq PL. HLS code is then used to convert the R,G and B channels into grayscale equivalent. Each pixel is then evaluated, if the value is below 127 it is now mapped to 0 and if it is greater than or equal to 127 then it is mapped to 255. This value is then duplicated back to a 3-channel grayscale image and placed into the output frame before sending it out of the PL via AXI interconnect and then to HDMI sink.*

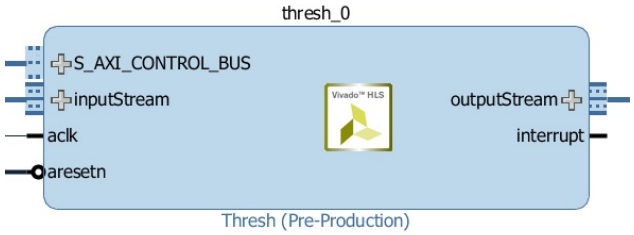


Figure 9: Threshold IP core block diagram.

Timing (ns)

Summary

Clock	Target	Estimated	Uncertainty
ap_clk	10.00	9.40	1.25

Latency (clock cycles)

Summary

Latency		Interval		
min	max	min	max	Type
11	2081163	4	2081164	dataflow

Figure 10: Threshold IP timing/latency.

Utilization Estimates

Summary

Name	BRAM_18K	DSP48E	FF	LUT
DSP	-	-	-	-
Expression	-	-	0	2
FIFO	0	-	40	160
Instance	-	3	575	531
Memory	-	-	-	-
Multiplexer	-	-	-	-
Register	-	-	-	-
Total	0	3	615	693

Figure 11: Threshold IP hardware utilization.

Table 2: Comparison of FPGA board resources utilization for Threshold IP core.

BOARD	LUTs	Flip-Flops	BRAM
Zybo	2.63%	1.75%	0.00%
Z1	1.30%	0.58%	0.00%
ZCU104 Ultrascale+	0.14%	0.02%	0.00%

Sobel then Thresholding IP Core

The Sobel-Threshold function is a compounded function that first calculates the Sobel derivative of the input and then calculates the Threshold of this intermediate value to produce the desired output. The same functions are implemented that were used in the separate Sobel and Threshold cores with some additional supportive features dealing with image type conversions.

In both the IP core code and the C++ equivalent code, the HDMI source signal is captured via the AXI interconnect and stored as an input frame f(x,y). HLS code is then used to calculate the Sobel derivative of the frame. Then, each pixel of the Sobel frame is first converted to grayscale as a pre-processing step. Each pixel is then mapped to either 0 or 255 based on whether it is less or greater than or equal to a specified threshold value. This value is then assigned to each of the original R,G and B channels maintaining the grayscale value while allowing for display to a monitor. The flowchart depicting this process for the IP core can been seen in figure 13.

After completing the design flow within Vivado HLS a Sobel-Threshold IP core was created that could be implemented within a video pipeline architecture. The top-level block diagram of this IP core appears in figure 14. The Vivado HLS process also provides diagnostic and implementation summaries to catalog the particular implementation of a HLS created IP core. Of particular interest are the timing/latency summaries in figure 15 and the hardware utilization summary in figure 16. A comparison of hardware utilization across the different boards appears in table 3. This custom IP core was validated using a grayscale image and a color image. These results are displayed in the figures 17 and 18.

Dilation

Dilation is one of the two fundamental operations of morphological image processing. Morphology provides methods to analysis the shape of an object in an image - the boundaries, hull, etc. Dilation, as its name implies, increases the number of pixels from an object boundary giving the visual appearance of "widening" the object. The rate of this pixel increase is based on the size and type of the structuring element used in the process.

For this implementation of dilation, both the HLS IP core and the software equivalent implemented a 3x3 structuring element as seen in figure 19. The flowchart depicting this process for the IP core can been seen in figure 20.

After completing the design flow within Vivado HLS a Dilation IP core was created that could be implemented within a video pipeline architecture. The top-level block diagram of this IP core appears in figure 21. The Vivado HLS process also provides diagnostic and implementation summaries to catalog the particular implementation of a HLS created IP core. Of particular interest are the timing/latency summaries in figure 22 and the hardware utilization summary in figure 23. A comparison of hardware utilization across the different boards appears in table 4. This custom IP core was validated using a grayscale image and a color image. These results are displayed in the figures 24 and 25.



(a) *Original color image.* (b) *Threshold processed.*

Figure 12: Threshold (threshold parameter = 127) - color image results.



Figure 13: Sobel-Threshold operation IP core flow chart. *The HDMI source signal is captured via the AXI interconnect and stored as an input frame within the Zynq PL. HLS code is then used to calculate the Sobel derivative of the frame. Additional HLS code is then used to convert the R,G and B channels into grayscale equivalent. Each pixel is then evaluated, if the value is below 127 it is now mapped to 0 and if it is greater than or equal to 127 then it is mapped to 255. This value is then duplicated back to a 3-channel grayscale image and placed into the output frame before sending it out of the PL via AXI interconnect and then to HDMI sink.*

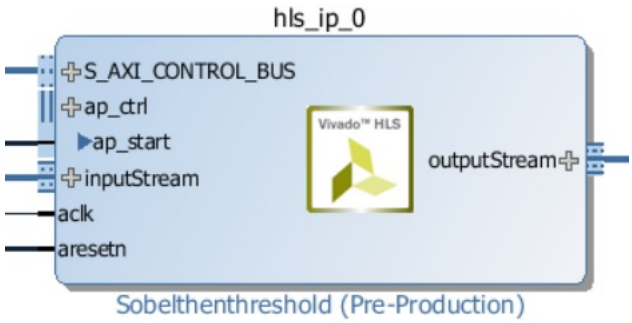


Figure 14: Sobel-Threshold IP core block diagram.

