Benchmarking of Embedded Object Detection in Optical and RADAR Scenes

A Thesis Presented for the

Masters of Science

Degree

The University of Tennessee, Knoxville

Vijaysrinivas Rajagopal

December 2022

© by Vijaysrinivas Rajagopal, 2022
All Rights Reserved.

Acknowledgments

This thesis was a composition of the work that I did during my time as a graduate research assistant. During that time, I was able to work on various research projects and met amazing people who helped shape this thesis. I would like to thank each and every one of them for their assistance.

Firstly, I would like to thank Dr. Abidi for his mentorship and guidance throughout both my undergraduate and graduate research. The knowledge and opportunities during my time at IRIS labs has changed me into a better scholar.

Secondly, I would like to thank my parents, Rajagopal Sampath and Usha Sadagopan, for encouraging me throughout my entire educational career to strive higher and harder in everything I do. Additionally, I would like to thank my brother and sister, Narayanan Rajagopal and Shreeya Iyengar, for their support during my thesis process.

Thirdly, I would like to thank Dr. Fathy and his research students - Abdel-Kareem Moadi, Chandler Bauder, Marvin Joshi, and Peter Twill - for the opportunity to work on vital sign-based research projects. Additionally, I would like to acknowledge MaXentric Technologies LLC and IUCRC for funding my research. From MaXentric Technologies, I would also like to thank Yao Yu and Luna Zhang for sharing expertise and data. During this research project, I learned many new concepts regarding RF and embedded technologies, and I hope to carry them with me into future endeavours.

Finally, I would like to thank the students that assisted with the development of the GUI Application for Augmentation Analysis. These students include undergraduate research students Rus Refalti and Priya Patel. Additional credit also goes to the UTK Fall 2021 COSC/ECE 401 and 402 Senior Design teams. Individuals include Everett Holzapfel, Ria Patel, Ross Ketron, Brandan Roachell, Skylar White, and Chris Romanelli.

Abstract

A portable, real-time vital sign estimation protoype is developed using neural network-based localization, multi-object tracking, and embedded processing optimizations. The system estimates heart and respiration rates of multiple subjects using directional of arrival techniques on RADAR data. This system is useful in many civilian and military applications including search and rescue.

The primary contribution from this work is the implementation and benchmarking of neural networks for real time detection and localization on various systems including the testing of eight neural networks on a discrete GPU and Jetson Xavier devices. Mean average precision (mAP) and inference speed benchmarks were performed. We have shown fast and accurate detection and tracking using synthetic and real RADAR data.

Another major contribution is the quantification of the relationship between neural network mAP performance and data augmentations. As an example, we focused on image and video compression methods, such as JPEG, WebP, H264, and H265. The results show WebP at a quantization level of 50 and H265 at a constant rate factor of 30 provide the best balance between compression and acceptable mAP.

Other minor contributions are achieved in enhancing the functionality of the real-time prototype system. This includes the implementation and benchmarking of neural network optimizations, such as quantization and pruning. Furthermore, an appearance-based synthetic RADAR and real RADAR datasets are developed. The latter contains simultaneous optical and RADAR data capture and cross-modal labels. Finally, multi-object tracking methods are benchmarked and a support vector machine is utilized for cross-modal association.

In summary, the implementation, benchmarking, and optimization of methods for detection and tracking helped create a real-time vital sign system on a low-profile embedded device. Additionally, this work established a relationship between compression methods and different neural networks for optimal file compression and network performance. Finally, methods for RADAR and optical data collection and cross-modal association are implemented.

Table of Contents

1 Introduction **1**
- 1.1 Motivation . . . 1
- 1.2 Pipeline of Thesis . . . 2
- 1.3 Thesis Contributions . . . 5
- 1.4 Synopsis . . . 9

2 Literature Review **11**
- 2.1 RADAR Detection . . . 11
- 2.2 Detection and Tracking Methods . . . 16
  - 2.2.1 Detection Methods . . . 16
  - 2.2.2 Tracking Methods . . . 20
  - 2.2.3 Cross-Modal Association . . . 21
- 2.3 Software . . . 22
  - 2.3.1 CNN Quantization . . . 22
  - 2.3.2 CNN Pruning . . . 24
  - 2.3.3 Synthetic Data for Training and Evaluation . . . 24
- 2.4 Evaluation Metrics for Detection and Tracking . . . 25
- 2.5 Previous Sensitivity Analysis Work . . . 27
- 2.6 Hardware . . . 27
  - 2.6.1 Embedded Mobile GPU . . . 30
  - 2.6.2 FPGA . . . 30

3 System Layout **34**
- 3.1 Edge vs IoT . . . 34
- 3.2 Sketch and Physical Layout . . . 35
- 3.3 Device Interconnections . . . 41

4 Real Time Detection and Tracking **42**
- 4.1 Optical Data . . . 42
  - 4.1.1 VOC2007 & VOC2012 Datasets . . . 42
  - 4.1.2 COCO 2017 Dataset . . . 42
  - 4.1.3 Multi-Object Tracking Datasets . . . 43
- 4.2 Techniques for Optical Object Detection . . . 43
- 4.3 RADAR Data . . . 47
  - 4.3.1 Synthetic RADAR Data . . . 47

4.3.2 Real RADAR Data . . . 48
4.4 Training on Synthetic RADAR Data . . . 53
4.5 Training on Real RADAR Data . . . 57
4.6 Cross-Modal Association . . . 61
4.7 Techniques for Multi-Object Tracking . . . 65

5 Data Augmentation Analysis **69**
5.1 GUI Application for Augmentation Analysis . . . 69
5.2 Compression Experiment Setup . . . 72
5.2.1 Compression Method Selection . . . 72
5.2.2 Neural Network Selection . . . 73
5.3 Image Compression Results . . . 79
5.4 Video Compression Results . . . 87
5.5 Compression Conclusions . . . 92
5.5.1 Observations . . . 92
5.5.2 Practical Applications . . . 92

6 Prototype Development **94**
6.1 Hardware Flow . . . 94
6.2 Software Flow . . . 94
6.2.1 TCP/UDP Interface . . . 98
6.2.2 Minimum Variance Distortionless Response . . . 98
6.2.3 Detection Method . . . 98
6.2.4 Multi-Object Tracking . . . 99
6.2.5 Phase Shift . . . 99
6.2.6 Extended Functionality . . . 99
6.3 CNN Quantization (TensorRT) . . . 102
6.3.1 General Quantization Process . . . 102
6.3.2 Quantization Analysis . . . 106
6.3.3 Quantization Conclusions . . . 106
6.4 CNN Pruning Performance (SparseML) . . . 111
6.4.1 General Pruning Process . . . 111
6.4.2 Pruning Analysis . . . 111
6.4.3 Pruning Conclusions . . . 111
6.5 Real-time Implementation . . . 112
6.5.1 CNN Optimizations . . . 112
6.5.2 RADAR Processing Optimizations . . . 116
6.6 RADAR Performance Benchmarks . . . 116
6.6.1 Heartrate and Respiratory-Rate Performance . . . 116
6.6.2 FPS Performance . . . 117
6.6.3 Power Analysis . . . 117
6.6.4 Thermal Analysis . . . 121

7 Conclusion **122**
7.1 Summary 122
7.2 Contributions 124
7.3 Future Work 127

Biblography **130**

Appendix **130**

Vita **145**

List of Tables

3.1 **Table comparison of different devices and their SWaP characteristics.** *The requirements for the real-time system make a traditional desktop GPU infeasible due to its large size and extensive power draw. The same size requirements make the Jetson Xavier AGX impractical. The TE0807 and Jetson Xavier NX are better fits for this type of form factor.*
** - Weight does not include any sort of cooler.*
*** - Maximum power usage is dependent on the code running on the FPGA and the 15W is taken from [11].*
**** - 20W power options are available in JetPack 4.6 and newer. Older versions support 15W max.* 38

4.1 **Table showcasing different YOLO-based networks and their parameter sizes.** *Networks were chosen based on their notable improvements upon YOLOv3's architecture and training procedures. Networks, such as YOLOv5 and YOLOv4 CSP, are "scalable" networks, which give a large range of network subtypes. Such examples are YOLO5L (large), YOLOv5M (medium), etc. The network sizes for these scaled networks were chosen based on their approximate size to YOLOv3. YOLOv3 is considered a baseline network due to its significant literature impact. The "miniaturized" versions of the original networks (denotated by "Tiny" or "Nano" or "Small") are chosen for their deployability to embedded devices.* 45

4.2 **Inference FPS of all YOLO-based networks across discrete GPU, Jetson Xavier AGX, and Jetson Xavier NX.** *Newer networks, such as YOLOv5L and YOLOv4 CSP, are more efficient than previous networks due to architectural and training advancements. Sub 10-million parameter networks are all above 30 FPS across all hardware platforms and are prime choices for implementation into the final real-time system. The performance magnitude between the discrete GPU and Jetson Xavier AGX and NX are dependent on network size. For networks with sub-10 million parameters, 6.79x and 7.86x speed-up between the discrete GPU and the AGX and NX respectively is seen. For bigger networks, the magnitude drops to 3.95x and 6.94x for the AGX and NX respectively. Between the AGX and NX, the AGX runs the networks an average of 1.48x faster.* 45

4.3 **Table of different clip descriptions from the Range-Velocity dataset.** 52

6.1 **Table of ResNet50 top-1 accuracy and FPS performance on Imagenette under different pruning types.** *As shown in pruning literature, unstructured pruning can have high sparsity while preserving the majority of the accuracy performance.* 114

1 **Hyperparameters for synthetic RADAR network training.** *Other hyperparameters associated with a given network are the default values of their COCO training.* 135

2 **Hyperparameters for real data RADAR network training.** *Other hyperparameters associated with a given network are the default values of their COCO training.* 135

3 **A table showing the overview of different types of YOLO networks and their mAP performance**. *Evaluation is done on two main detection datasets - VOC2007 and COCO (2014 and 2017 versions). For papers that offered multiple network configurations, the most accurate versions are displayed.* 144

4 **Table of respiration and heartrate errors based on different ranges.** *The table also breaks error down based on single and multi-subject recordings. 1 meter recordings have the best performance regardless of number of subjects.* 144

List of Figures

1.1 **Different applications for a portable, real-time vital-sign estimation system.** *The system's real-life applications include surveillance of large medical areas, as well as detecting life signatures in first responder and military operations.* 3

1.2 **Thesis pipeline of training, testing, refining, and deploying various types of detection, vital sign estimation, and streaming methods for real-time application.** *This diagram splits the thesis into two main phases - the "Analysis" and "Application" Phase. The former consists of developing methods (such as detection via neural networks) by training them against optical and non-optical datasets. Additionally, testing is done for baseline performance, but there is also "robustness" analysis where networks are subject to degraded inputs. This phase is cyclical in nature as we analyze new networks, optimize them, and re-analyze their performance. These types of analysis can give an insight into potential performances discrepancies when deployed in the field. In the "Application" Stage, we take the methods and lessons learned and applying them to a prototype system for vital sign estimation. This thesis analyzes and benchmarks system performance and adds additional functionality, such as GUI and web-based streaming.* 4

2.1 **Figure 1 from [1] shows an illustration of the hardware and general software process for RF-Capture.** *The hardware is composed of two antenna arrays shaped in a "T" arrangement. The "T" arrangement is composed of vertical antenna transmitters and horizontal receivers. In terms of analyzing and producing a pose estimation from the antenna array, RF-Capture first runs a "Coarse-to-Fine" algorithm that generates a "coarse" RF scan, which is used to localize and help retrieve a higher resolution scan. The higher resolution scans are stacked together over time, and this allows for body segmentation. An SVM was also used to do subject classification on the capture human figure.* 13

2.2 **Illustration of the cross-modal "teacher-student" training process.** *RF-Pose utilizes a well-established RGB-based pose estimation network as a "teacher" network. The RF-based pose estimation network is a "student" network, which will train on the RADAR data while receiving loss data information from the teacher network. The idea is to transfer knowledge of an RGB-derived pose to produce an RF-derived equivalent pose.* 13

2.3 **Network pipeline illustration redesigned from Figure 6 in [36].** *In this pipeline, RF readings from the FMCW RADAR are put through a feature extractor (ResNet50), and the feature maps are then processed through a Regional Proposal Network (RPN) to generate limb localization. The RPN output is then processed through an LSTM to produce the final pose estimation output. Additionally, a "descriptive encoding" is produced from the RoI pooling, which can be used for pose classification in multi-subject scenarios.* . . . 15

2.4 **Overview of RODNet's cross-modal training structure.** *This diagram was taken from the original paper [54]. RODNet takes advantage of a Camera-Radar-Fusion technique that utilizes an optical 3D detection method and CRF for RADAR peak detection. These two inputs generate automatic annotations, which are then fed into the RODNet's training cycle. RODNet comes in three distinct architectures with varying AP and FPS performance. All networks take in clips of RAMap data to produce their confidence maps.* 15

2.5 **Common computer tasks and their differences against object detection.** *The diagram is adapted from Figure 1 in the COCO dataset paper [32]. Object detection is conceptually similar to instance segmentation, but it differs in the coarseness of the localization. Rather a pixel-level mask, it requires a "bounding box" that represents both the location of an object and the classification of the object.* 17

2.6 **Structure of a YOLO CNN object detector**. *This diagram is taken from [24]. It shows the general process of a one-stage detector. An image is resized into the CNN's input shape and put through a detector. The resulting bounding boxes are filtered with an algorithm called non-max suppression (NMS). As the the input is passed and processed through the network layers, the intermediate values generated are intended to represent certain characteristics of a given image, whether that is the shapes, texture, or spatial relationship of the objects in the scene.* 17

2.7 **List of different types of tracking.** *This figure is taken from UniTrack [55] and shows Single Object Tracking (SOT) and Multi-object Tracking (MOT). Both are bounding-box-based tracking tasks, but there are analogous tracking tasks for both segmentation and pose-estimation outputs [55].* 19

2.8 **Diagram of categorized tracker methods discussed in this work.** *A caveat in this categorization is that the IoUTracker does not have a predictive function.* 19

2.9 **General overview of ViFi's modalities, task, and proposed solution.** *The diagram is Figure 2 from the original paper [33]. The deep-learning-based solution exhaustively matches features from person's optical data to a phone's MTU and FTM data. They utilize a LSTM to retain memory from previous frames to improve accuracy.* 23

2.10 **Illustration of MOTSynth's different annotations output [16].** *These annotations are achieved by utilizing the in-game scripting language of Grand Theft Auto V.* 23

2.11 **Decomposition structure of HOTA metric.** *This diagram is inspired by Figure 3 from [51], and describes the "modularity" of the HOTA metric. We calculate detection accuracy (DetA) and then calculate the association accuracy (AssA) with regards to the sets of true positives from DetA. Finally, we can calculate a secondary localization accuracy metric from HOTA from the composition of DetA. The localization metric tells us how well the tracking method is able to place a bounding box over a given object.* 28

2.12 **Examples of augmentations and corrections applied to images and detection result differences across hardware platforms.** *Because of limited operations on the Xilinx FPGA platform chosen in [11], the implemented networks are slightly modified and can behave differently under certain augmentations. Additionally, correction methods for each augmentation chosen in Leach and Cornett's work show effective and fast methods of image reconstruction with noise and blur-based augmentations. Finally, networks of all types can suffer from degrading augmentations, such as pose and dept estimation network.* 29

2.13 **Block Diagrams of Jetson NX and AGX.** *Taken from NVIDIA Datasheets from the respective devices, and both block diagrams show the wide array of I/O opportunities for streaming in and out large amounts of data. For example, the Camera Serial Interface (CSI) ports on both devices enable for 1080p and higher resolution input from a camera, and USB 3.x and Gigabit Ethernet for high throughput video streaming. Their respective Volta-based GPUs and ARM-based CPUs deliver competitive SWaP (Size, Weight, and Power) characteristics compared to other embedded devices; especially for neural network applications.* 31

2.14 **Block diagram of the 7EV MPSoC.** *The MPSoC contains multiple ARM Cortex-based CPUs as well as a ARM Mali-based GPU. These enable the ZCU104 to achieve high throughput tasks, such as streaming multiple high-resolution videos at once. Additionally, a Data Processing Unit (DPU) enables fast neural network operations such as convolution, dense, and pooling operations.* 33

3.1 **Illustration of the typical flow of data and processing edge-based devices and IoT-based devices.** *For edge-based devices, the sensor data is processed on the device itself. Edge devices can also display the data to an immediate end-user as well as streaming data to a network of other end-users. IoT-based devices collected sensor data and transmit that data to a compute architecture. This architecture typically contains more compute power and processes the data quickly. The results can be distributed to other users and the IoT device itself.* 36

3.2 **Examples of IoT and Edge Devices.** *With IoT devices such as the Aurdino Uno Wifi and NodeMCU, the hardware consists of a main re-programmable microcontroller with low compute power. There are additional chips for connectivity, including low-power bluetooth and WiFi modules. Software-wise, they require a host computer to flash a program onto the microcontroller. IoT devices also very affordable, with both the Aurdino Uno WiFi and NodeMCU under $60 per unit. For edge devices, the Jetson Nano and Raspberry Pi 4 are traditional examples with far more capable and powerful ARM-based processors. They also have similar connectivity modules, but the Jetson devices can cost more than $200 per unit depending on the device type. The software development is similar to traditional development, where code can be assembled and executed on the device itself.* 37

3.3 **An initial sketch of the design of the integrated system for vital-sign system.** *The compute device portion was the primary focus of this thesis as its purpose is to run sensor collection and processing code. Additionally, it was also responsible for tasks, such as detection and classification of individuals as well as regression for heart rate and respiratory rate. Similar to a typical smartphone design, the integrated system would consist of the compute device between the end-user display and the sensors.* 38