Timing (ns)

Summary

Clock	Target	Estimated	Uncertainty
default	10.00	8.08	1.25

Latency (clock cycles)

Summary

Latency		Interval		
min	max	min	max	Type
155	2095147	155	2095147	dataflow

Figure 15: Sobel-Threshold IP timing/latency.

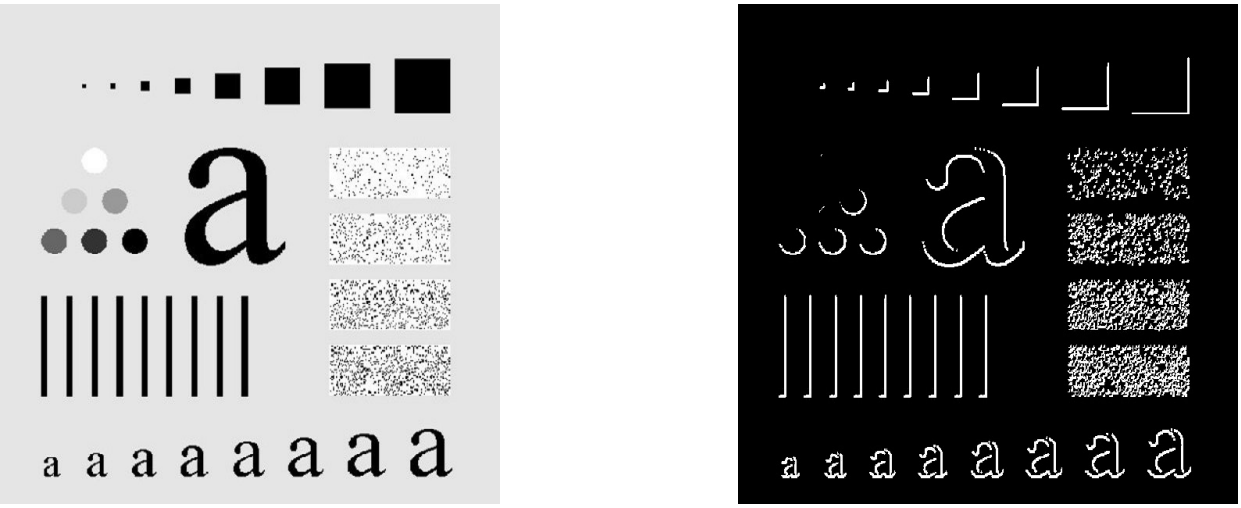
Summary

Name	BRAM_18K	DSP48E	FF	LUT
Expression	-	-	0	9
FIFO	0	-	255	1132
Instance	18	39	4085	6662
Memory	-	-	-	-
Multiplexer	-	-	-	56
Register	-	-	39	-
Total	18	39	4379	7859

Figure 16: Sobel-Threshold IP hardware utilization.

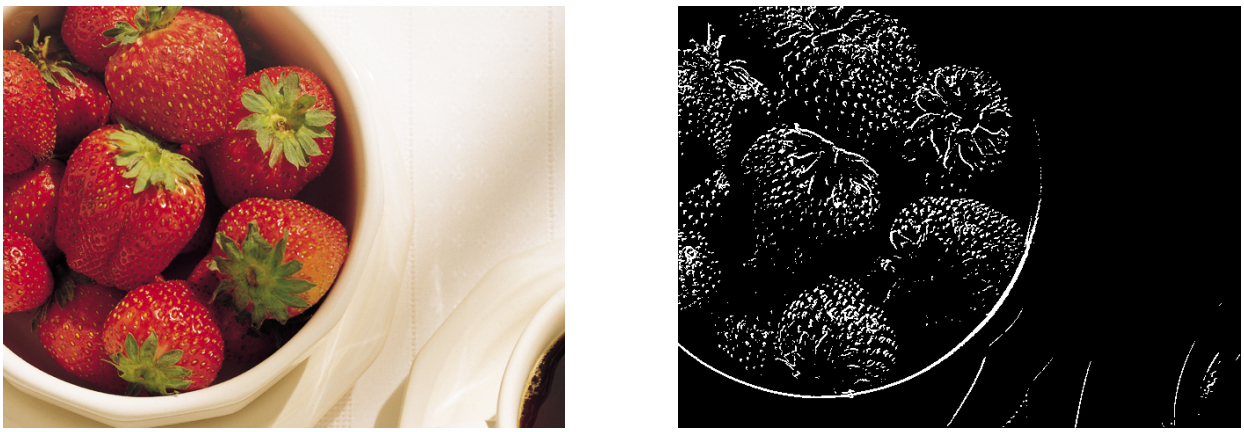
Table 3: Comparison of FPGA board resources utilization for Sobel-Threshold IP core.

BOARD	LUTs	Flip-Flops	BRAM
Zybo	29.77%	12.44%	7.50%
Z1	14.77%	4.12%	2.86%
ZCU104 Ultrascale+	1.56%	0.11%	0.05%



(a) *Original grayscale image.* **(b)** *Sobel-Threshold processed.*

Figure 17: Sobel-Threshold - grayscale image results (image source [10]).



(a) *Original color image.* (b) *Sobel-Threshold processed.*

Figure 18: Sobel-Threshold - color image results (image source [10]).

$$\begin{matrix} 0 & 1 & 0 \\ 1 & 1 & 1 \\ 0 & 1 & 0 \end{matrix}$$

Figure 19: Dilation structuring element.



Figure 20: Dilation operation IP core flow chart. *The HDMI source signal is captured via the AXI interconnect and stored as an input frame within the Zynq PL. HLS code is then used to apply the structuring element to the pixels in the frame performing the dilation morphological operation and placed into the output frame before sending it out of the PL via AXI interconnect and then to HDMI sink.*

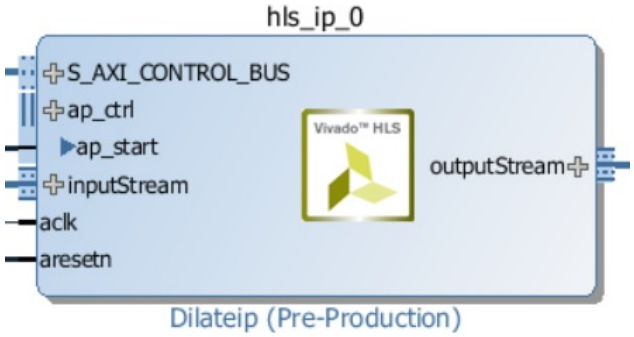


Figure 21: Dilation IP core block diagram.

Timing (ns)

Summary

Clock	Target	Estimated	Uncertainty
default	10.00	8.73	1.25

Latency (clock cycles)

Summary

Latency		Interval		
min	max	min	max	Type
141	2094061	141	2094061	dataflow

Figure 22: Dilation IP timing/latency.

Summary

Name	BRAM_18K	DSP48E	FF	LUT
Expression	-	-	0	3
FIFO	0	-	60	264
Instance	9	-	1300	1885
Memory	-	-	-	-
Multiplexer	-	-	-	12
Register	-	-	11	-
Total	9	0	1371	2164

Figure 23: Dilation IP hardware utilization.

Table 4: Comparison of FPGA board resources utilization for Dilation IP core.

BOARD	LUTs	Flip-Flops	BRAM
Zybo	N/A	N/A	N/A
Z1	4.07%	1.29%	1.43%
ZCU104 Ultrascale+	0.43%	0.03%	0.02%



(a) *Original grayscale image.*



(b) *Dilation processed.*

Figure 24: Dilation - grayscale image results.

Erosion

Erosion is the other fundamental operation of morphological image processing (along with dilation). Erosion, as its name implies, reduces the number of pixel from an object boundary giving the visual appearance of "thinning" or "shrinking" the object. As with dilation, the rate of this pixel reduction is based on the size and type of the structuring element used in the process.

For this implementation of erosion, both the HLS IP core and the software equivalent implemented a 3x3 structuring element as seen in figure 26. The flowchart depicting this process for the IP core can been seen in figure 27.

After completing the design flow within Vivado HLS an Erosion IP core was created that could be implemented within a video pipeline architecture. The top-level block diagram of this IP core appears in figure 28. The Vivado HLS process also provides diagnostic and implementation summaries to catalog the particular implementation of a HLS created IP core. Of particular interest are the timing/latency summaries in figure 29 and the hardware utilization summary in figure 30. A comparison of hardware utilization across the different boards appears in table 5. This custom IP core was validated using a grayscale image and a color image. These results are displayed in the figures 31 and 32.

Inversion

Image inversion performs bit-wise inversion of each pixel in an image. This operation can be used to make the content of an image clearer to a human observer. For a binary image the logical NOT operation would invert the pixel intensities. For 8-bit grayscale images inversion can be achieved by the function as seen in equation 1 below. The flowchart depicting this process for the IP core can been seen in figure 33.

$$Pixel_{inverted} = 255 - Pixel_{original} \tag{1}$$

After completing the design flow within Vivado HLS an Inversion IP core was created that could be implemented within a video pipeline architecture. The top-level block diagram of this IP core appears in figure 34. The Vivado HLS process also provides diagnostic and implementation summaries to catalog the particular implementation of a HLS created IP core. Of particular interest are the timing/latency summaries in figure 35 and the hardware utilization summary in figure 36. A comparison of hardware utilization across the different boards appears in table 6. This custom IP core was validated using a grayscale image and a color image. These results are displayed in the figures 37 and 38.

Gaussian Blur

The Gaussian Blur operation is a method that blurs an image by convolving it with a Gaussian function. The Gaussian Blur smooths the image by reducing the image's high-frequency components. This operation is essentially a low pass filter that reduces image noise and detail. For this implementation of the Gaussian Blur, a 5 x 5 kernel as seen in figure 39 representing the Gaussian function was applied to the image frame. The flowchart depicting this process for the IP core can been seen in figure 40.



(a) *Original color image.* **(b)** *Dilation processed.*

Figure 25: Dilation - color image results (image source [10]).

$$\begin{matrix} 0 & 1 & 0 \\ 1 & 1 & 1 \\ 0 & 1 & 0 \end{matrix}$$

Figure 26: Erosion structuring element.

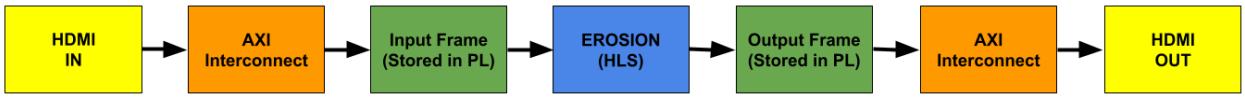


Figure 27: Erosion operation IP core flow chart. *The HDMI source signal is captured via the AXI interconnect and stored as an input frame within the Zynq PL. HLS code is then used to apply the structuring element to the pixels in the frame performing the erosion morphological operation and placed into the output frame before sending it out of the PL via AXI interconnect and then to HDMI sink.*

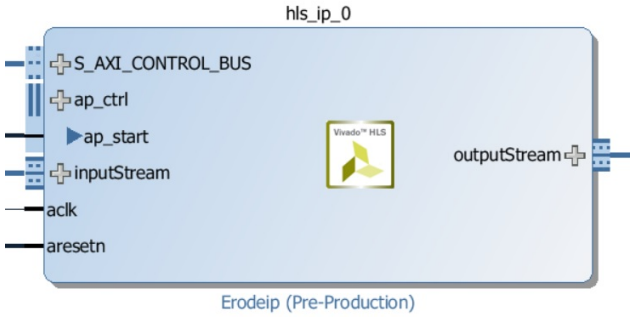


Figure 28: Erosion IP core block diagram.