3.4 **Three main hardware considerations - Xilinx UltraScale+ MPSoC ZCU104, Jetson Xavier AGX, and Jetson Xavier NX.** *The ZCU104 provides a robust level of compute power as it contains media engines that can stream multimedia (such as 4k video input) as well as utilizing the DPU for accelerate neural network inferencing. Both Jetson Xavier devices also provided equivalent performance for their size and weight, with the AGX able to handle large neural networks with its 32GB of shared memory. The Xavier NX provides similar proportional performance when considering the size-to-compute ratio.* 40

4.1 **Image examples from the VOC datasets, COCO 2017 validation dataset, and MOT20 dataset.** *All of the aforementioned optical datasets contain a diverse set of images and a relatively large number of classes. In terms of total dataset sizes, VOC2007 contains 9,963 images, VOC2012 contains 11,540 images, and COCO 2017 contains 123,287 (not including the testset) images. The VOC family of datasets contain 20 generic classes (persons, bottle, cars, etc.) while the COCO dataset contains 91 classes with similar generic labels. The important distinctive for class labels in COCO val2017 is between "object" and "stuff" categories. "Stuff" class labels are applied to things in an image that do not have clearly defined boundaries. For our purposes, we do not include the "stuff" category classes, which sets the number of usable classes to 80. The MOT data is composed of video folders with each image being an individual frame.* 44

4.2 **Graph showing the mAP results of various YOLO-based networks on COCO val2017.** *All networks were evaluated directly from the Python COCO evaluation tools, and the mAP results are broken into mAP@0.5 (mAP at an IoU threshold = 0.5) and mAP@[0.5:0.95:0.05] (mAP at a range of IoU thresholds from 0.5 to 0.95 in intervals of 0.05). The latter enforces a **stricter** mAP computation, which is believed to have a closer association with real-life detection performance. From the evaluation, YOLOv4 achieves the highest mAP@0.5 and mAP@[0.5:0.95:0.05] with 68.21% and 49.41%. Although, all other similarly-sized YOLO networks achieve very similar mAP scores. Between the miniaturized models (sub-10 million parameter networks), YOLOv4 CSP Small achieves the best mAP scores while containing **less** parameters than YOLOv3 Tiny. YOLOv5n is also very competitive as it does better than YOLOv4 Tiny and YOLOv3 Tiny while being 3 to 4x smaller. Observations of the two mAP metrics show that there is just a static offset, and there is no change in network rankings between the two metrics.* 46

4.3 **Example outputs from Synthetic RADAR Data v1.** *Because no prior RADAR data directly applicable to the vital-sign product was available, the first attempt of creating synthetic was very simple. There would be a single signature with a set of static, curved lines to produce some perturbations. Additionally, a similar curve goes through the signature to add more irregularity.* 49

4.4 **YOLOv3 trained on Synthetic RADAR Data v1 tested on the Range-Velocity test data.** *The mAP results on the Range-Velocity dataset average out to 71%. This can be attributed to the Range-Velocity data having similar visual simplicity compared to Synthetic RADAR Data v1.* 50

4.5 **YOLOv3 trained on Synthetic RADAR Data v1 tested on MVDR-based test data.** *Unlike Figure 4.4, YOLOv3 trained on Synthetic RADAR Data v1 scores an average mAP of 31%. The inconsistent performance between the two real RADAR data indicates that the Synthetic RADAR Data v1 does not generalize well.* 50

4.6 **Example outputs from the appearance-based synthetic RADAR.** *Unlike Synthetic RADAR Data v1, this dataset utilizes a range of different canvas scale. Additionally, the radial line is removed and replaced with stochastic perturbations that can "smear" the primary, circular signature. Furthermore, there are additional degradation methods including static, background perturbations as well as signatures with variable intensity during a certain path.* 51

4.7 **Examples of Range-Velocity RADAR data.** *The data was collected by Yao Yu with a Sensorlogic UWB RADAR. Fast Fourier Transform (FFT) and a clutter suppression algorithm is applied to produce a 2D matrix, where the x-axis represents "range" and the y-axis representing "speed".* 52

4.8 **Examples of MVDR-based RADAR data.** *With MVDR, positional data is more intuitive compared to the Range-Velocity approach. The x-axis is the angle from the center (where the RADAR is located) while the y-axis is the physical range from the RADAR. For the FMCW RADAR device, the maximum range is around 7 meters.* 54

4.9 **mAP comparison of multiple object detection networks on the Range-Velocity test set.** *The dataset is composed of two subjects walking in five different methods and in two different environments - through wall and non-through wall (also annotated as "freeAir"). Unlike Figure 4.4, the YOLOv3 network trained on Synthetic RADAR Dataset v2 scores 9% higher mAP. All networks, even the networks with less than 10 millions parameter, achieve a higher average performance than the performance of YOLOv3 shown in Figure 4.4.* 55
4.10 **mAP comparison of multiple object detection networks on a MVDR-based RADAR test set.** *The dataset is composed of four different clips of a single subject walking in a given direction or pattern. The average mAP score between the YOLOv3 shown in this figure is ∼40% higher than the YOLOv3 shown in Figure 4.4. This indicates that this dataset is more generalizable than the first version. Additionally, a comparison between the different YOLO-based networks show that YOLOv5L on average scores the highest mAP across both datasets. The performance is quite noticeable on the Range-Velocity data. Meanwhile, the mAP delta of YOLOv5L and YOLOv4 CSP on the MVDR-based is less notable. Sub-10 million parameter networks predictably have worse performance than their larger counterparts. Additionally, their performance rankings greatly differ amongst themselves between the two datasets.* 56
4.11 **Example of dataset collected using real-time data collection.** *This dataset collected clips of one, two, and three subjects. Each clip was 300 frames long and simultaneously captured RADAR and optical data. The RADAR data was processed using MVDR, similar to the MVDR-based dataset in Chapter 4.* 58
4.12 **Examples of dataset annotations.** *Similar to the datasets detailed in Chapter 4, this dataset provides bounding box annotations for RADAR signatures. To expand further than the other two datasets, faces in the optical dataset are also annotated. For future research objectives, "association" annotations are also created to help map detections across modalities.* 59
4.13 **Training results with sub-10 million parameter networks on the real-time MVDR data.** *There is no consistent trend regarding network performance across image sizes, but YOLOv4 CSP Small still produced the best results with an image size of 416, and it out-performed its synthetic counterpart with an mAP of 84.61% vs 64.52%.* 60
4.14 **Diagram showing the reframing of association task as a graph-like problem.** *A SVM-based binary classifier determines whether or not a given RADAR and Optical pair exists. The pairs represent an edge in a graph, and a complete bipartite graph is created during training and inferencing. During inferencing, the predictions classified as false are removed.* 62

4.15 **Classification accuracy of SVM training for cross-modal associations.** *10-folder cross validation is done on the training set on a seperate validation set for each fold. The average validation accuracy 96.26%. After each fold, we also run a seperate test set on the SVM, and the average test accuracy is 94.25%. The SVM fitted on a certain fold with the highest test accuracy is saved.* 63
4.16 **Erroneous predictions produced by SVM-based association technique on testset.** The majority of errors occur when detections in either modalities are relatively close to one another. This causes the SVM to produce a false positive association. 64
4.17 **Bar chart showing the HOTA scores for all trackers for both MOT20 and MOT17.** *From this evaluation, we can see that BYTE outperforms all other trackers in both datasets. It can also be seen that SORT has a significantly worse performance on MOT20 compared to other trackers. Additionally, the delta between SORT's performance on MOT20 and MOT17 is much higher than the delta of other trackers. This can attributed to MOT20's higher density environments along with SORT restricting itself to only analyzing locations. DeepSORT utilizes the same location analysis but also incorporates appearance vectors, which allows it to have a notable 19% higher HOTA on MOT20 compared to SORT. IoU Tracking scores a competitive HOTA score compared to other methods. This is due to the predictable nature of the majority of subjects in the video clips.* 66
4.18 $HOTA_\alpha$ **curves for both MOT17 and MOT20 training sets.** *The curves are an intermediate step for the final HOTA values shown in Figure 4.17. But analyzing these curves can give an idea how well the tracker is able to maintain its HOTA value as α increases. In this case, α is just the IoU threshold for the detection matching phase defined in Chapter 2. This essentially means the trackers are offered less and less boxes to do detection and association matching. This typically means the HOTA at higher α thresholds are lower. From Figure b, we can see that SORT suffers HOTA loss at much earlier α value compared to the rest of the trackers. This can be because of the higher pedestrian density in MOT20 compared to MOT17.* 67
4.19 **Comparison of average tracking method speeds on Desktop CPU and GPU, AGX, and NX.** *For all compute platforms, all methods ran on the CPU (AMD Ryzen 2700X) and DeepSORT ran on the Desktop GPU (NVIDIA 2070 Super). The fastest method was IoU Tracker, and this is due to the lack of advanced state estimation or neural networks, which are present in the other methods. Compared to the desktop, AGX and NX run times were 2.3x and 3.9x slower for BYTE, SORT, and IoUTracker on both datasets, respectively. DeepSORT fails to keep real-time speeds on MOT20 due to its high density environments. All MOT17 and MOT20 runs for each platform and for each tracker can be found in Figures 6, 7, and 8 in the Appendix.* . . 68

5.1 **Screenshot of the main window of the GUI Application for augmentation analysis.** *To add or remove or edit augmentations, clicking the "Add Augmentations" button in the bottom-left corner will present the dialog window shown in (b). After adjusting augmentations, parameters, neural networks, and images, clicking the "Run Experiments" button in the bottom-right corner will run the experiment. The results are shown in a separate experiment dialog box (c).* 70

5.2 **Software flow of GUI Application for augmentation analysis.** *The GUI reads from three main data sources. The first source is an augmentation file that contains all the low-level functions to transform an image or frame. Each one of these functions are wrapped around an abstract class called an "Augmentation". This abstract class allows all functions to integrate into the "AugmentationPipeline" class, which controls the execution and order of all chosen augmentations. Similarly, neural networks are managed via a dictionary of network objects. Each network object is a custom class that inherits another abstract class called a "Model". This abstract class requires an inherited class to implement functions such as initialization, inference, and drawing. This is so that calls and output to each network are standardized and makes developing and process later elements much more efficient.* 71

5.3 **Example of JPEG compression applied to an image.** T*he parameter being changed within JPEG is the "Quantization Level", which is one of the final steps in the algorithm and is what decides which image features are kept.* 74

5.4 **Example of WebP compression applied to an image.** *The variable is the "Quality", which is similar to JPEG compression's "Quantization Level".* 75

5.5 **Examples of Bilinear Resizing, H264, and H265 compression to a given image.** The variable for bilinear resizing is the percentage of the original width and height of the image. For H264 and H265, the typical parameter to adjust would have been the "Constant Rate Factor" (CRF). Rather than using CRF, this analysis uses the quantization parameter in order to be consistent in terms of benchmarking on COCO val2017. 76

5.6 **Example of Gaussian Entropy Network from [9].** *The network can provide different levels of compression, but it is incorporated into the training process and network structure. Training the network would take a prohibitively long time, so the variable is the different "quality" models that have been pretrained by CompressionAI. This quality ranges from 1 to 6, where 1 produces the most compressed image.* 77

5.7 **Diagram illustrating two main compression methods.** *In a streaming-based scenario, the streaming device would have to compress the sensor input and send it over the network to a larger compute architecture for processing. The analysis's compression methods are broken into three different categories and concepts. Image, video, and neural network-based compression are different in terms of resulting data that is transmitted. For image-based compression, the ultimate output is a compressed image. That image itself is sent over the network. For video-based compression, the typical data that is transmitted is a stream of encoded video rather than a discrete frame. The receiving client needs to have a playback or decoder of some kind to reinterpret the H264 encoding into a frame. The Gaussian Entropy Network is made up of an encoder and decoder. The encoder network turns an image to quantized feature maps that describe the image. Those feature maps are transmitted to the decompression network, which reconstructs the image.* 78
5.8 **Bilinear Compression of COCO val2017 and evaluation of YOLOv3, YOLOX, and DETR.** *Performance across all networks start to decrease significantly around 40% of the original image size.* 79
5.9 **Detection examples from DETR, YOLOv3 Ultra, and YOLOX on Bilinear Resizing.** *DETR is observed to produce more false positives at levels below 40% when compared to YOLOv3 Ultra and YOLOX. For this particular augmentation, this might be due to DETR being trained at large image sizes (minimum width of 800) compared to the YOLO networks (a ratio-preserved 640x640 image).* 80
5.10 **JPEG Compression of YOLOv3, YOLOX, and DETR.** *The general trend is similar to the observations of [45], where the mAP@0.5 for all networks is within 10% of their original score between quantization level of 100 to 40. Once quantization level of 20 is reached, all networks see a precipitous drop in mAP. At the highest quantization level, all networks are very close to 0% mAP. This is reasonable since the compression has removed many features to an unrecognizable point.* 81
5.11 **Detection comparisons between DETR, YOLOv3 Ultra, YOLOX on JPEG Compression.** *All detections were filtered at a confidence threshold of 35%. All networks decline in localization accuracy as the quantization level decreases. Similar to the bilinear resizing, DETR produces a higher rate of false positive detections up until a quantization level of 0. At this level, all networks report little to no high confidence detections even though the batter, catcher, and umpire are recognizable to a human.* 82
5.12 **WebP Compression of YOLOv3, YOLOX, and DETR.** *WebP retains more mAP performance at more aggressive compression parameters compared to JPEG. Although, the dropoff in performance for DETR and YOLOX occurs sooner than their performance on JPEG.* 83
5.13 **Detection examples of all networks on WebP Compression.** *All networks produce stable bounding boxes across all of the WebP parameters.* . 84

5.14 **Gaussian Entropy Network Compression of YOLOv3, YOLOX, and DETR.** *The Gaussian Entropy Network's mAP decline is more linear and less impactful than other image compressions.* ***Note:*** *parameter "7" is the image with no compression.* 85
5.15 **Detection examples of all networks on the Gaussian Entropy Network.** *YOLO-based networks lose smaller objects, such as the soccer audience, at lower quality levels. On the other hand, DETR produces more false positives, which follows observations in Bilinear and JPEG.* 85
5.16 **Average size reduction from image matrices in COCO val2017.** *The Gaussian Entropy Network is able to reduce the original image matrix to the smallest amount of encoded data. This supports the findings in the original paper and CompressionAI's benchmarks [7] [9].* 86
5.17 **Average percent reduction of encoded image data in COCO val2017.** *This graph complements Figure 5.16 by comparing the highest quality encoded data to data that is further compressed. For JPEG and WebP, the majority of reduction occurs within the lowest compression parameters. Bilinear resizing does not encode the image matrix so trend is identical to Figure 5.16 is seen here. With Gaussian Entropy Network, a quality parameter of 1 is 80% smaller than the highest quality parameter.* 86
5.18 **H264 Compression of YOLOv3, YOLOX, and DETR.** *mAP performance degrades starting at a quantization level of 30%. This corresponds with image quality as it also degrades significantly after that quantization level (refer to Figure 5.19).* 87
5.19 **Detection examples of all networks on H264 compression.** *There is an intense decrease in image quality from quantization level of 35 to 60, which corresponds with the drop in mAP performance.* 88
5.20 **H265 Compression of YOLOv3, YOLOX, and DETR.** *mAP performance is almost identical to trends seen in H264, but in the 30 to 50 range, all networks suffer worse mAP performance under H265.* 89
5.21 **Detection examples of all networks on H265 compression.** *All quantization parameters retained enough information for all networks to detect the major objects (batter, catcher, umpire, etc.).* 90
5.22 **Average percent reduction of video clips in the MOT17 training set.** *CRF was the variable since this is the processing of a video. But H264 dynamically adjusts the quantization parameter to be around the set CRF value, so there can be some correlation made between the reduction from the CRF and mAP performance of the quantization parameter. Similar to observations seen in Figure 5.22, H264 outperforms H265 in terms of video file size reduction. Both methods see a plateau of size reduction around a CRF of 30. Quantization levels around this level show stable mAP performance for all networks. This indicates an optimal balance of size reduction and mAP performance at a CRF of 30.* 91

6.1 **Hardware diagram showcasing the specific devices used in the final prototype.** *Reiterating the decision made in Chapter 3, we utilize the Jetson Xavier NX module for the main compute device. It is connected via ethernet to the RFBeam VMD3 and via CSI to a IMX519 camera. A 5.5inch AMOLED display is also connected via HDMI to the Xavier NX.* 95
6.2 **Images of the prototype enclosure for vital-sign estimation.** *We replaced the NVIDIA Jetson DevKit shown in Figure 6.3 with the NCB00. The frame that holds the AMOLED screen was also heightened to give more room to the low-profile fan on the NCB00.* 96
6.3 **Diagram showing the different prototype iterations.** *The first prototype was a simple connection of the different core devices. The second prototype had a rectangular enclosure that housed all the major components as well as a cooling system. Both iterations utilized a Jetson Xavier NX DevKit.* 97
6.4 **HOTA performance of tracking methods on Range-Velocity and MVDR-based RADAR datasets.** *DeepSORT does not benefit from encoding any appearances of the RADAR signatures due to their consistent changes. BYTE still proves to be the most efficient at tracking MVDR videos. Although, the overall HOTA scores are significantly lower than in optical methods. This can be because of detections not be temporally stability, causing excessive track ID creation.* 100
6.5 **High-level and low-level breakdown of system software components.** *At the launch of the integrated system, the program starts with the "Main Process", which creates other asynchronous processes. Each asynchronous process is labeled with a number to represent which order they are started and a header for their general purpose. The "Communication" module receives the VMD3 data and writes to a pool of shared memory. Once a set number of frames have been collected by the VMD3, it is processed for targets in the main process. The "Respiration" and "Heartrate" modules take the angle and range locations to compute their respective outputs for each target. The outputs are sent back to the main process for visualization.* 101
6.6 **Illustration of GUI-based streaming method.** *The GUI consisted of a series of demos that could run on the Quark NX, and the GUI communicated with the Quark via an Ethernet system. This same connection system would later be employed with a web-based streaming system.* 103
6.7 **Diagram flow of streaming via ethernet between a Jetson device and a "Client" Computer.** *The Quark NX was the main use case for this since it is "headless" (i.e. no display outputs). The streaming, inferencing, and command-listening components on the Quark side happen on their own separate threads. Streaming is enabled via GStreamer and NVIDIA-accelerated H264 encoding. Receiving the stream uses FFMPEG's RTP streaming protocol.* 104

6.8 **Diagram showing current web-based streaming system.** *A web process handles two things - the streaming protocol (WebRTC) and handling web traffic (AGSI web server). A client device can connect and access the stream via a simple website. WebRTC on both the NX and a client device establish connection and start an RTP stream that is similar to "GUI-Based Streaming Data".* 105
6.9 **Screenshot of the website a client device would visit in order to see the Web-RTC-enabled video stream.** *The website is a simple static website with very a local javascript file that does the WebRTC negotiation* . . 105
6.10 **mAP @ IoU=0.5 performance of YOLOv3, YOLOX, and DETR on COCO val2017.** *mAP performance between FP32 and FP16 models are identical because the range of values between the two data types are approximately equal. When it comes to INT8 quantization, this is not the case and network performance can drastically drop if the weights are not rescaled to INT8 properly. With the previously mentioned TensorRT quantization process, all networks lost less than 8% of their original mAP performance. All networks used the same quantization algorithm and the same calibration dataset. YOLOX fared the best in terms of mAP delta between INT8 and FP32.* 107
6.11 **Inference speed for YOLOv3, YOLOX, and DETR at different precision types on the Jetson Xavier NX.** *YOLOv3 inference speed increased the most across FLOAT16 and INT8 precision types. There was a 3.95x increase in speeds when comparing the FLOAT16 model to the FLOAT32 model. There is a 6.61x increase between INT8 and FLOAT32 and a 1.67x increase between INT8 and FLOAT16 respectively. Meanwhile, YOLOX runs the fastest overall with an INT8 model reaching an FPS of 43.36. This can be attributed to the lower parameter count when compared to YOLOv3 and DETR. DETR, even at the lowest precision type, does not run as fast as the CNN-based detectors. Since DETR's feature extractor is ResNet50, it is probable that the bottleneck is the traditional transformer network.* 108
6.12 **Inference speed for YOLOv3, YOLOX, and DETR at different precision types on the Jetson Xavier AGX.** Run time improvements on the Xavier AGX follow a similar trend as the Xavier NX. 109
6.13 **Detection differences between FLOAT32/FLOAT16 and INT8 on YOLOv3, YOLOX, DETR.** There is notable decrease in detections on the INT8 results, where subjects and items in the background are lost or fall under the confidence threshold (0.25). 110

6.14 **Comparison of ResNet50 file sizes between structured and unstructured pruning.** *For unstructured pruning, the maximum filesize reduction was over 70% (from 85,000 bytes to 25,000 bytes) at an initial compression ratio of 1. Proceeding compression ratios reduced filesizes to a much lesser extent but consumed more CPU utilization as the ratio increased. The initial filesizes without compression are the same between pruned and unpruned because all the pruned weights are "zeroed" out. Additionally, the compression applied to the unpruned models have very little effect after the first compression ratio. For structure pruning, entire convolutional filters were removed rather than zeroed out, which results in the model automatically being smaller without any compression. Applying compression yields very little filesize savings and like its unpruned counterpart, does not change after compression ratio of 1. As expected, structured pruning yielded less sparsity with a maximum filesize reduction of ∼56% at no compression.* 113

6.16 **mAP comparison between different sparsity levels of YOLOv2.** *This bar chart benchmarks the mAP performance of YOLOv2 at no sparsity, 77%, 71%, and 66% sparsity on VOC2007 test set. All these models are sourced from Xilinx's Model Zoo for the Ultrascale FPGA products (although these runs were on a discrete GPU). This shows us that heavily pruned object detectors can still retain the majority of their original mAP score. Class-wise, the mAP discrepancy ranges from 1% to 6%.* ***Credit for assistance: Priya Patel*** 114

6.15 **Performance and magnitude improvements between non-vectorized algorithm, vectorized algorithm on CPU, and vectorized algorithm on GPU.** *From sub-figure a, we can see that the original non-vectorized algorithm was not meeting the minimum FPS threshold (1 FPS) on the Xavier NX. Vectorizing and implementing on the GPU allowed a speed up of 50x (sub-figure b) on the Xavier NX. Reducing the number frames fed into MVDR can give even more headroom for other processes.* 115

6.17 **Respiration-rate (a,c) and Heart-rate (b,d) error rates.** *The average heartrate and respiration rate error across all ranges for single subject is 2.52 and 0.346 BPM, respectively. For multi-subject, average heartrate and respiration rate error across all ranges is 2.7 and 0.68 BPM, respectively.* ***Credit for assistance: Abdel-Kareem Moadi*** 118

6.18 **Chart showing the average FPS of the real-time pipeline over 2000 timesteps.** *The speed is the accumulated time from the "wait" for RADAR data accumulation, processing through a given DoA method (ex: MVDR), localization with the NN method, tracking with BYTE, and other miscellaneous operations that occur on the main thread. It can be seen that the higher wattage power profiles are able to run the pipeline at higher speeds. It can also be observed that the pipeline under 10W 4 Core is marginally faster than 15W 2 Core. This signals that the number of cores matters more than the frequency each core is running on. Figure 6.5 supports this conclusion since the pipeline relies heavily on multi-processing.* 119

6.19 **Chart showing power usage of the Xavier NX over 300 timesteps.** *It is quite obvious that the higher wattage power profiles with associated higher CPU core counts consume more wattage. What is interesting though is the relative similar measurements between 15W 2 Core and 10W 4 Core.* 120
6.20 **Chart showing the average temperature observed on both the CPU and GPU temperature sensor over 2000 timesteps.** *Each run is buffered with a 10 to 20 minute wait to reset to an idle temperature of around 45 to 50C. Regardless of power profile, the temperature increase is almost identical in both value and in their linear trend. Also, as maximum wattage increases, the slope of the temperature increase increases.* 120

7.1 **Proposed multi-RADAR system for monitoring moving subject respiration and heartrate.** *Previous works by [61] have shown the possibility of using encoder-decoder neural networks to remove movement noise from a RADAR device. Exploring different network structures for this noise removal as well as incorporating more than two RADAR devices can improve respiration estimation method robustness. This also implies that the current real-time system described in Chapter 6 would need to be adjusted to handle additional sensor processing. A reconsideration of compute devices would be necessary to handle this amount of RADAR data.* 129
2 **Training mAP graph for YOLO-based networks on Synthetic RADAR v2.** 136
3 **Training mAP graph for YOLO-based networks on real-time MVDR-based RADAR data.** 137
4 **Training localization loss graph for YOLO-based networks on Synthetic RADAR v2.** 138
5 **Training localization loss graph for YOLO-based networks on real-time MVDR-based RADAR data.** 139
6 **Comparison of tracking method speeds on Desktop CPU and GPU.** *All methods ran on the CPU (AMD Ryzen 2700X), but due to its small CNN feature extractor, DeepSORT also utilized the Desktop GPU (NVIDIA 2070 Super). The model was kept at FLOAT32 and had a batchsize that changed to fit the total number of individuals detected. The fastest method was IoU Tracker, and this is due to the lack of advanced state estimation or neural networks, which are present in the other methods.* 141
7 **Comparison of tracking method speeds on Jetson Xavier AGX.** *All methods ran on the same hardware layout as described in Figure 4.19. The feature extractor in DeepSORT was converted to TensorRT. It was quantized to FP16 to give it an equal footing with other methods while not reducing any potential mAP loss via PTQ. Trends seen in Figure 4.19 mostly maintain in the AGX runtimes, with the only difference being BYTE outperforming IoUTracker in MOT20-03. The AGX runtimes were on average 2.3x slower for BYTE, SORT, and IoUTracker across both datasets. For DeepSORT, MOT17 runtimes were 4.1x slower and MOT20 runtimes were 6.2x slower.* . 142

8 **Comparison of tracking method speeds on Jetson Xavier NX.** *Trends seen in the AGX runtimes continue in the NX runtimes. The NX runtimes were on average 3.9x slower for BYTE, SORT, and IoUTracker across both datasets. For DeepSORT, MOT17 runtimes were 5x slower and MOT20 runtimes were 8.5x slower.* 143

Chapter 1

Introduction

The objective in this thesis is to benchmark and analyze different methods for traditional computer vision tasks, such as detection of common objects. These methods include real-time object detectors and multi-object trackers. Additionally, we extend these methods to RADAR-based data to integrate vital sign estimation methods in order to produce a robust real-time system. Furthermore, we optimize neural network methods necessary for real-time speeds.

Finally, software and hardware architecture decisions for the real-time system are produced. In addition to the core detection methods and vital sign estimation functions, auxiliary functions, such as a GUI-based streaming method and a web-based streaming method, are implemented. For practical purposes, the real-time system's power usage, pipeline speed, and temperature curves are benchmarked.

1.1 Motivation

Since early 2010s, there has been an exponential increase in the attention and development of Deep Learning-based neural networks. In computer vision, Convolutional Neural Networks (CNNs) have drastically improved tasks related to localization and classification of various objects. This has been facilitated by large, open-sourced datasets such as ImageNet and Common Objects in Context (COCO), which contain thousands of images and a wide array of object classes. With this large amount of data, CNNs have been able to achieve state-of-the-art (SoTA) performance in the context of matching human localization and classification abilities.

Furthermore, recent advancements in the area of neural networks acceleration and miniaturization have allowed deployment on relatively small embedded devices. This occurred along side rapid advancements in computation power of CPU, GPU, and FPGA-based platforms. Both kinds of advancements have coalesced into unique hardware and software accelerators for neural networks. For hardware acceleration, the creation of DPUs, TPUs, and Tensorcores allow for hardware-level improvements in most neural network architectures. In terms of software acceleration, TensorRT, Tensorflow Lite, and Alibaba's MNN toolkit

provide methods such as quantization and pruning. In terms of practicality, these edge-based neural networks offer rich, real-time analysis within a wide array of environments. One such example is using these edge devices for instant analysis of CTI, X-ray, and ultrasound scans [27]. Recent neural networks for detection and classification of abnormalities within these types of medical scans can potential be lifesaving. These applications need to be run on edge devices because these devices "reduce latency and keep traffic local and distributed" [27]. Additionally, edge devices can be deployed in a wider array of environments, including difficult weather or environments with a lack of connectivity [27]. This example and many others like it show that edge-based neural networks help an end-user accomplish their task quicker and more efficiently.

Similar to the case example, this thesis will focus on neural networks for various computer vision tasks, including object detection and multi-object tracking on edge devices. Additionally, analysis will be done on methods for each task regarding their robustness, learning ability, and run-time performance. The analysis of these different methods will accumulate towards a prototype device that will assist in a "vital sign" objective. The task of "vital sign" detection and estimation is part of an active research project at the University of Tennessee, Knoxville. The general goal is to be able to detect people in optical, infrared, and RADAR data and estimate vital signs such as heart rate, respiration rate, and oxygen saturation. All of this functionality must be accessible via a portable system that can be carried around by an end-user.

This particular system can be incredibly useful for many practical settings, such as search-and-rescue operations, medical monitoring, and military operations (Figure 1.1). There are many challenges to the system that will be tackled in this thesis. Such challenges include creating a real-time method of detecting people in optical and RADAR data, associating them through modalities, and optimizing each component in the system to run at real-time speeds. Topics such as RADAR data and vital sign estimation algorithms will be touched upon but only through the lens of acceleration and optimization for embedded platforms.

1.2 Pipeline of Thesis

The pipeline for this work is integral for creating a real-time application of detecting and estimating vitals of people with RADAR and optical imagery. The pipeline consists of two distinct phase. The first phase is the "Analysis Phase", where training, testing, and other optimizations are done to proposed methods. Additionally, this phase consists of analysing aspects of methods such as robustness, task performance, and speed in order to justify usage in the real-time system. The other phase is the "Application Phase", where the methods created in the Analysis Phase are implemented onto the chosen compute device. This involves utilizing different frameworks to further optimizer and convert. Additionally, construction of the prototype includes things such as crafting the 3D CAD enclosure. Other features, such as GUI-based and web-based streaming to other device, improve the quality of user interaction.



(a) Mass patient monitoring in triage centers can help prioritize treatment.



(b) Device can penetrate through material to find trapped individuals and aid in rescue prioritization.



(c) Identify individuals through dry-wall for military operations.

Figure 1.1: Different applications for a portable, real-time vital-sign estimation system. *The system's real-life applications include surveillance of large medical areas, as well as detecting life signatures in first responder and military operations.*

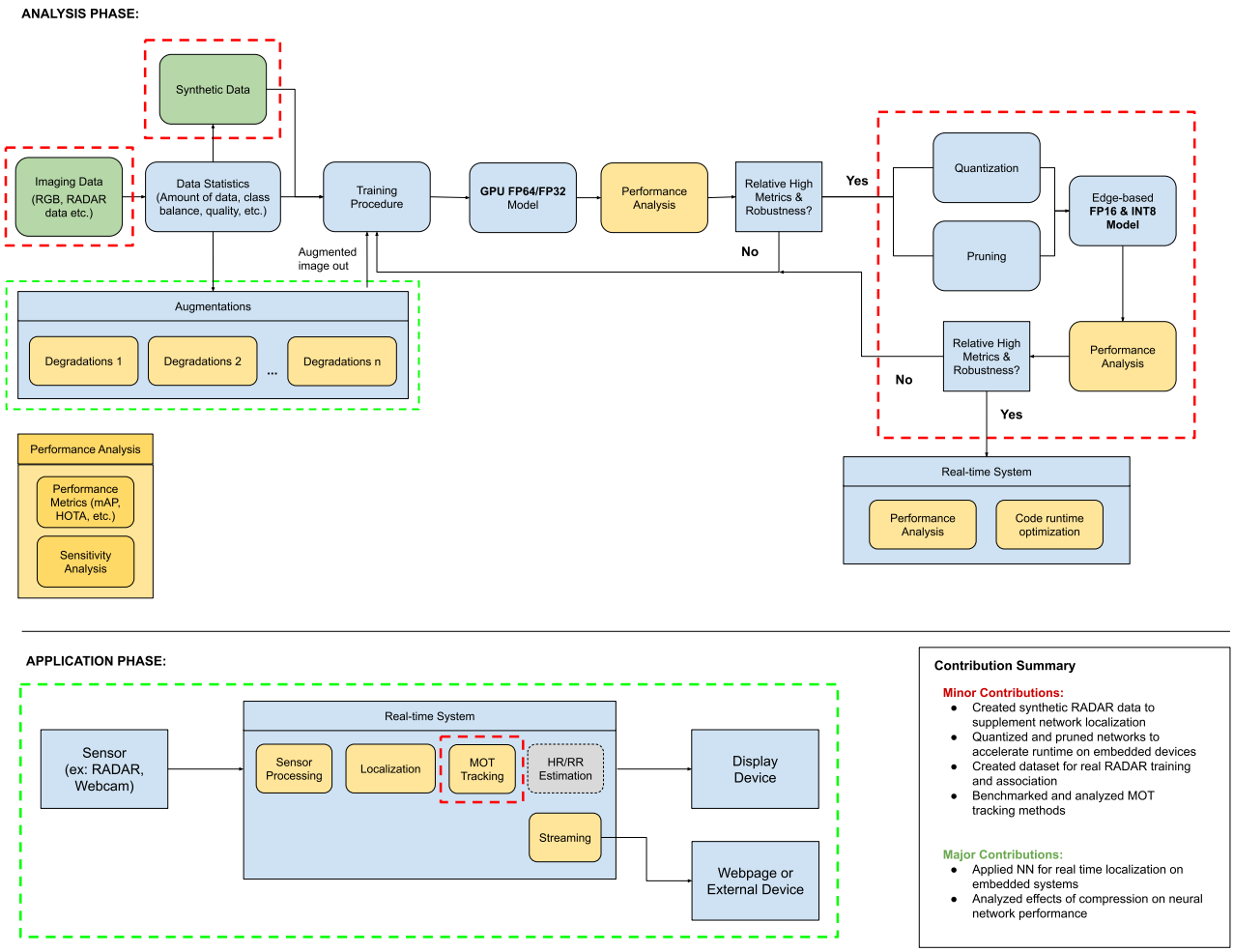


Figure 1.2: Thesis pipeline of training, testing, refining, and deploying various types of detection, vital sign estimation, and streaming methods for real-time application. *This diagram splits the thesis into two main phases - the "Analysis" and "Application" Phase. The former consists of developing methods (such as detection via neural networks) by training them against optical and non-optical datasets. Additionally, testing is done for baseline performance, but there is also "robustness" analysis where networks are subject to degraded inputs. This phase is cyclical in nature as we analyze new networks, optimize them, and re-analyze their performance. These types of analysis can give an insight into potential performances discrepancies when deployed in the field. In the "Application" Stage, we take the methods and lessons learned and applying them to a prototype system for vital sign estimation. This thesis analyzes and benchmarks system performance and adds additional functionality, such as GUI and web-based streaming.*

1.3 Thesis Contributions

There are six total contributions stated in this thesis. Based on their impact to the research project, depth of analysis, and novelty, they will be categorized into "major" or "minor" contributions.