Timing (ns)

Summary

Clock	Target	Estimated	Uncertainty
default	10.00	8.73	1.25

Latency (clock cycles)

Summary

Latency		Interval		
min	max	min	max	Type
141	2094061	141	2094061	dataflow

Figure 29: Erosion IP timing/latency.

Summary

Name	BRAM_18K	DSP48E	FF	LUT
Expression	-	-	0	3
FIFO	0	-	60	264
Instance	9	-	1300	1885
Memory	-	-	-	-
Multiplexer	-	-	-	12
Register	-	-	11	-
Total	9	0	1371	2164

Figure 30: Erosion IP hardware utilization.

Table 5: Comparison of FPGA board resources utilization for Erosion IP core.

BOARD	LUTs	Flip-Flops	BRAM
Zybo	N/A	N/A	N/A
Z1	4.07%	1.29%	1.43%
ZCU104 Ultrascale+	0.43%	0.03%	0.02%



(a) *Original grayscale image.*



(b) *Erosion processed.*

Figure 31: Erosion - grayscale image results.



(a) *Original color image.* (b) *Erosion processed.*

Figure 32: Erosion - color image results (image source [10]).



Figure 33: Inversion operation IP core flow chart. *The HDMI source signal is captured via the AXI interconnect and stored as an input frame within the Zynq PL. HLS code is then used to invert the pixel values in the frame and then placed into the output frame before sending it out of the PL via AXI interconnect and then to HDMI sink.*

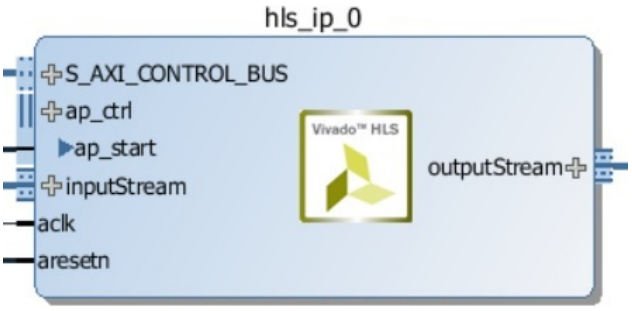


Figure 34: Inversion IP core block diagram.

Table 6: Comparison of FPGA board resources utilization for Inversion IP core.

BOARD	LUTs	Flip-Flops	BRAM
Zybo	3.54%	1.49%	0.00%
Z1	1.76%	0.49%	0.00%
ZCU104 Ultrascale+	0.19%	0.01%	0.00%

Timing (ns)

Summary

Clock	Target	Estimated	Uncertainty
default	10.00	4.77	1.25

Latency (clock cycles)

Summary

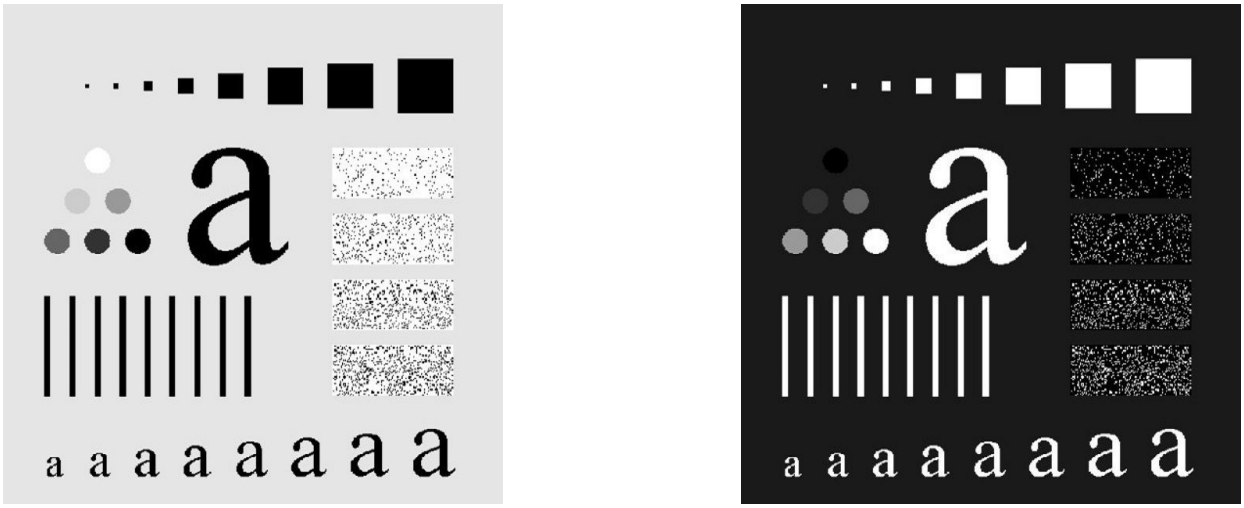
Latency		Interval		
min	max	min	max	Type
4	2077924	4	2077924	dataflow

Figure 35: Inversion IP timing/latency.