Contribution 1 (Major)

One of the major contributions of this thesis is the application of neural networks for real time detection and localization on various embedded systems. Within this contribution, there are multiple analyses done. One such analysis is on the baseline performance of YOLO-based networks on COCO val2017 with additional measurement of inference speeds across different GPU platforms. From this benchmark, we see newer networks such as YOLOv4, YOLOv4-CSP and YOLOv5L achieving very comparable performances. YOLOv4 achieves the highest mAP@0.5 and mAP@[0.5:0.95:0.05] with 68.21% and 49.41% respectively, but both YOLOv4 CSP and YOLOv5L contain less parameters and run faster on all GPU-based systems. In terms of analysis of inference speed measurements, there is a trend between the speed-up magnitude of the discrete GPU and the Jetson Xavier devices. For sub-10 million parameter networks, the discrete GPU is 6.79x and 7.86x faster when compared to the Xavier AGX and NX respectively. For networks with over 10 million parameters, we only see a 3.94x and 6.94x speed up on the discrete GPU when compared to the Xavier AGX and NX respectively. This trend is not seen to this extent when comparing the AGX to the NX. Sub-10 million networks were on average 1.20x times faster on the AGX, and for larger networks, were on average 1.76x times faster on the AGX. Regardless, all sub-10 million parameter networks had real-time inference speeds on the Xavier AGX and NX. This makes ones with relatively high mAP scores realistic targets for a real-time system.

Another analysis that falls under this contribution is taking the networks from the COCO val2017 evaluation and retraining them on real and synthetic RADAR datasets. For synthetic RADAR training, all networks were set with roughly the same hyperparameters and tested against a Minimum Variance Distortionless Response (MVDR) and Range-Velocity datasets. From this training and evaluation, YOLOv5L performs the best with an average mAP score of 78.1% and 87.11% on the MVDR and Range-Velocity datasets respectively. When comparing performance amongst sub-10 million parameter networks, the leading network depends on the dataset. For MVDR-based data, YOLOv4-Tiny performed the best with an mAP of 67.73%. For the Range-Velocity data, YOLOv3 Tiny yielded an mAP of 84.15%. With real RADAR data, only the sub 10-million parameter networks were trained again. Training occurred with all the same hyperparameters as the synthetic trainings excluding the epoch value. The networks were tested on only the MVDR-based dataset since the dataset itself was captured using MVDR, and the networks were trained with different input sizes as well. Comparatively to the synthetic RADAR, the real-time dataset increased network mAPs by an average of 19.25%, but YOLOv4-CSP continues to outpeform all other sub-10 million networks. Across the input size of 208, YOLOv4-CSP continues to lead with an mAP of 84.03%. At the smallest input size, YOLOv5n performs the best with an mAP of 82.39%.

Contribution 2 (Major)

Another major contribution is establishing a relationship between data augmentations and neural networks performance. More specifically, the contribution measures the negative mAP impact of different image and video compression methods on SoTA detectors trained on COCO 2017. Furthermore, the contribution is an extension of existing sensitivity analysis work done by IRIS students and other researchers. Additionally, possible future iterations of the vital-sign based system could pivot towards a system architecture that requires streaming compressed data.

The compression benchmarks for all image and video compression methods (excluding the Gaussian Entropy Network) produced a negative logarithmic trend in mAP. This can be described as the mAP performance holding steady for most of the lowest compression parameters until at a certain mid to high-range compression parameter, the mAP drops precipitously. The analysis also ran average file size reduction for each image compression method on the COCO val2017 dataset as well as the MOT17 training videos for the video compression methods. The Gaussian Entropy Network reduced the images the most, but the quantization of features map was not real-time. Both video compression methods showed similar file size reduction, but H265 produced higher image quality.

In terms of an optimal choice in compression, it is highly dependent on the streaming device's hardware and the characteristics of the data being streamed. For situations where whole images need to be transmitted, an embedded device with a mid-range CPU should utilize WebP because it is shown to reduce image size better than other compression methods while retaining the most mAP. Although, it is known that the WebP encoder is 6x slower than a JPEG encoder. If an embedded device does not contain enough compute power for WebP, JPEG compression should be used as it exhibits competitive file size reduction compared to WebP and acceptable mAP degradation before a quantization level of 20. For video compression, H264 and H265 show similar mAP degradation across all networks. So, there are two other attributes to consider when choosing between them: importance of image quality and hardware acceleration availability. If an embedded device has access to a hardware-based H265 encoder, then utilizing it to preserve mAP performance and image quality is optimal. Encoder availability is not always a given since H265 is a relatively new specification, and hardware encoders for H264 are far more ubiquitous. In the case of no H265 hardware encoding, utilizing H264 at a CRF of 30 will be optimal.

Contribution 3 (Minor)

The first minor contribution encompasses the work done on Synthetic RADAR data and its various iterations. The contribution implements a program that can generate data with different size and position characteristics. Additionally, training a neural network on the synthetic data showed the need for reflection-based perturbations that mimic the appearance of the existing MVDR or Range-Velocity RADAR data. The synthetic RADAR data goes through two iterations. The first one was based on various autonomous car tasks, where the sensor is in the bottom-center and objects near the top would be the farthest away. This

first iteration was trained on YOLOv3 and scored a mAP of 71.73% and 31.02% on Range-Velocity and MVDR testsets respectively. This disparity in mAP performance created a second iteration focused on emulating a mix of appearances shared between the two datasets. This second iteration contained parameters to create clips of RADAR signatures moving on a canvas. The new frames would also contain various perturbations within the scene, and the signature would have inconsistent reflections. Re-evaluating YOLOv3 with the same training parameters resulted in a mAP of 84.15% and 72.22% on Range-Velocity and MVDR datasets respectively.

Contribution 4 (Minor)

Another minor contribution is the optimization of various neural networks for run time acceleration on embedded devices. These optimizations are quantization of network weights into FLOAT16 and INT8 as well as unstructured and structured pruning for image classification and object detection. For the quantization analysis, the same networks used in Contribution 2 (YOLOv3 Ultralytics, YOLOX, DETR) are utilized here. Quantizing models to FP16 increased average inference speed by 3.27x on the Xavier AGX and 2.80x on the Xavier NX. Because the precision of the weights is still within range for both FLOAT32 and FLOAT16, there is no drop in mAP performance. For INT8 quantization, mAP performance drops on average by 4%, with YOLOv3 suffering the most at 7% mAP decrease. However, INT8 quantization results in an average speed up of 3.8x and 4.88x on the Xavier AGX and NX respectively when compared to FLOAT32. Between INT8 and FLOAT16, the average speed up was 1.35x and 1.48x on the AGX and NX respectively. The pruning analysis was done on an image classifier called ResNet50 and an object detector called YOLOv2.

For object detection, mAP evaluation was done on YOLOv2 as well as versions of it pruned to different sparsity levels of 66%, 71%, and 77%. These models were provided by Xilinx and were trained on VOC2007 and VOC2012. Across all sparsity levels, the mAP loss was minimal with the maximum mAP being 6%. Pruning on ResNet50 was done with the help of SparseML, a toolkit for pruning and quantizing models. ResNet50 was pruned in an unstructured and structured method with sparsity levels of 70% and 40% respectively. Unstructured pruning showed lower loss of accuracy (-3.33% from baseline) than structured pruning (-6.6%) at a higher level of sparsity. Additionally, we explore effects of weight file compression on both types of pruning methods. Due to higher sparsity target, the unstructured ResNet50 compresses to 25,000 bytes (70% savings) compared to structured ResNet50 compressing to 40,000 bytes (56% savings). Increasing GZ compression ratios can bring unstructured ResNet50 down further to 20,000 bytes. Although, it should be noted that unstructured pruning has little to no effect on run time when it comes to most GPUs, due to inefficient sparse matrix operations.

Contribution 5 (Minor)

Since there are no readily available RADAR, infrared, or optical datasets geared towards the vital estimation task, the 5th contribution encompasses the data collection tasks performed for RADAR detection and cross-modal association. RADAR datasets provided by Yao Yu

from MaXentric and Marvin Joshi from the University of Tennessee allowed us to conduct analyses, such as the synthetic data benchmarking and MOT tracking. This contribution also encompasses the involvement of the author in the annotation process of both datasets. Additionally, when the real-time system was at a mature phase, real-time collection and new annotation methods allowed for the collection more data. This contribution also encapsulates the real-time collection process as well as the autolabeling that was used to speed up the RADAR and optical annotations. Additionally, new annotations such as association across modalities future-proofs this dataset for additional research.

Contribution 6 (Minor)

In addition to Contribution 1, we addressed multi object tracking in the optical domain and applied the findings to the real-time system. Furthermore, the cross-modal association task is also tackled with an introductory method.

The MOT benchmarking included taking different tracking methods based on their unique attribute. These methods are IoU Tracker, SORT, DeepSORT, and BYTE. Their unique attributes are, in respective to the order, computational simplicity, state-based estimation, state-based estimation with neural network-derived features, and stated-based estimation with low-confidence thresholds. All trackers were benchmarked on MOT17 and MOT20 and utilize the same bounding boxes from YOLOX-X, which is a near-SOTA detector retrained on various pedestrian dataset. This detector was taken from the original BYTE implementation. In addition to HOTA benchmarking, we also benchmark the run time speeds of each tracker to establish feasibility in the real-time system.

From this optical benchmarking analysis, we found that BYTE worked the best because of its usage of low-confidence detections to reduce dropped tracks. DeepSORT's appearance vectors allowed it to perform notably better than SORT on MOT20. This can be due to MOT20's higher pedestrian density making simple position information not as reliable as a combination of position and appearances. Run time analysis on a traditional desktop CPU, Jetson Xavier AGX, and Jetson Xavier NX shows that traditional DeepSORT is not viable for embedded system working in high density environments (such as ones in MOT20). BYTE offers the most balance in terms of high HOTA performance and real-time speeds while IoUTracker could be feasible for low-density environments on a low-power device.

For the cross-modal association task, we utilize a simple support vector machine (SVM) to do binary classification against a complete bipartite graph. The two sets within the bipartite graph are detections from RADAR and RGB detections respectively. Cross validation is done on the SVM with a fold value of 10 and the validation accuracy was 96.26% and testing accuracy was 94.25%.

1.4 Synopsis

In Chapter 2, we summarize previous works in different topics area that prelude the features within the real-time system. This is to inspire implementations in later chapters as well as discover areas where the system itself can be expanded to. Firstly, works related to RADAR-based localization and classification networks as well as RADAR and RGB data fusion show solid accuracy and robustness. Furthermore, works related to tracking methods and real time, neural network-based detections methods give background information for later analyses utilizing these methods. Additionally, an explanation of the main metrics for those methods are stated to give the reading a better understanding of later analysis values. Hardware literature mainly focuses on popular, high-compute embedded devices, such as mobile GPU devices. We compare different SWaP characteristics of the Jetson Xavier devices as well as the Xilinx Ultrascale+ ZCU104 FPGA. Finally, established literature on CNN optimization and pruning are discussed to tie together the software methods and hardware platform choices.

In Chapter 3, we compare the choice of framing the vital sign system as an "edge-based" paradigm or an "IoT-based" paradigm. This comparison concludes with choosing an edged-based paradigm due to increasingly flexible and powerful embedded devices (such as the ones discussed in the previous chapters). Furthermore, this chapter provide the general layout of the real-time system as well as possible components that would fit that layout. In terms of these specific components, another debate regarding compute platform is discussed with the Jetson Xavier NX chosen as the primary compute platform. The main reason for the Xavier NX is a combination of low-profile size, relative ease of development when compared to the ZCU104, and reasonable power consumption.

In Chapter 4, we utilize different optical datasets and detection methods. Additionally, a mAP evaluation of different YOLO-based networks is done on the COCO 2017 validation dataset. The evaluation shows that similarly-sized networks above 10 million parameters achieve very similar mAP@0.5 results, but YOLOv4 showed the highest mAP performance of 68.21%. But, why do newer networks such as YOLOv4 CSP and YOLOv5L marginally under-perform against older networks. These works focused on scalability of their architecture beyond simply a "tiny" version of their network. All networks were also evaluated for their inference speeds on three separate GPU platforms - discrete, desktop GPU, Jetson Xavier AGX, and Jetson Xavier NX. This chapter also generates synthetic RADAR data by relying on appearances only. All aforementioned YOLO networks are retrained on synthetic data. Additionally, real MVDR-based RADAR data is collected for the sole purpose of localization, and the same YOLO networks are trained against this dataset. Both synthetic and real training produce networks with relatively high mAP performance, but predictably, the models trained on real data performed much better on the MVDR processing algorithm.

In Chapter 5, a relationship between compression methods and neural network performance is established. The reason behind this analysis is based on extending previous work discussed in Chapter 2 and in the context of a hypothetical IoT-based paradigm for the

system. This compression analysis consists of taking three networks with different network architectures and evaluating them on a dataset with different compression methods under different compression parameters. Ultimately, the analysis shows similar mAP performance degradation across all compressions and all networks (excluding Gaussian Entropy Network). For guidance on specific compression utilization and parameters, there many caveats for both image and video-based compression methods. For image-based methods, WebP at a quantization level of 50 will save you 95.7% of the original image matrix while mAP performance only degrades by a maximum of 5%. Albeit, WebP encoding is known to be relatively more computationally expensive than other similar methods. So, if a specific compute platform is not able to produce fast enough speeds with WebP, JPEG at a quantization level of 50 could yield similar mAP performance and size reduction but with worse image quality. For video compression, streaming-based systems should utilize H265 at a minimum CRF of 30 only if a H265 encoder is available. If not, utilizing H264 at the same CRF minimum would suffice but with potentially worse image quality.

In Chapter 6, some of the methods and techniques shown in previous chapters are employed in a real-time embedded system for vital sign estimation. Optimization methods for neural networks are also implemented onto different object detection and image classification networks. These optimization methods are quantization to FLOAT16 and INT8 as well as unstructured and structured pruning. The quantization process showed a run time acceleration of at least 2x for both FLOAT16 and INT8 networks. The two aforementioned pruning methods are analyzed to show their hypothetical advantages in terms of file size savings.

In Chapter 7, the findings and deliverables from previous chapters are reiterated. We restate the main objective and the importance of each finding in the pursuit of this objective. Finally, this thesis discusses future work that other individuals can do to further the system itself as well as the established methods, measurements, and optimizations.

Chapter 2

Literature Review

2.1 RADAR Detection

Previous works related to detecting people in RADAR modality can be broken down to two main approaches. Earlier works attempted to determine the location of objects within RADAR data via traditional machine learning methods, such regression and SVM-based methods, as well as purely-algorithmic constructions. Later works have used Deep Learning in the place of these algorithms for greater robustness and learned understanding of the given RADAR data. Such deep learning-based methods have focused on pose-estimation in through-wall RADAR scenarios as well as detection in 2D birds-eye-view RADAR scans. Although the pose-estimation methods are not the same task that this thesis focuses on, they require localization of subjects before predicting poses, so it can be beneficial to understand their approaches and shortcomings.

RF-Capture

In Fadel et. al, they propose an algorithmic pipeline of estimating human poses. Previous implementations of skeletal acquisition required sensors on subjects, which made them impractical in real-life scenarios. RF-Capture consisted of two main components - the "Coarse-to-Fine" algorithm and the "Motion-based Figure Capture" [1]. In the former process, a "coarse" resolution scan is done to identify areas in 3D space where large reflections occur. The antennas are then focused into that area and recursively produce a "finer" resolution scan to collect a person's reflection [1].

The "Motion-based Figure Capture" process uses consecutive RF snapshots and pieces together partial humans scans into a complete skeleton. More specifically, "[the method] operates by segmenting the reflections according to the reflecting body part, aligning them across snapshots while accounting for motion" [1]. Attributes, such as depth and person sway, were considered in this process since both significantly affect the resolution and the predictability of the reflections [1].

Additionally, extracted features from the compiled snapshots of a subject were also used to do person classification. This classification was done by reducing the features' dimensionality

with PCA and using an SVM for the classification task. Benchmarks of RF-Capture found that this SVM-based classification method had an accuracy of 95.7% (5 users present) and 88.2% (15 users present) and a skeletal movement accuracy of 99.13% when users were 3 meters away and 76.4% when users were 8 meters away. The skeletal movement was computed by measuring the predicted skeletal output of RF-Capture at every frame with the skeletal capture from a Microsoft Kinect sensor.

Although, it is important to keep in mind that there a few limitations in regards to this method. The first is that the pipeline assumes that the users start by walking toward the T-shaped antennas. Without this assumption, the "Motion-based Figure Capture" cannot execute compensations for depth and sway properly. Additionally, the dataset used to compute the classification and skeleton tracking accuracy is limited in the total amount of recorded "clips" of individuals walking. Finally, the SVM-based human classification method is not online-based. It must be retrained against new data for new persons to be classified.

RF-Pose

Zhao et. al introduces a Deep Learning-based approach to estimating human poses with through-wall RADAR signals. More specifically, they introduce a neural network system called RF-Pose that can parse wireless signals and estimate 2D human poses in fully or semi-occluded optical scenes [59]. Similar to RF-Capture, RF-Pose also attempts to do person classification from the estimated poses. A major contribution from this paper is outlined in Figure 2.2. The RF-based pose estimation networks utilizes a cross-modal (data from multiple types of mediums) training system to help increase its performance [59].

This training system follows a "teacher-student" (also called distillation learning) learning paradigm. This paradigm contains a "teacher", which is a pose estimation network trained on optical images, and a "student", which is the RF-based pose estimation networks. More specifically, the RF-based pose estimator receives features from two encoder networks, each responsible for extracting features out of the horizontal and vertical RF frames respectively. It is important to note, like RF-Capture, RF-Pose also considers multiple "frames" (called "snapshots" in RF-Capture) to counter-act the issue of inconsistent reflections [59]. The estimated pose from the RF-Pose network is compared to the output of the optical-based pose estimator. The loss objective is to minimize the difference between the student's RADAR-based output ($S(R)$) and the teacher image-based output ($T(I)$). Zhao et. al summarizes the training loss as the following:

$$min \sum_{(I,R)} L(T(I), S(R)) \tag{2.1}$$

The expansion of $L(T(I), S(R))$ is described as the binary-cross entropy of the two pose confidence maps:

$$L(T(I), S(R)) = \sum_{(c)} \sum_{(i,j)} S^c_{i,j} log(T^c_{i,j} + (1 - S^c_{i,j}) log(1 - T^c_{i,j}) \tag{2.2}$$

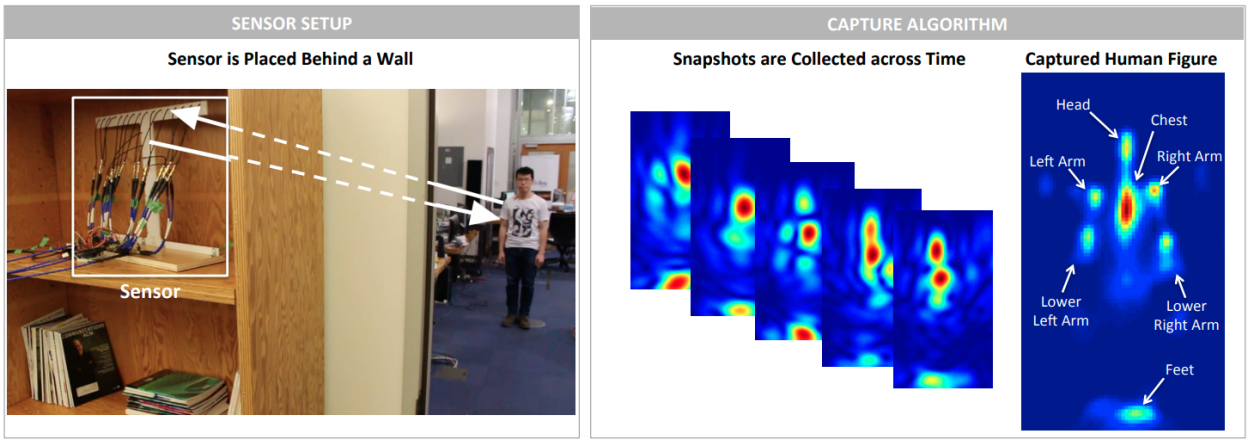


Figure 2.1: Figure 1 from [1] shows an illustration of the hardware and general software process for RF-Capture. *The hardware is composed of two antenna arrays shaped in a "T" arrangement. The "T" arrangement is composed of vertical antenna transmitters and horizontal receivers. In terms of analyzing and producing a pose estimation from the antenna array, RF-Capture first runs a "Coarse-to-Fine" algorithm that generates a "coarse" RF scan, which is used to localize and help retrieve a higher resolution scan. The higher resolution scans are stacked together over time, and this allows for body segmentation. An SVM was also used to do subject classification on the capture human figure.*

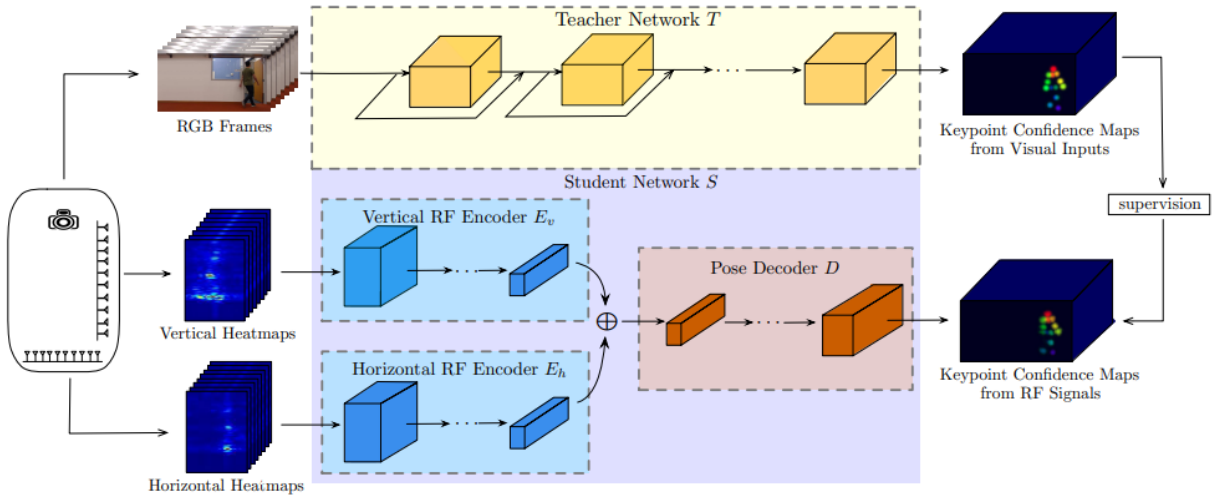


Figure 2.2: Illustration of the cross-modal "teacher-student" training process. *RF-Pose utilizes a well-established RGB-based pose estimation network as a "teacher" network. The RF-based pose estimation network is a "student" network, which will train on the RADAR data while receiving loss data information from the teacher network. The idea is to transfer knowledge of an RGB-derived pose to produce an RF-derived equivalent pose.*

where i, j are the coordinates of the pose confidence maps and c is the current corresponding confidence maps.

For person classification, Zhao et. al utilizes each pose's segments as features, which are then fed into a "vanilla" CNN classifier. This classifier is composed of 10 convolutional layers. The assumption of this training method is that the CNN will be able to discern individuals based on their pose's height, strut, and other pose-based information.

In terms of performance, RF-Pose was trained and benchmarked on an internal dataset composed of RF and corresponding optical-based recordings of 50 different environments. Each environment contained one or more persons doing activities such as walking, jogging, sitting, reading, and etc. 70% of this dataset is used for training while the other 30% is used for testing. Training and testing do not share any of the environments. On this dataset, RF-Pose is able to achieve a mean average precision (mAP) of 58.1% for pose estimation. Obviously, an equivalent optical-based pose estimator achieve an mAP of 0% in the fully-occluded image scenes. This mAP computation is based on the COCO mAP computations of thresholds from 0.5 to 0.95 in steps of 0.05. Furthermore, RF-Pose reports the AP_{50} and AP_{75} by taking the average of the OKS of each keypoint at that given threshold. In terms of performance, RF-Pose performs relatively well in both visible and obstructed scenarios, with an AP_50 of 93.3% and 85.0% in visible and obstructed scenarios respectively [59]. Additionally, for person classification, the classification CNN is able to achieve 84.4% and 83.4% accuracy in RF and optical scenes respectively.

Many-to-Many Encoder

Similar to RF-Capture and RF-Pose, Meng et. al produces a 2D pose estimation output from a RADAR device. In this particular work, a FMCW RADAR device is used for through-wall readings. Meng et. al asserts that observations of RF show that current RF imaging is not a "one-to-one" scenario. In terms of tasks related to pose estimation, critical limb components will not always show up in one "frame" of an RF reading. RF-Capture and RF-Pose also acknowledged this as well but attempted to compensate by only accumulating RADAR signal over time. Meng et. al utilizes an Long-short Term Memory (LSTM) network to mitigate this issue. The LSTM network fit in a four-component pipeline detail in 2.3. It takes the RPN output at the current frame and interpolates the final pose estimation based on the relationships it sees between limb localization at frame N and limb localizations at $N-1$, $N-2$, $N-3$, ... $N-K$, where K is the size of the LSTM's memory.

Again, similar to RF-Capture RF-Pose, [36] produced their own datasets from their FCMW RADAR and measured the OKS_50 and OKS_75 for the series of poses in a certain clip. OKS (Object Keypoint Similarity) is typically used for measuring the performance of pose estimation tasks. Unlike RF-Pose, Meng et. al does not provide the AP_50 and AP_75 of the OKS but does provide the OKS for each keypoint. The LSTM-based pipeline shows high degree of accuracy on both the optical-based data and RF-based data with an average OKS_{50} of 81% and 97% in obstructed and visible scenes, respectively.

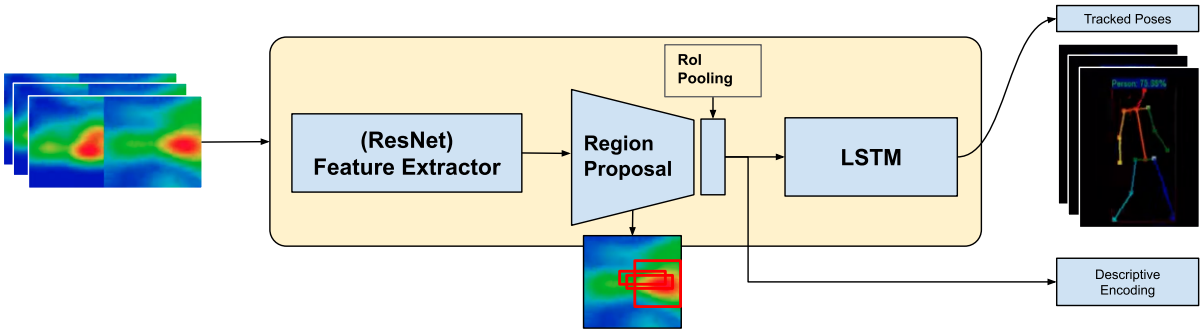


Figure 2.3: Network pipeline illustration redesigned from Figure 6 in [36]. *In this pipeline, RF readings from the FMCW RADAR are put through a feature extractor (ResNet50), and the feature maps are then processed through a Regional Proposal Network (RPN) to generate limb localization. The RPN output is then processed through an LSTM to produce the final pose estimation output. Additionally, a "descriptive encoding" is produced from the RoI pooling, which can be used for pose classification in multi-subject scenarios.*

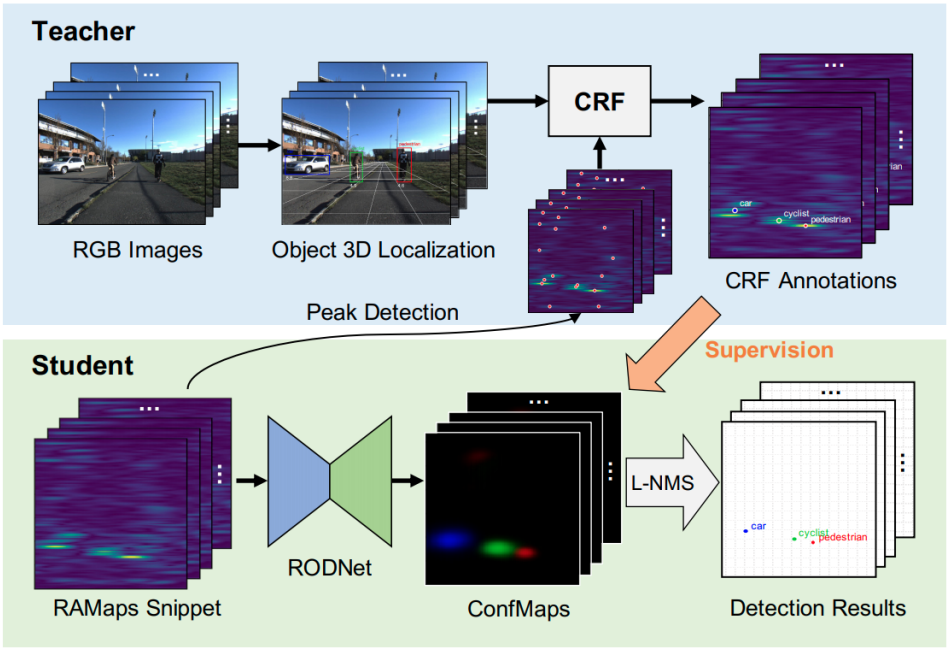


Figure 2.4: Overview of RODNet's cross-modal training structure. *This diagram was taken from the original paper [54]. RODNet takes advantage of a Camera-Radar-Fusion technique that utilizes an optical 3D detection method and CRF for RADAR peak detection. These two inputs generate automatic annotations, which are then fed into the RODNet's training cycle. RODNet comes in three distinct architectures with varying AP and FPS performance. All networks take in clips of RAMap data to produce their confidence maps.*

RODNet

Wang et al. introduce a neural network that can output a 2D confidence map of different objects in a Range and Azimuth output (RAMap) from an FMCW radar [54]. This network follows other previously mentioned networks by integrating cross-modality supervision in its training pipeline. A large dataset is collected of simultaneous optical car dash-cam data and FMCW RADAR data. The dataset is then used to train two main networks. The optical-based network is trained on 3D localization and is the "teacher". Consequently, the RF-based network is the "student" (Figure 2.4). But, unlike previously discussed works, the teacher network is only responsible for generating RADAR annotations. The annotations are created by generating possible locations in the RAMap with an adaptive peak detection algorithm called CFAR. The locations are then refined with the z-axis information from the teacher network [54]. With the cleaned RADAR annotations, the student network is responsible for taking in a set of RAMaps and generating a set of probability maps [54].

2.2 Detection and Tracking Methods

2.2.1 Detection Methods

The object detection task is well-researched domain in computer vision and encompasses a large number of tasks, such as face detection, salient object detection, and pedestrian detection [24]. The task itself can be described as "detecting instances of objects of a certain class" and is visualized in Figure 2.5 [24]. In the recent decade, Convolutional Neural Networks (CNNs) have advanced the speed, accuracy, and robustness of object detectors [24]. The main advantage of CNNs for object detection is the fact that they can learn relevant features from a set of images and generalize what they have learned onto other, unseen images. For object detection specifically, there are two typical setups - "two-stage" and "one-stage" detectors [24].

Both kinds of networks are composed of a "feature-extraction" network component and the "bounding box" network component. The former network is "acting as the basic feature extractor for object detection task which takes images as input and outputs feature maps of the corresponding input image" [24]. "Two-stage" detectors, such as Faster R-CNN (Faster Regional-CNN), split the feature extraction output into a "Region of Interest" network and a RoI Pooling Layer. The Regional Proposal Network (RPN) attempts to predict bounding boxes and classes for each bounding box. The output of RPN is filtered with the RoI Pooling layer, which also uses the original feature maps, to output the final predicted bounding boxes and classes.

For "one-stage" detectors, the process is similar but the RPN is removed. Instead, the feature maps are directly used in a RoI Pooling-like step to generate the final prediction boxes and classes. Typically, "one-stage" detectors are faster than "two-stage" detectors because of the lack of a RPN but can have lower performance metrics. One popular set of "one-stage" detectors are the "You Only Look Once" detection family. The original YOLO network was an early "one-stage" detector and was able to score a relatively high mAP score on the

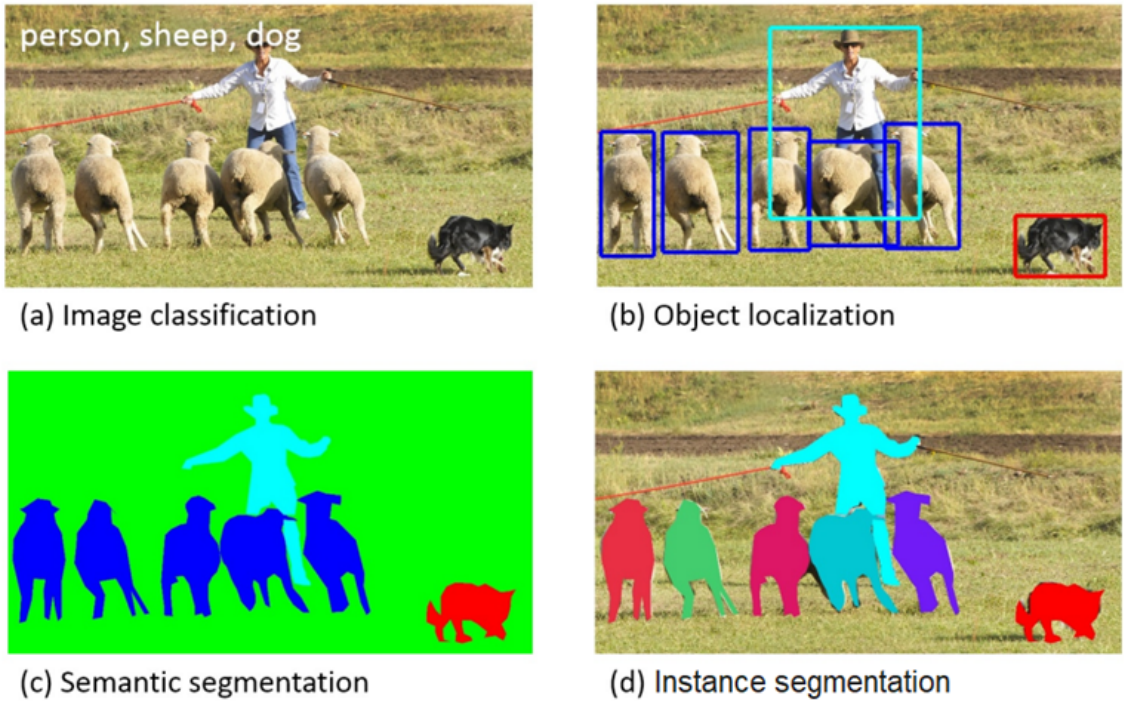


Figure 2.5: Common computer tasks and their differences against object detection. *The diagram is adapted from Figure 1 in the COCO dataset paper [32]. Object detection is conceptually similar to instance segmentation, but it differs in the coarseness of the localization. Rather a pixel-level mask, it requires a "bounding box" that represents both the location of an object and the classification of the object.*

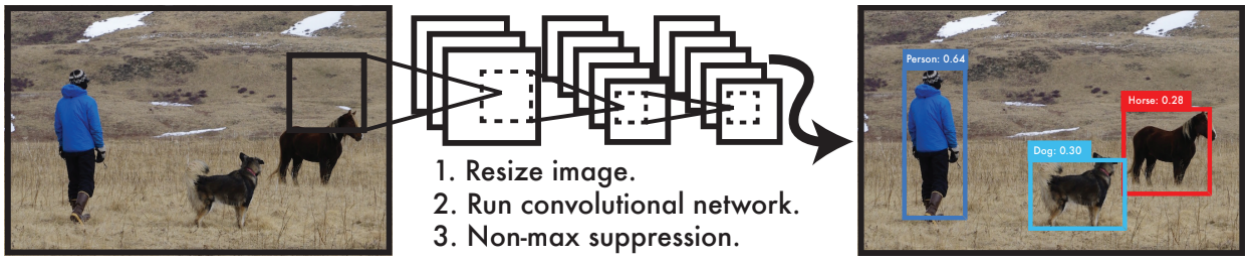


Figure 2.6: Structure of a YOLO CNN object detector. *This diagram is taken from [24]. It shows the general process of a one-stage detector. An image is resized into the CNN's input shape and put through a detector. The resulting bounding boxes are filtered with an algorithm called non-max suppression (NMS). As the the input is passed and processed through the network layers, the intermediate values generated are intended to represent certain characteristics of a given image, whether that is the shapes, texture, or spatial relationship of the objects in the scene.*

COCO and VOC2007+2012 test sets while maintaining "real-time" speeds [46]. It also was composed of relatively simple operations, which lends itself to relatively easy re-calibration and high performance efficiency [2].