Summary

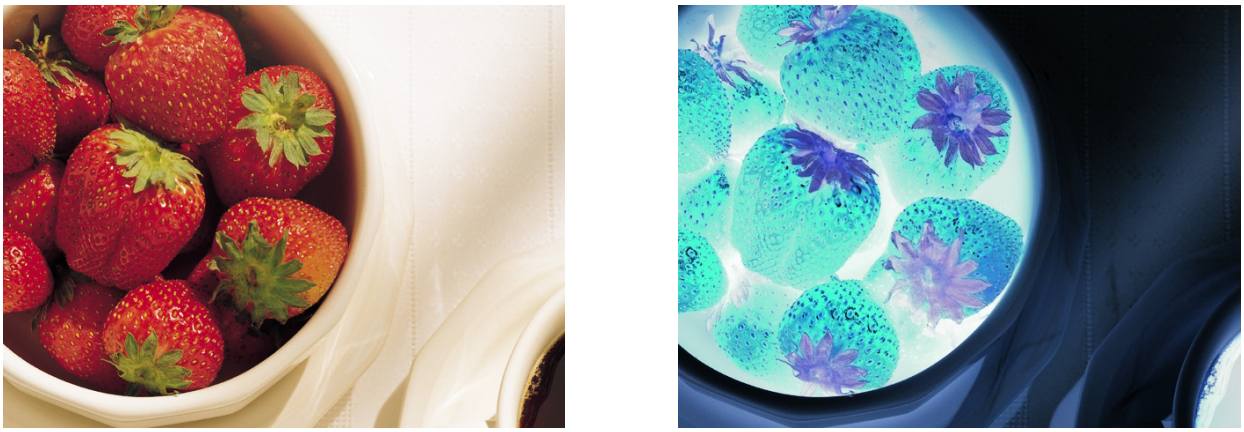
Name	BRAM_18K	DSP48E	FF	LUT
Expression	-	-	0	3
FIFO	0	-	70	312
Instance	-	-	442	603
Memory	-	-	-	-
Multiplexer	-	-	-	16
Register	-	-	13	-
Total	0	0	525	934

Figure 36: Inversion IP hardware utilization.



(a) *Original grayscale image.* (b) *Inversion processed.*

Figure 37: Inversion - grayscale image results (image source [10]).



(a) *Original color image.* (b) *Inversion processed.*

Figure 38: Inversion - color image results (image source [10]).

After completing the design flow within Vivado HLS a Gaussian Blur IP core was created that could be implemented within a video pipeline architecture. The top-level block diagram of this IP core appears in figure 41. The Vivado HLS process also provides diagnostic and implementation summaries to catalog the particular implementation of a HLS created IP core. Of particular interest are the timing/latency summaries in figure 42 and the hardware utilization summary in figure 43. A comparison of hardware utilization across the different boards appears in table 7. This custom IP core was validated using a grayscale image and a color image. These results are displayed in the figures 44 and 45.

Sharpen

The Sharpen operation is a method that sharpens the contrast of an image by convolving it with a kernel. For this implementation of the Sharpening operation, a 3 x 3 kernel as seen in figure 46 was applied to the image frame. The flowchart depicting this process for the IP core can been seen in figure 47.

After completing the design flow within Vivado HLS a Sharpen IP core was created that could be implemented within a video pipeline architecture. The top-level block diagram of this IP core appears in figure 48. The Vivado HLS process also provides diagnostic and implementation summaries to catalog the particular implementation of a HLS created IP core. Of particular interest are the timing/latency summaries in figure 49 and the hardware utilization summary in figure 50. A comparison of hardware utilization across the different boards appears in table 8. This custom IP core was validated using a grayscale image and a color image. These results are displayed in the figures 51 and 52.

Contrast Reduction (pixel operation)

Arithmetic pixel-level operations can also be realized via hardware processing. The operations of addition, subtraction, multiplication and division can be leveraged to augment an input frame to produce a desired result. One can use pixel-level manipulations to achieve numerous functions. For example, pixel-level division can be used to produce contrast reduced images. The IP core here uses a divide-by-3 function to achieve this reduced contrast exposure. The flowchart depicting this process for the IP core can been seen in figure 53.

After completing the design flow within Vivado HLS a contrast reduction IP core was created that could be implemented within a video pipeline architecture to produce contrast reduced images. The top-level block diagram of this IP core appears in figure 54. The Vivado HLS process also provides diagnostic and implementation summaries to catalog the particular implementation of a HLS created IP core. Of particular interest are the timing/latency summaries in figure 55 and the hardware utilization summary in figure 56. A comparison of hardware utilization across the different boards appears in table 9. This custom IP core was validated using a color image. These results are displayed in the figure 57.

Kernel Convolution

Image filtering via kernel convolution allows for several operations depending on the particular kernel used. Operations include smoothing, blurring, sharpening, etc. This versatility in operation is particularly desirable when the kernel can be hot-swapped to

$$\frac{1}{84}\begin{pmatrix} 1 & 2 & 3 & 2 & 1 \\ 2 & 5 & 6 & 5 & 2 \\ 3 & 6 & 8 & 6 & 3 \\ 2 & 5 & 6 & 5 & 2 \\ 1 & 2 & 3 & 2 & 1 \end{pmatrix}$$

Figure 39: Gaussian blur kernel.



Figure 40: Gaussian Blur operation IP core flow chart. *The HDMI source signal is captured via the AXI interconnect and stored as an input frame within the Zynq PL. HLS code is then used to blur the pixel values in the frame via the 5x5 Gaussian kernel and then placed into the output frame before sending it out of the PL via AXI interconnect and then to HDMI sink.*

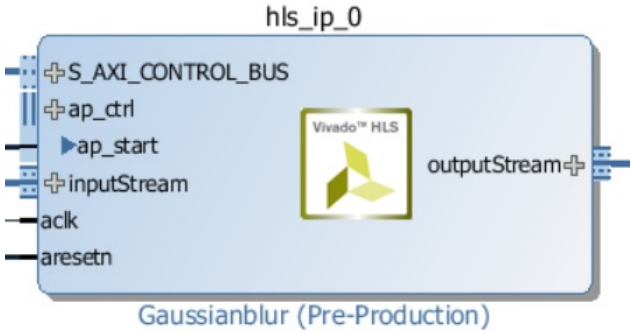


Figure 41: Gaussian Blur IP core block diagram.

Timing (ns)

Summary

Clock	Target	Estimated	Uncertainty
default	10.00	8.67	1.25

Latency (clock cycles)

Summary

Latency		Interval		
min	max	min	max	Type
267	2107227	267	2107227	dataflow

Figure 42: Gaussian Blur IP timing/latency.

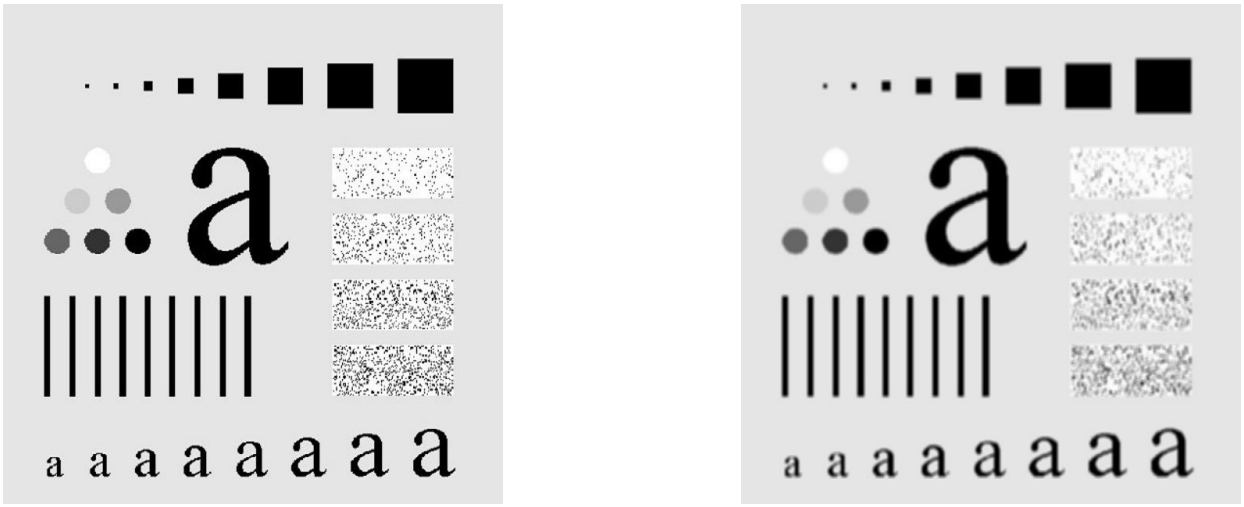
Summary

Name	BRAM_18K	DSP48E	FF	LUT
Expression	-	-	0	3
FIFO	0	-	60	264
Instance	15	72	4530	5822
Memory	-	-	-	-
Multiplexer	-	-	-	12
Register	-	-	11	-
Total	15	72	4601	6101

Figure 43: Gaussian Blur IP hardware utilization.

Table 7: Comparison of FPGA board resources utilization for Gaussian Blur IP core.

BOARD	LUTs	Flip-Flops	BRAM
Zybo	23.11%	13.07%	6.25%
Z1	11.47%	4.32%	2.38%
ZCU104 Ultrascale+	1.21%	0.11%	0.04%



(a) *Original grayscale image.* **(b)** *Gaussian Blur processed.*

Figure 44: Gaussian Blur - grayscale image results (image source [10]).



(a) *Original color image.* **(b)** *Gaussian Blur processed.*

Figure 45: Gaussian Blur - color image results (image source [10]).

$$\begin{pmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{pmatrix}$$

Figure 46: Sharpen kernel.



Figure 47: Sharpen operation IP core flow chart. *The HDMI source signal is captured via the AXI interconnect and stored as an input frame within the Zynq PL. HLS code is then used to sharpen the pixel values in the frame via a 3x3 kernel and then placed into the output frame before sending it out of the PL via AXI interconnect and then to HDMI sink.*

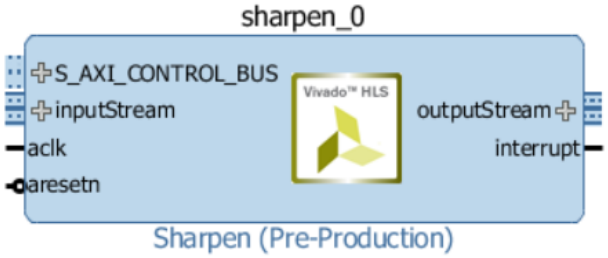


Figure 48: Sharpen IP core block diagram.

Timing (ns)

Summary

Clock	Target	Estimated	Uncertainty
ap_clk	10.00	10.25	1.25

Latency (clock cycles)

Summary

Latency		Interval		
min	max	min	max	Type
200	2087208	175	2087183	dataflow

Figure 49: Sharpen IP timing/latency.