The original YOLO network was composed of a modified GoogLeNet CNN as a feature extractor [46]. Bounding box regression was accomplished by splitting the image into a grid and have the network train on a "multi-objective" loss function for location (coordinates of the center of a given box, width, and height) and classification probability. The classification probability is based on the product of the class probabilities and the probability an object is within the given grid cell (also called "objectness") [46].

Future YOLO-based networks improve upon both the feature extraction, bounding box regression, and training techniques. For example, YOLOv2 increased network input size from 224x224 to 448x448 and retrained the feature extractor on ImageNet for classification [47]. Afterwards, the network was finetuned for detection on the VOC and COCO datasets, and all these changes increased the mAP score on VOC2007 by 4% [47]. Another example is YOLOv3 reframing of the classification loss from a mean-squared form to categorical cross-entropy [48]. One example is YOLOv3 replacing the existing feature extractor with a ResNet-inspired network called Darknet53 [48]. Similar to YOLOv2, YOLOv3 experimented with an increase to its input size to 608x608 [48]. The development and improvements of networks upon the YOLO-paradigm have also increased the mAP performance metric on the COCO 2014 and 2017 dataset. Refer to Table 3 to see a compilation of the mAP scores on their respective dataset (all metrics taken from [46], [47], [48])

Finally, the most recent developments in YOLO-based networks has been a divergence from traditional residual blocks that created the majority of the feature extractor. Rather, developments from other detection methods, bled into architectures of YOLOv4 and Scaled-YOLOv4. Particularly, the integration of Cross-Stage Partial (CSP) blocks enabled YOLOv4 and Scaled-YOLOv4 to be near SOTA for real detectors on the COCO benchmark [6] [53]. The Feature Pyramid Network (FPN), which connected the feature extractor to the detection head, was also replaced with a version of the Path Aggregating Network (PAN) in YOLOv4, Scaled-YOLOv4, YOLOv5, and YOLOX [6] [53] [25] [20]. YOLOX itself removed the anchor-based approach that was central to one-stage detectors since the introduction of YOLO. Instead of one set of feature maps representing detections, confidences, and classifications, YOLOX outputs all in seperate branches and directly without any anchor boxes [20].

Apart from YOLO-based networks, a new neural network structure called the Transformer network has shown competitive performance when compared against traditional CNN-based methods [8]. A transformer-based object detector called DETR (DEtection TRansformer) introduces the architecture to object detection by framing the problem as a "direct set prediction problem" [2]. This means the transformer utilizes its self-attention mechanism in order to generate unique box predictions [2]. Unique predictions mean utilizing post-processing techniques, such as non-maximum suppression, less than other detection methods.

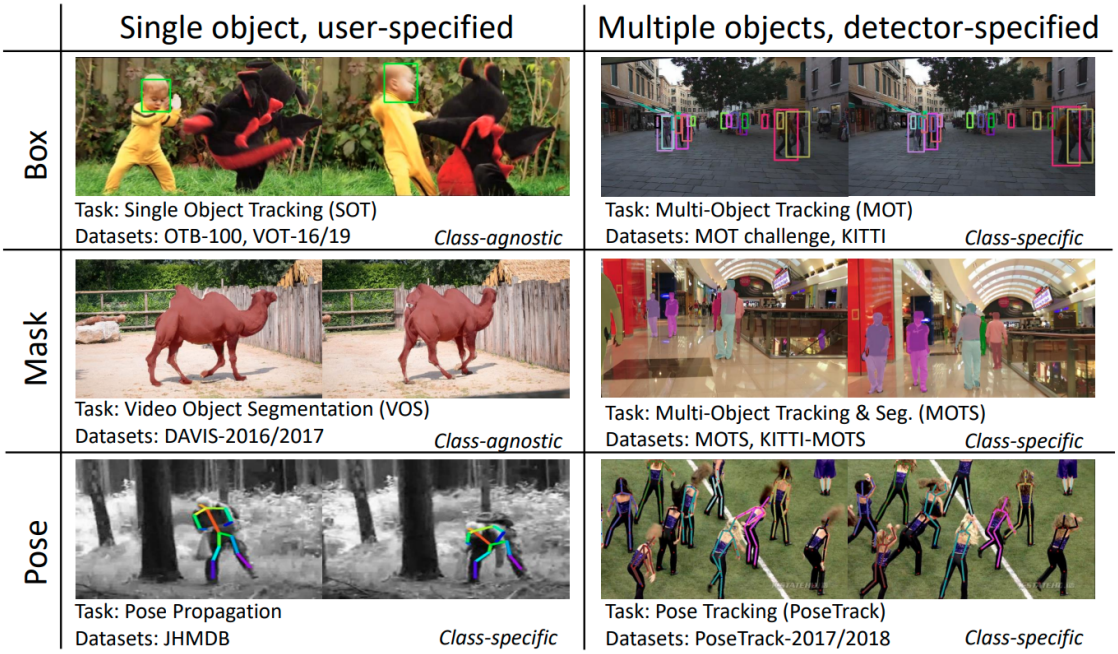


Figure 2.7: List of different types of tracking. *This figure is taken from UniTrack [55] and shows Single Object Tracking (SOT) and Multi-object Tracking (MOT). Both are bounding-box-based tracking tasks, but there are analogous tracking tasks for both segmentation and pose-estimation outputs [55].*

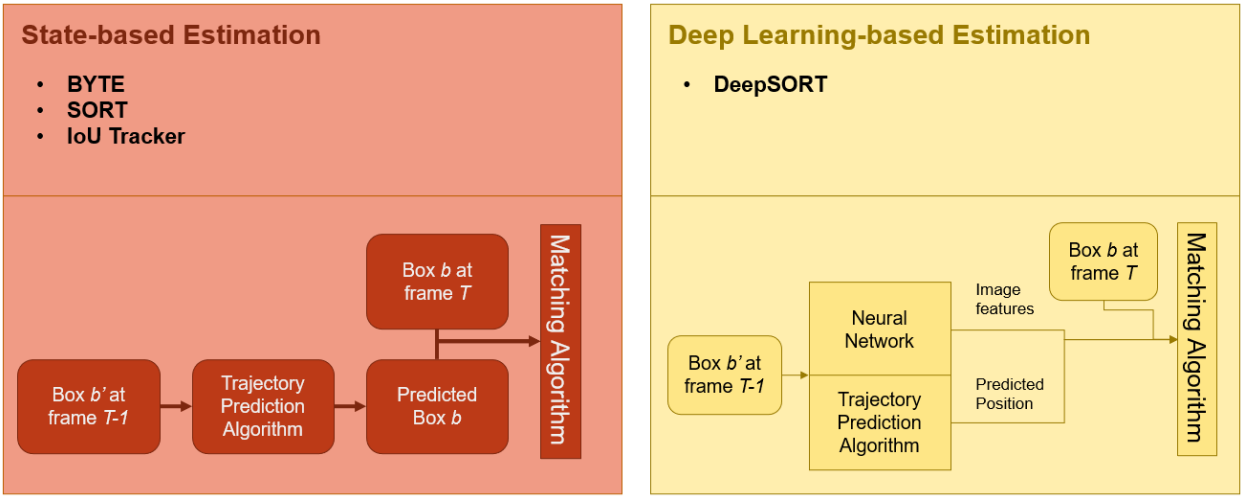


Figure 2.8: Diagram of categorized tracker methods discussed in this work. *A caveat in this categorization is that the IoUTracker does not have a predictive function.*

2.2.2 Tracking Methods

"Tracking" in computer vision is a domain that encompasses many different tasks. Such tasks include Single Object Tracking (SOT) and Multi-object Tracking (MOT) [10]. Both SOT and MOT are bounding-box based tasks, but there are other sub-tasks illustrated in Figure 2.7.

In SOT, a singular object has its initial position (typically defined by another method or user-input) propagated by an SOT method into future occurring frames [10]. Contrastly, in MOT, bounding box information is provided for each frame by a detection method. Then, a given MOT method is used for association of active tracks that have propagated to the current frame. Due to MOT's associative aspect, MOT methods would help differentiate individuals for further processing and functionality.

MOT methods have also progressed from state-based algorithms to Deep Learning-based networks in attempt to improve re-identification of subjects. In this work, a similar range of real-time MOT methods are evaluated both on optical and non-optical data to see their overall metric-based performance as well as their impact on overall object detection accuracy within a video clip.

IoU Tracker

This tracking method is a simple method that develops tracks based on the spatial overlap. This spatial overlap is defined as the intersection over union (IoU) and is a ratio that is defined in Equation 2.3.

$$IOU(a,b) = \frac{Area(a) \cup Area(b)}{Area(a) \cap Area(b)} \tag{2.3}$$

If an IoU between an unassociated box and an "active" association track is greater than a certain threshold, that box will be added into that track. Association tracks that have not had a match in a certain number of frames are removed from consideration.

SORT

Simple Online Real-time Tracking (SORT) is an algorithm that is based on a Kalman filter and Hungarian matching algorithm to determine track associations. This approach allows SORT to run at real-time speeds. SORT receives bounding boxes from a CNN Object Detector and computes a predictive box from each Kalman filter [5]. This predictive box is then matched to any of the CNN-produced bounding boxes with the Hungarian matching algorithm.

DeepSORT

Although SORT achieved "higher than average" performance when paired with a state-of-the-art (SOTA) detector, it had a tendency for extensive "identity-switching". "Identity-switching" (also referred to as ID switching) is typically defined as when an active trajectory

changes from its corresponding ground truth identity [29]. This is attributed to the fact that the only aspect of the scene that is taken into account is the bounding box locations. This is unadvantageous when camera stability is not guaranteed and there is a large shift in bounding box positions between frames [56].

To alleviate this problem, DeepSORT incorporated "appearance" descriptors from a neural network as another feature in SORT. The neural network in the original paper is a simple 10-layer CNN. This CNN took in a cropped image of a given object and outputs a 128-dimensional feature vector that is the "appearance" descriptor of that object. From there, a two-step metric for final track association or creation is considered. The first step is original SORT, which considers the current bounding box locations and the projected position from a track's given Kalman Filter.

The Mahanlobis distance is calculated between the projection and current boxes, and under a high threshold (95% reported in the paper), a likelihood score is assigned to each track for that given bounding box [56]. Additionally, active tracks' previous appearance vectors are compared to current appearance vectors based on their cosine similarity. Both metrics are combined as a weighted sum with a hyperparameter λ to determine the final association of a bounding box.

BYTE

Providing bounding boxes from an object detector to processes, such as IoUTracker or SORT, usually entails providing only the boxes with a relatively high confidence threshold. Unfortunately, this filtering of low-confidence detections can lead to loss of tracking for occluded objects. Zhang et. al introduces a real-time method called BYTE that considers low-confidence object detection output.

BYTE sorts bounding boxes into "high" and "low" confidence groups. The high-confidence bounding boxes are matched with tracks based on a Kalman Filter and Hungarian Algorithm (similar to SORT) [58]. Low confidence bounding boxes are then matched with the remaining tracklets using IoU as a similarity metric [58]. This algorithm is fairly simple, and so, it can be adapted into other frameworks. For example, DeepSORT with BYTE would replace the high-confidence matching metric from IoU to an appearance-based metric [58].

Zhang et. al constructs a simple real-time pipline called ByteTracker, which is composed of YOLOX and BYTE as detector and tracker, respectively. With YOLOX trained on the CrowdHuman dataset, ByteTrack is able to achieve SOTA performance on MOT17 and MOT20. On MOT17, it achieved an MOTA of 80.3 and HOTA of 63.1, and on MOT20, it achieved an MOTA of 77.8 and a HOTA of 61.3 [58].

2.2.3 Cross-Modal Association

Works such as RF-Pose and Many-to-Many encoder utilized network training with cross-modal supervision in order to produce highly accurate RF-based networks for localization

or pose estimation. There are other works that try to utilize multiple modalities as inputs, rather than just training components. Vi-Fi from Liu et al. showcases a system that can associate individuals with a given smartphone via optical data and a smartphone's Fine Timing Measurement (FTM) and Inertial Measurement Unit (MTU) data (Figure 2.9). They create a deep-learning method and compare it against a bipartite matching method, which acts as a baseline. The bipartite matching method utilizes "motion profiles" generated by the main camera as well as the FTM and MTU data [33]. This motion profile contains attributes such as trajectory and distance from initial position, which is generated from a variety of different processes depending on the modality.

The bipartite process matches the subjects in the camera and the smartphones from the WiFi modality based the similarity index of the different motion profiles. The neural network introduced consists of a bi-directional LSTM that takes in multiple inputs from the optical domain (localization and depth estimation) and WiFi domain (FTM and IMU measurements). This LSTM generates an embedding for each domain's input at every frame [33]. A large feature matrix is assembled that consists of every combination of optical and WiFi domain embeddings for every domain. This feature matrix is processed through several convolutional layers to produce a 2D affinity matrix, where each element is the probability that a certain subject in the optical domain is associated with a certain smartphone in the WiFi domain [33]. Overall, the online classification accuracy of the deep learning method is 81% [33]. The bipartite matching algorithm achieved a classification accuracy of 50%.

2.3 Software

2.3.1 CNN Quantization

The process of quantization in neural networks can be summarized as the reduction of the datatype precision for each neuron's weight. Typically, networks are trained with a relatively high-precision datatype such as 32-bit floating point (FLOAT32) numbers. Quantization techniques can reduce FLOAT32 to 16-bit floating point (FLOAT16), 8-bit integer (INT8), and 2-bit integer (BINARY). Regardless of the target datatype, the reduction in precision can drastically increase the inference speeds of a given network. For example, [64] showed that ResNet50 can be quantized to INT8 while retaining the baseline accuracy. Furthermore, the same network quantized to BINARY was within 3% of the baseline.

Typically, quantization can be broken into two separate tasks - "Post-Training Quantization" (PTQ) and "Quantization-Aware Training" (QAT). PTQ takes a trained network at a high-precision datatype and recalculates the weight ranges to a lower-precision datatype while preserving network output and performance. This method is fairly easy, computationally light, and relatively low in hyperparameter tuning [38]. Contrastly, QAT methods retrain a given network while "simulating" quantization constraints. QAT is more involved in terms of development time and computation, but because you are actively training the network against quantization-related performance loss, there will be less of a performance gap between lower and higher precision versions of a network.

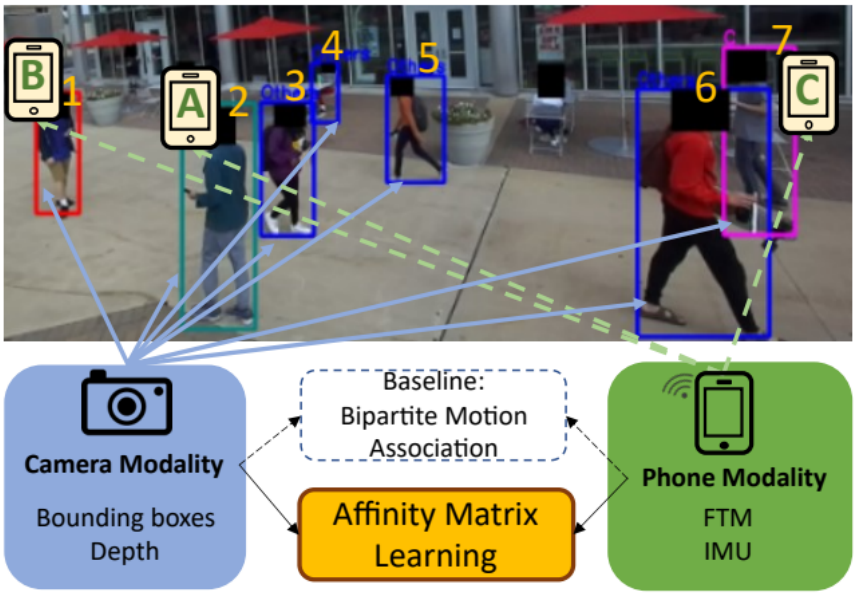
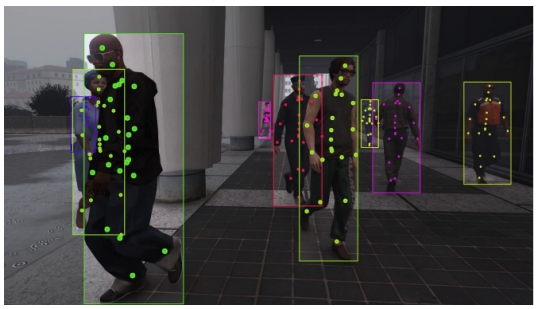
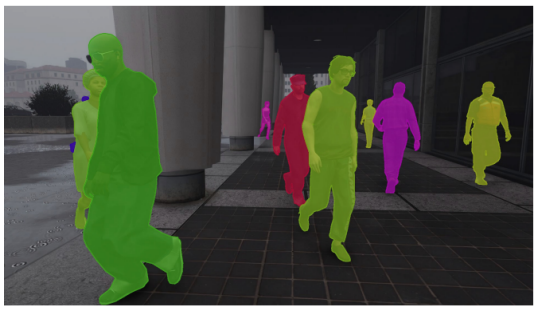


Figure 2.9: General overview of ViFi's modalities, task, and proposed solution. *The diagram is Figure 2 from the original paper [33]. The deep-learning-based solution exhaustively matches features from person's optical data to a phone's MTU and FTM data. They utilize a LSTM to retain memory from previous frames to improve accuracy.*



(a) Bounding Box and Joint Annotation



(b) Semantic Segmentation Annotation



(c) Depth Mask Annotation

Figure 2.10: Illustration of MOTSynth's different annotations output [16]. *These annotations are achieved by utilizing the in-game scripting language of Grand Theft Auto V.*

2.3.2 CNN Pruning

Similar to quantization, pruning is an acceleration technique that can find a particular set of neurons that do not contribute to the ultimate output. Removal of these neurons results in a "sub" neural network that can have increased inference speeds as well as be trained for a shorter time. Additionally, removing these neurons can reduce the filesize of the network weights, which can help in rapid deployment and over-the-air (OTA) updates to edge-based devices. For example, [19] showed a method that could remove 90% of neurons from ResNet50 and retained 98% of the original baseline accuracy.

There are multiple pruning methods that offer varying degrees of sparsity and improvements to inference speeds. These methods are categorized under "structured" and "unstructured" method. "Unstructured" pruning involves removing neurons when a given pruning method has marked them for deletion under a certain condition. Likewise, structured pruning removes groups of neurons that are related to one-another, such as channels or filters in convolutional layers.

Under unstructured pruning, run time improvement is not typically feasible because the result of removing individual neurons from a given layer will result in a sparse matrix. Sparse matrix calculations are typically not faster than traditional matrix calculations in GPUs (an exception is the NVIDIA GPUs Amphere architecture) [17]. Structured pruning has the ability to decrease inference time since removal of groups of neurons, such as channels and filters in convolutions, means a reformatted layer in the network structure [30]. The downside is that structured pruning tends to produce less sparsity [39].

There are many different types of pruning methods, but [19] has shown that Gradual Magnitude Pruning (GMP) remains one of the most effective and simplest methods of pruning - particularly with Convolutional and Fully Connected layers. GMP works by putting a pretrained network through an additional training loop, and under a given range of epochs, the weights equal or close to zero ("close" is a hyperparameter) at the end of each of each epoch are removed [23].

2.3.3 Synthetic Data for Training and Evaluation

The acceleration of performance and usage of Deep Learning-based methods has been in part due to the large, diverse range of datasets. Datasets, such as ImageNet and COCO, contain thousands of different images and hundreds of thousands of human-annotated labels. This data collection, organization, and annotation can take many years and a large amount of human work. There is also the added public concern of citizen privacy when it comes to face detection and recognition in-wild datasets [16]. These concerns manifest themselves in bills such as the General Data Protection Regulations (GDPR) in the European Union [16]. Because of these reasons, many works attempt to create "synthetic" data to supplement or replace real-world data.

One area for synthetic data usage is multi-object tracking. MOTSynth [16] and MOTX [34] are both works that attempt to create fully synthetic dataset of dense, wide-ranging pedestrian environments. Both works utilize game engines to create both the physical environment as well as scripting non-playable character (NPC) behaviors. This enabled them to create these environments in real-time while maintaining a level of graphical realism.

MOTSynth utilizes a popular multiplayer game called "Grand Theft Auto V" (GTA-V), which contains a large, "open-world" setting with a diverse set of NPC character models. With the built-in game engine scripting program, MOTSynth automatically annotates characters with bounding boxes, instance segmentation masks, joint positions, and depth maps. Figure 2.10 visually illustrates the different annotations.

MOTX creates a dense pedestrian environment by utilizing the Unity 3D Game engine. Using custom character models and the pathfinding mechanism in Unity, MOTX also asserts that using the synthetic dataset on MOT-based tracking networks is equivalent to using the training set of real world data, such as MOT17 and MOT20. Both works show that synthetic data can replace or contribute heavily towards network learning and generalization.

2.4 Evaluation Metrics for Detection and Tracking

Mean Average Precision

Mean Average Precision (mAP) is the primary metric used in Object Detection tasks and is employed by dataset challenges like VOC2007, VOC2012, and COCO. MAP is, by its name, the mean of the Average Precision (AP) of each class within the detection task. In turn, the AP per class is computed by the area under the curve (AUC) of a Precision-Recall graph. Precision and recall are defined in Equation 2.4 and Equation 2.5. The definitions for True Positive (TP), False Positive (FP), and False Negative (FN) are as follows. TP is defined as the number of ground truth and detection box pair that has a intersection over union (IoU) greater than a given threshold. FP is defined the number of unpaired detection boxes, and similarly, FN is defined as unpaired ground truth boxes.

$$Precision = \frac{TP}{TP + FP} \tag{2.4}$$

Note: TP+FP can be interpreted as the total number of detections.

$$Recall = \frac{TP}{TP + FN} \tag{2.5}$$

Note: TP+FN can be interpreted as the total number of ground truths.
Each precision and recall value is computed at a range of confidence thresholds to produce a Precision and Recall graph. The AUC of a PR graph differs between dataset challenges. For example, VOC2007 computes its AUC by taking the Riemann Sum with 11 interpolation points. COCO computes AUC by taking 101 interpolation points. Additionally, COCO takes multiple mAP scores at different IoU threshold ranges - from 0.5 to 0.95 in intervals

of 0.05.

Multiple Object Tracking Accuracy

The main metric used for MOT-based evaluation is Multiple Object Tracking Accuracy (MOTA) [37]. MOTA is part of a larger evaluation kit called CLEAR and combines three separate errors related to tracking - (FN_t), (FP_t), and ID Switching ($IDSW_t$) where t is the current frame. [51]. These errors are used in Equation 2.6 to generate the MOTA score.

$$MOTA = 1 - \frac{\sum_t (FN_t + FP_t + IDSW_t)}{\sum_t GT_t} \tag{2.6}$$

High Order Tracking Accuracy

If a tracker produces enough of the errors in Equation 2.6, then the MOTA score can actually be negative [37]. This and other issues regarding over representing detection accuracy leads to MOTA being an undescriptive metric for tracking performance and behavior [51].

HOTA (Higher Order Tracking Accuracy) attempts to rectify MOTA's issues. HOTA works in two distinctive phases - Detection and Association [51]. For the detection-matching phase, "True Positive" (TP) is defined as a pair of detection and ground truth boxes that have the highest "similarity index" that is above the similarity threshold (defined as α). For bounding box-based MOT tasks, the similarity index is the IoU threshold. "False Positive" (FP) is defined as a ground truth box that has not been matched to a detection box. Likewise, "False Negative" (FN) is defined as a detection box that has not been matched to a ground truth box.

For situations where multiple detection boxes could match with different ground truths and still satisfy the similarity conditions, HOTA runs different combinations to determine the final optimal HOTA score, which in turn is the optimal detection-matching results. For the association-matching phase, HOTA defines "TP Association" (TPA), "FP Association" (FPA), and "FN Association" (FNA) as per set of TP calculated in the detection-phase. A set of TPAs is equal to a set of TPs that have the same track ID between each detection as well as the same track ID between each ground truth. A set of FPAs are equal to a set of TPs with detection boxes of the same track IDs, **but** the corresponding ground truth track IDs are different. Similarly, a set of FNAs are equal to a set of TPs with the same ground truth track IDs, **but** the corresponding detections have different track IDs.

With these definitions, the final HOTA score is computed in two steps. The first step is the calculation of HOTA at different α ranges. The HOTA at α is simply the square root of the sum of the association-phase function (defined in Equation 2.8) over the sum of the detection-phase results (defined in Equation 2.7). The next step is calculating the area under the curve (AUC) formed by plotting the HOTA_α against the α values (defined in Equation 2.9).

$$HOTA_\alpha = \sqrt{\frac{\sum_{c \in TP_{set}} A(c)}{|TP| + |FN| + |FP|}} \tag{2.7}$$

$$A(c) = \frac{|TPA(c)|}{|TPA(c)| + |FPA(c)| + |FNA(c)|} \tag{2.8}$$

$$\int_1^0 HOTA_\alpha, d\alpha \tag{2.9}$$

Because of the way each component of the final HOTA number is computed, it can be decomposed into five different statistics that described their respective portions of the detection and association ability of the tracker (and to an extent, the detector). These five statistics are described in Figure 2.11.

2.5 Previous Sensitivity Analysis Work

Work done by previous IRIS Lab students have shown that different types of noises, blurs, and other image augmentations can drastically affect the performance of CNNs. Clay Leach and David Cornett's work on FPGA-based CNNs showed that augmentations, such as Salt & Pepper and Gaussian Noise, dropped CNN mAP performance by as much as 30% while retaining a relatively high similarity to the original image (similarity score computed with SSIM) [28] [11]. Both works explored fast, image processing-based corrections to these augmentations and found that mAP recovery was effective at only specific correction parameters. For example, a median filter with a kernel size of 3x3 applied to Salt & Pepper images recovered a maximum of 39.96% mAP on YOLOv3 [11]. Furthermore, Cornett also showed that similar degradation in SSIM scores closely correlated with mAP degradation across classification and detection networks. Matthew Seals also showed similar trends with additional augmentations such as image resizing and brightness, but methods, such as sharpness, showed very little effect on CNN performance [50].

Furthermore, Poyser et. al also analyzes lossy image and video compression on various neural networks for object detection, depth estimation, and pose estimation [45]. Their results show that lossy image compression methods, such as JPEG compression, do not affect mAP performance until at a relatively high compression level. Additionally, they apply H264 encoding to various video datasets and find that encodings with a Constant Rate Factor (CRF) of greater than 40 significantly affect the performance of all aforementioned networks.

2.6 Hardware

This section will discuss various compute platforms and their unique configuration. With this configuration, specific devices of a given platform will be briefly discussed.

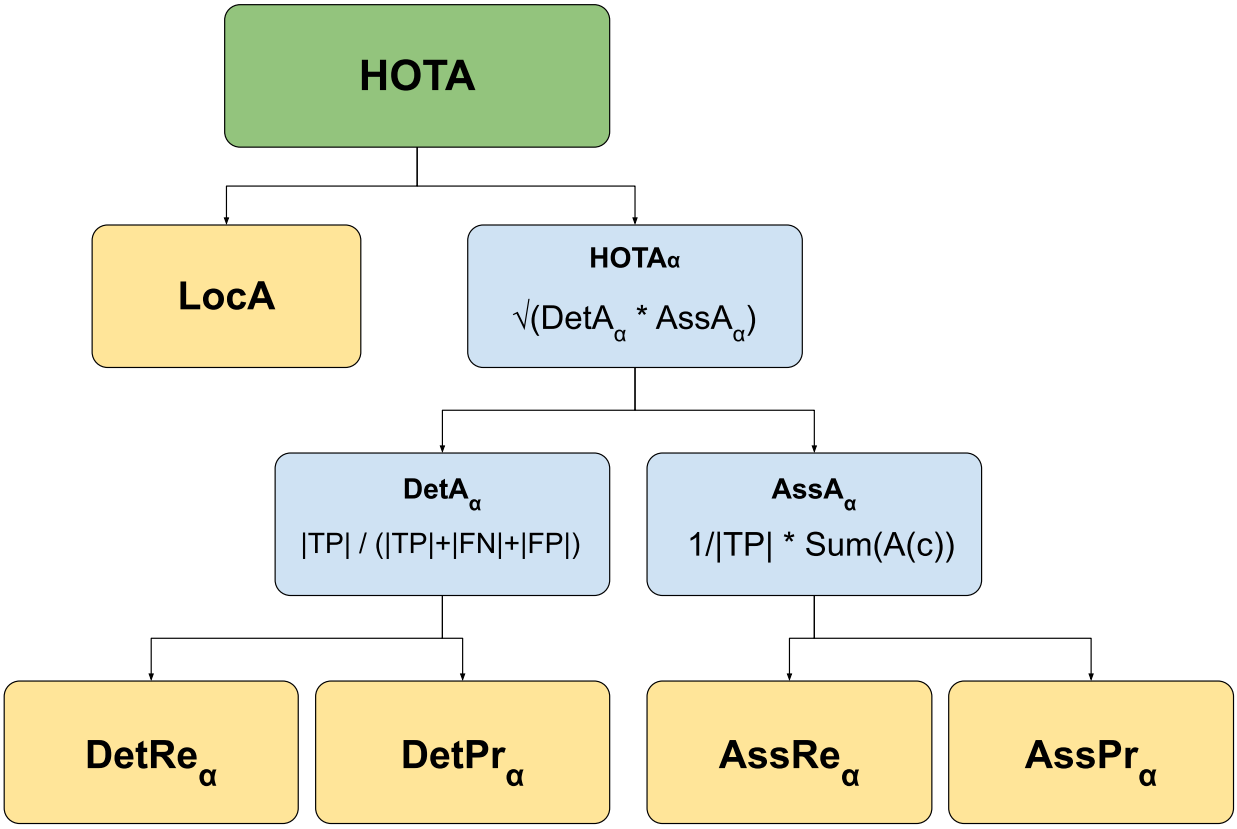
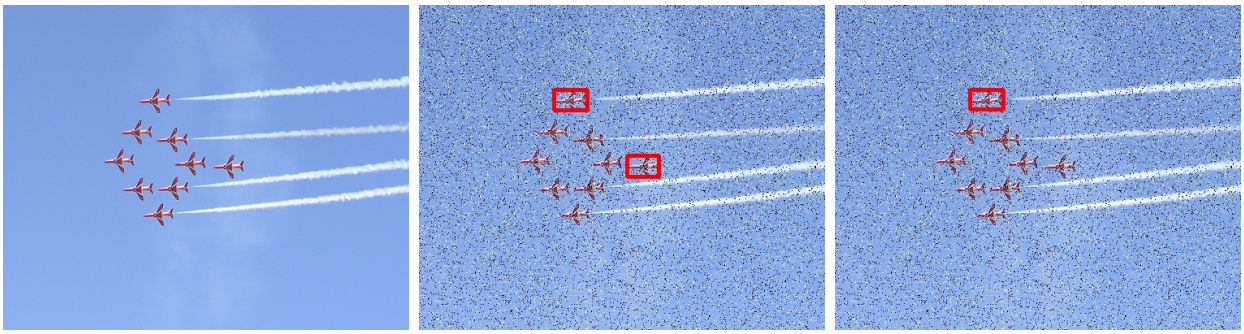


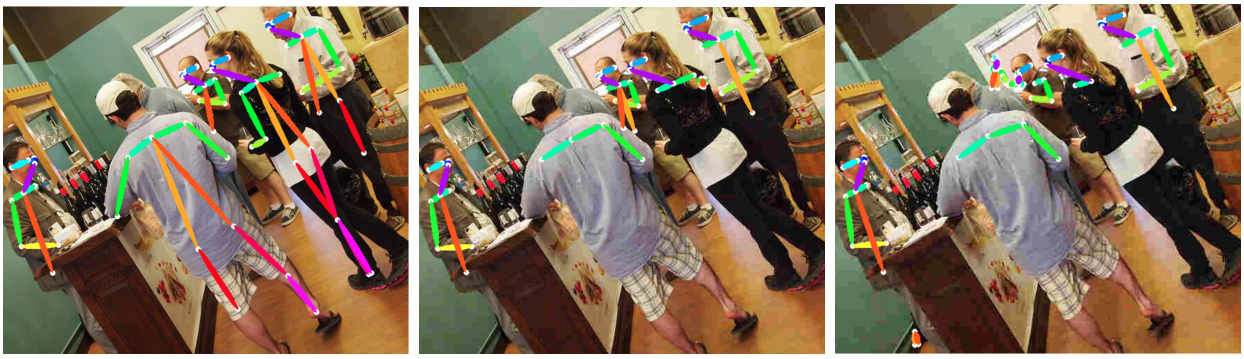
Figure 2.11: Decomposition structure of HOTA metric. *This diagram is inspired by Figure 3 from [51], and describes the "modularity" of the HOTA metric. We calculate detection accuracy (DetA) and then calculate the association accuracy (AssA) with regards to the sets of true positives from DetA. Finally, we can calculate a secondary localization accuracy metric from HOTA from the composition of DetA. The localization metric tells us how well the tracking method is able to place a bounding box over a given object.*



(a) Differences in GPU (center) and FPGA (right) detections



(b) Applied augmentation (salt & pepper) and a correction method (median filter) to recover image features



(c) JPEG compression reducing pose estimation model's accuracy

Figure 2.12: Examples of augmentations and corrections applied to images and detection result differences across hardware platforms. *Because of limited operations on the Xilinx FPGA platform chosen in [11], the implemented networks are slightly modified and can behave differently under certain augmentations. Additionally, correction methods for each augmentation chosen in Leach and Cornett's work show effective and fast methods of image reconstruction with noise and blur-based augmentations. Finally, networks of all types can suffer from degrading augmentations, such as pose and dept estimation network.*

2.6.1 Embedded Mobile GPU

NVIDIA Jetson

The NVIDIA Jetson platform is a group of hardware devices that are primarily aimed to run in "edge environments". For example, Jetson devices have had success in the mining sector, where a Jetson TX2 processed and streamed data from a UAV for 3D reconstruction [35]. This reconstruction would be later used to avoid potential construction planning mistakes. The Jetson devices is essentially a Single Board Computer (SBC) with a Socket on Chip (SoC) configuration. The specific characteristics of a Jetson SoC differ based on the type of Jetson device, but for this work, we focus on the Jetson Xavier line of products. The Xavier-based SoC contains an integrated Volta-based GPU and an ARM-based Carmel CPU [41]. Additionally, other components such as Ethernet interface and RAM are integrated within the SoC (Figure 2.13).