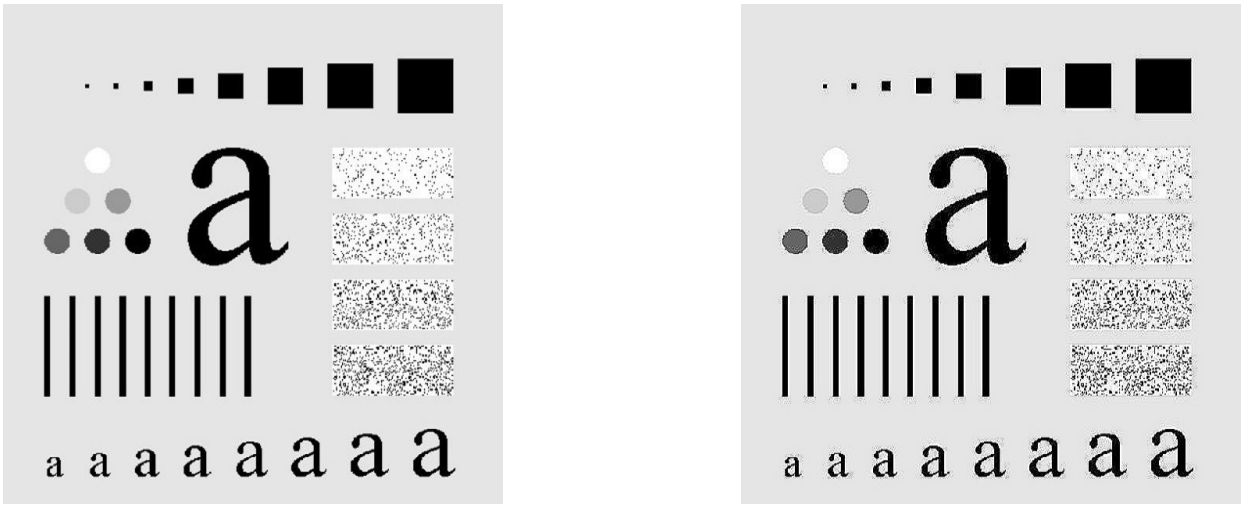
Summary

Name	BRAM_18K	DSP48E	FF	LUT
DSP	-	-	-	-
Expression	-	-	0	22
FIFO	0	-	75	300
Instance	9	27	1623	1295
Memory	-	-	-	-
Multiplexer	-	-	-	9
Register	-	-	9	-
Total	9	27	1707	1626

Figure 50: Sharpen IP hardware utilization.

Table 8: Comparison of FPGA board resources utilization for Sharpen IP core.

BOARD	LUTs	Flip-Flops	BRAM
Zybo	6.16%	4.85%	3.75%
Z1	3.06%	1.60%	1.43%
ZCU104 Ultrascale+	0.32%	0.04%	0.02%



(a) *Original grayscale image.* **(b)** *Sharpen processed.*

Figure 51: Sharpen - Grayscale image results (image source [10]).



(a) *Original color image.* **(b)** *Sharpen processed.*

Figure 52: Sharpen - color image results (image source [10]).

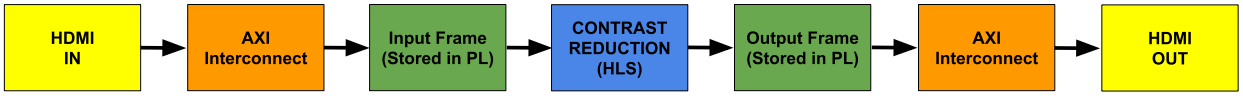


Figure 53: Contrast Reduction (contrast reduced image created by pixel operation of division) IP core flow chart. *The HDMI source signal is captured via the AXI interconnect and stored as an input frame within the Zynq PL. HLS code is then used to reduce the image contrast by dividing each R, G and B value of the pixel by three and then placed into the output frame before sending it out of the PL via AXI interconnect and then to HDMI sink.*



Figure 54: Contrast Reduction (pixel operation) IP core block diagram.

Timing (ns)

Summary

Clock	Target	Estimated	Uncertainty
ap_clk	10.00	6.38	1.25

Latency (clock cycles)

Summary

Latency		Interval		
min	max	min	max	Type
6	2081163	4	2081164	dataflow

Figure 55: Contrast Reduction (pixel operation) IP timing/latency.

Utilization Estimates

Summary

Name	BRAM_18K	DSP48E	FF	LUT
DSP	-	-	-	-
Expression	-	-	0	2
FIFO	0	-	30	120
Instance	-	1	411	461
Memory	-	-	-	-
Multiplexer	-	-	-	-
Register	-	-	-	-
Total	0	1	441	583

Figure 56: Contrast Reduction (pixel operation) IP hardware utilization.

Table 9: Comparison of FPGA board resources utilization for Contrast Reduction (pixel operation) IP core.

BOARD	LUTs	Flip-Flops	BRAM
Zybo	2.21%	1.25%	0.00%
Z1	1.10%	0.41%	0.00%
ZCU104 Ultrascale+	0.12%	0.01%	0.00%



(a) *Original color image.*



(b) *Contrast Reduction (33.3%).*

Figure 57: Contrast Reduction (pixel operation) - color image results.

accommodate the situation at hand. Convolution is also a significant portion of the operations applied to input images in CNNs. For this implementation of the Convolution operation, a 3 x 3 edge detection kernel as seen in figure 58 was applied to the image frame. The flowchart depicting this process for the IP core can been seen in figure 59.

After completing the design flow within Vivado HLS a Sharpen IP core was created that could be implemented within a video pipeline architecture. The top-level block diagram of this IP core appears in figure 60. The Vivado HLS process also provides diagnostic and implementation summaries to catalog the particular implementation of a HLS created IP core. Of particular interest are the timing/latency summaries in figure 61 and the hardware utilization summary in figure 62. A comparison of hardware utilization across the different boards appears in table 10. This custom IP core was validated using a grayscale image and a color image. These results are displayed in the figures 63 and 64.

-1	-1	-1
-1	8	-1
-1	-1	-1

Figure 58: Edge detection kernel.



Figure 59: Convolution operation IP core flow chart. *The HDMI source signal is captured via the AXI interconnect and stored as an input frame within the Zynq PL. HLS code is then used to detect edges in the frame via a 3x3 kernel and then placed into the output frame before sending it out of the PL via AXI interconnect and then to HDMI sink.*

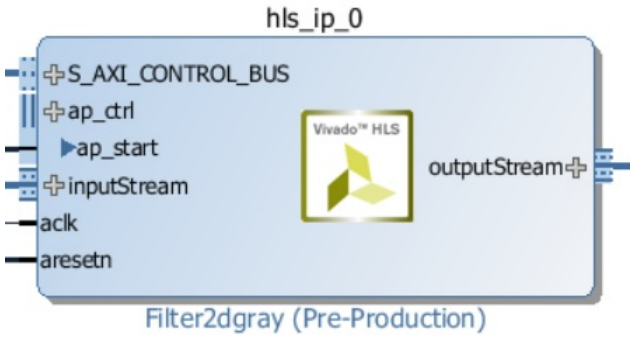


Figure 60: Convolution IP core block diagram.

Timing (ns)

Summary

Clock	Target	Estimated	Uncertainty
default	10.00	8.02	1.25

Latency (clock cycles)

Summary

Latency		Interval		
min	max	min	max	Type
192	2096256	167	2096231	dataflow

Figure 61: Convolution IP timing/latency.

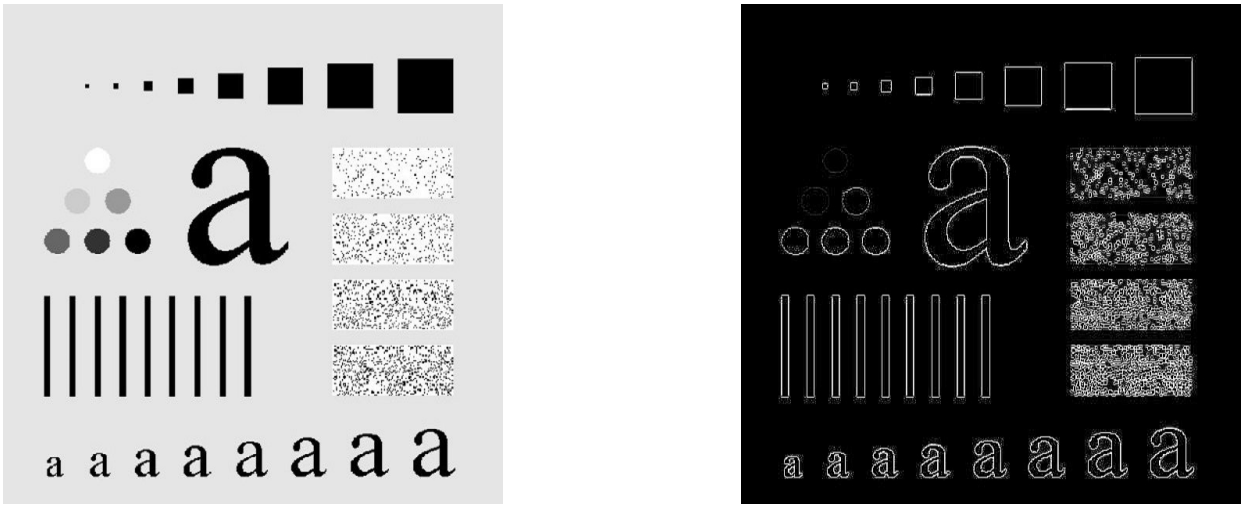
Summary

Name	BRAM_18K	DSP48E	FF	LUT
Expression	-	-	0	3
FIFO	0	-	105	444
Instance	9	27	2333	3226
Memory	-	-	-	-
Multiplexer	-	-	-	30
Register	-	-	21	-
Total	9	27	2459	3703

Figure 62: Convolution IP hardware utilization.

Table 10: Comparison of FPGA board resources utilization for Convolution IP core.

BOARD	LUTs	Flip-Flops	BRAM
Zybo	14.03%	6.99%	3.75%
Z1	6.96%	2.31%	1.43%
ZCU104 Ultrascale+	0.73%	0.06%	0.02%



(a) *Original grayscale image.* **(b)** *Convolution processed.*

Figure 63: Convolution - grayscale image results (image source [10]).



(a) *Original color image.* **(b)** *Convolution processed.*

Figure 64: Convolution - color image results (image source [10]).

Vita

David Cornett III was born April 22, 1979 in Lexington, KY. He graduated from the University of Kentucky with a Bachelor of Science in Computer Science in 2004. After 9 years of professional experience he returned to academia in 2013 to deepen his technical skills and transition his career path into research. He graduated in 2015 from the University of Kentucky with a second Bachelor of Science degree in Computer Engineering while maintaining full-time employment as an engineering intern. From there he entered graduate school at the University of Tennessee in 2016 in a part-time capacity while maintaining full-time employment first as a software developer and then transitioning to research staff at Oak Ridge National Laboratory focused in the areas of computer vision, machine learning and biometrics. He completed his Master of Science in Computer Engineering in 2021.