NVIDIA Jetson AGX

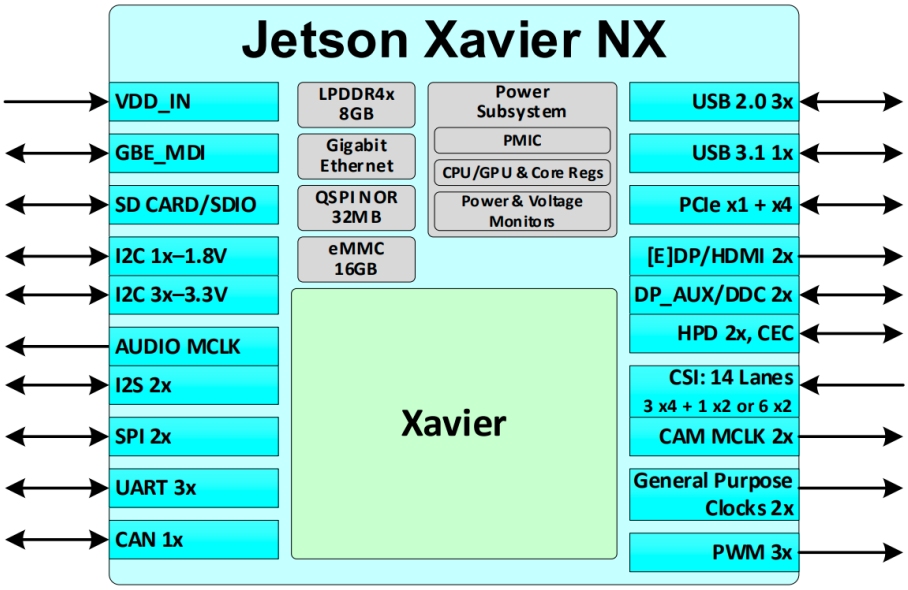
The NVIDIA Jetson AGX is composed of an 8-core Carmel ARM CPU at a maximum frequency of 2.26 GHz [40]. The GPU is based on the Volta architecture, which supports 64 Tensor cores for AI acceleration [40]. The AGX utilizes a "unified" memory structure, where the 16GB memory pool is shared between the CPU and GPU.

NVIDA Jetson NX

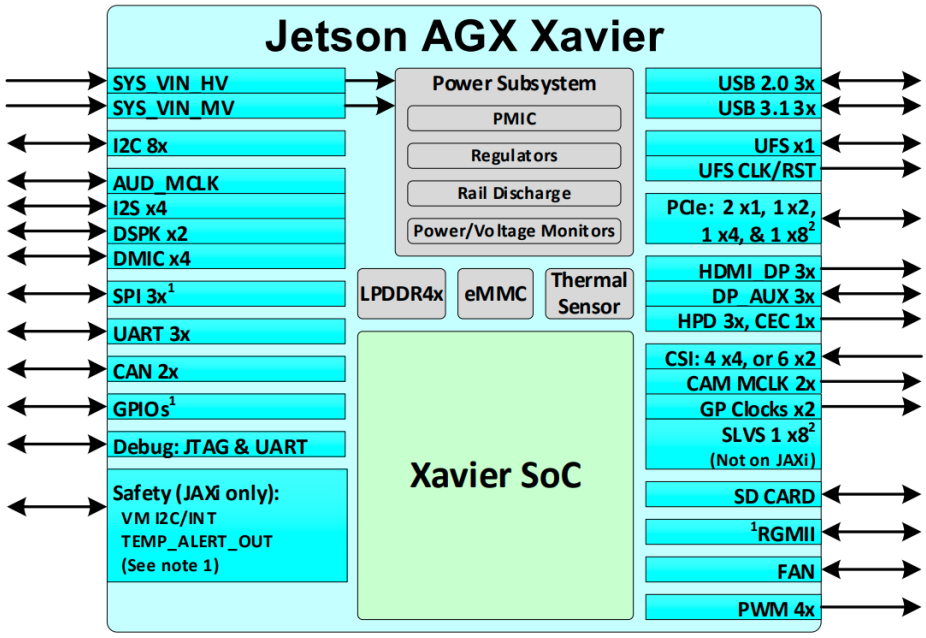
The NVIDIA Jetson NX is very similar to the AGX, but it is composed of a 6-core NVIDIA Carmel ARM processor at a maximum frequency of 1.9 GHz [41]. The GPU is based on the same Volta architecture as the AGX, but it contains only 384 CUDA cores and 48 Tensor cores [41]. The Xavier NX also contains a unified memory structure between the CPU and GPU but only contains 8 GB of that memory [41].

2.6.2 FPGA

As a possible alternative to a mobile or discrete GPU platform, FPGAs have been increasing in popularity in embedded design. FPGA designs are typically created with a language such as Verilog or HDL, and these languages help describe the structure and behavior of the FPGA fabric [11]. Once testing of this language design output is complete, it will be compiled into a bitstream that will be used to modify the FPGA circuit. This predefined, well-tested circuitry is the main reason why FPGAs are popular in embedded designs. Unlike a traditional CPU or GPU instruction set, the FPGA programmable fabric has a fixed computation route for one main primary process, and it can execute this process very quickly. Additionally, this programming logic also reduces overall power usage. Many FPGA platforms also have additional processing units and peripheral connections to allow for rich interactions and interfacing with more traditional processes. For example, the Xilinx Zynq contains a programmable logic with various low-power application processing units (APUs). These allow interfacing with custom peripherals (Ethernet, HDMI, USB, etc.) and accelerator components.



(a) Jetson NX Block Diagram



(b) Jetson AGX Block Diagram

Figure 2.13: Block Diagrams of Jetson NX and AGX. *Taken from NVIDIA Datasheets from the respective devices, and both block diagrams show the wide array of I/O opportunities for streaming in and out large amounts of data. For example, the Camera Serial Interface (CSI) ports on both devices enable for 1080p and higher resolution input from a camera, and USB 3.x and Gigabit Ethernet for high throughput video streaming. Their respective Volta-based GPUs and ARM-based CPUs deliver competitive SWaP (Size, Weight, and Power) characteristics compared to other embedded devices; especially for neural network applications.*

Xilinx UltraScale+ ZCU104

In addition to the SoC shown in the Jetson Xavier NX and AGX, this thesis also discusses aspects of a particular FPGA platform. This platform is the Xilinx UltraScale+ ZCU104, which is a mid-range FPGA capable of a wide number of operations. These operations are possible by the 7EV MPSoC (Multiprocessor SoC) that contains multiple ARM Cortex CPUs, Mali GPUs, and DPU for neural network hardware acceleration (Figure 2.14). The general process for FPGA development described in the previous paragraph follows for the Ultrascale+ lineup, and a developer can utilize Vivado to accelerate coding in HDL and testing it via simulation.

For machine learning-based processes, one can manually program things like convolutional, linear, and pooling operations within in HDL as well as the forward pass behavior and pre/post-processing. But, this is an extensive task, and an existing library called Vitis AI is used to help accelerate implementations of neural networks on the ZCU104. It provides pre-built neural network operations for the FPGA, access to the DPU, and a workflow for model conversion on Tensorflow and PyTorch. In addition to Vitis AI, there are other libraries that alleviate the work of HDL programming. This includes Xilinx's PYNQ platform, which utilizes prebuilt and custom bitstreams as "packages" that can be loaded into a Python interpreter. The PYNQ platform is primarily used in the ZYNQ series of FPGAs for education purposes. The ZYNQ series do not have comparable compute power to the ZCU104 and do not continue any deep learning execution ability.

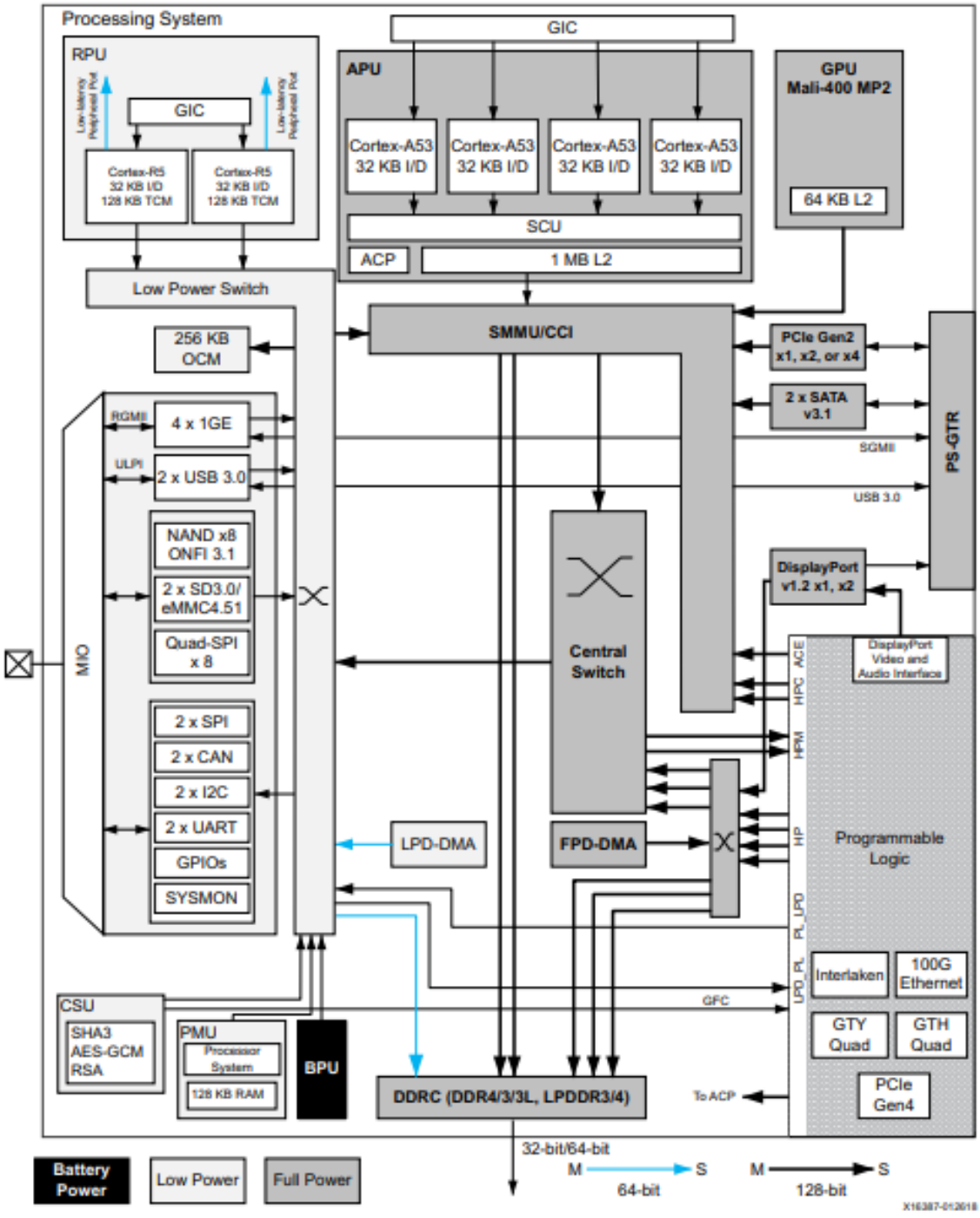


Figure 2.14: Block diagram of the 7EV MPSoC. *The MPSoC contains multiple ARM Cortex-based CPUs as well as a ARM Mali-based GPU. These enable the ZCU104 to achieve high throughput tasks, such as streaming multiple high-resolution videos at once. Additionally, a Data Processing Unit (DPU) enables fast neural network operations such as convolution, dense, and pooling operations.*

Chapter 3

System Layout

3.1 Edge vs IoT

An important factor to consider in Deep Learning-based embedded processing is the distinction between "edge" and "IoT" devices. An edge device is an extension of a traditional compute device, where the input is received and processed on the same device. The main distinction is that an "edge" device needs to do its computation in constrained SWaP environments [11]. Meanwhile, an IoT device's primary responsibility is to take sensor input, preprocess it (if applicable), and send it back to a larger, more traditional computing architecture (ex: cloud computing, traditional work station) for the data processing. The output from the traditional architecture is then streamed to different endpoints, including back to the IoT device [11]. Both types of device architectures are visualized in Figure 3.1.

Additionally, there is a significant difference in hardware and software structures. IoT devices typically contain very low compute power and are sometimes the sensor itself with networking capabilities (ex: ethernet-enabled cameras) [12]. Software-wise, IoT devices have little to no real operating system overhead and can only, in most circumstances, run a single-threaded program [12]. A good example of IoT devices are the Aurdino devices with Aurdino IoT Cloud support and the Raspberry Pi Pico W. For these example devices, the software development process involves coding a program on a host computer and flashing that program onto the processor's internal memory. In terms of edge device hardware, compute power is equatable to some modern low to medium end computers. Software-wise, the expanded hardware specifications allow for more traditional operating systems and code that can utilize multiple threads, cores, and different hardware components. Examples of edge devices include the aforementioned Xilinx ZCU104 UltraScale+ and NVIDIA Jetson series. In terms of software development process, these edge devices can be directly programmed on and interacted with without any flashing or host computer.

There are advantages and disadvantages to both types of devices. IoT devices can trade their low compute performance for higher power efficiency, lower heat, lower costs, and a smaller form-factor. While, edge devices can work irrespective of connection strength. Additionally, assuming the processing code is efficient, edge devices can have lower latency with higher

sensor data density compared to an IoT device. When it comes to building the real-time vital-sign system outlined in this thesis, it is important to consider the work requirements. As mentioned in Chapter 1, this system can be used in many different environments. This also means that the system needs to be flexible for many different types of users with their own unique set of requirements.

For example, utilizing this prototype in a search and rescue or military setting would make requirements such as low-latency and self-contained computation extremely important. Contrastly, in a healthcare setting, connectivity is more realistic and interactivity with other devices and servers is most likely a requirement. In recent years, small edge devices have had their capabilities rapidly expanded. Advancements in mobile, ARM-based CPUs, mainly driven by increased computational roles of smartphones, have turned devices such as the Raspberry Pi into viable personal computers while being no bigger than a pack of cigarettes. The same edge devices are now equipped with similar network connectivity options compared to traditional IoT devices. This makes edge-based devices applicable to far more environments, and because of this, this particular system will adopt an edge-based paradigm.

3.2 Sketch and Physical Layout

Designing any kind of system requires working under certain constraints. For this particular system, the end-user requires a hand-held device that can process sensor data and produce localization of a subject, classification of an object, and determine both the heart rate and respiratory rate in realtime (Figure 3.3). As with the majority of handheld devices, this particular system adopts a "smartphone"-like enclosure. This enables the device to be easily holdable and familiar to the end-user. The primary categories for each hardware component within the enclosure are the compute device, sensors, and a display output.

In terms of sensors, the RADAR data is produced from RFBeam's VDM3, which can sample a scene at a 100 FPS, contains steerable antennas, and is packaged in a small form-factor. Additionally, an Arducam IMX519 16MP Autofocus camera is used to record optical data. This camera offers video up to 1080p at 30 FPS and, as the name states, smartphone-like autofocus controlled via Arducam's open-source software. The display currently being used is Waveshare's 5.5 Inch AMOLED Capacitive Touch display. This display is produce in a convenient, pre-built form factor where the AMOLED and touch layers are already attached to a display board with display and touch I/O. The display is rated for a maximum resolution of 1080p and a pre-framed cover with "toughened" glass [14].

To determine the appropriate edge compute platform, different hardware platform's SWaP (Size, Weight, and Power) attributes were analyzed as well as factors regarding development time. The devices that will be considered are the Xilinx UltraScale+ ZCU104, NVIDIA Jetson Xavier AGX, and NVIDIA Jetson Xavier NX. The NVIDIA 2070 Super was also analyzed as a baseline comparison for high compute capabilities. The embedded devices are

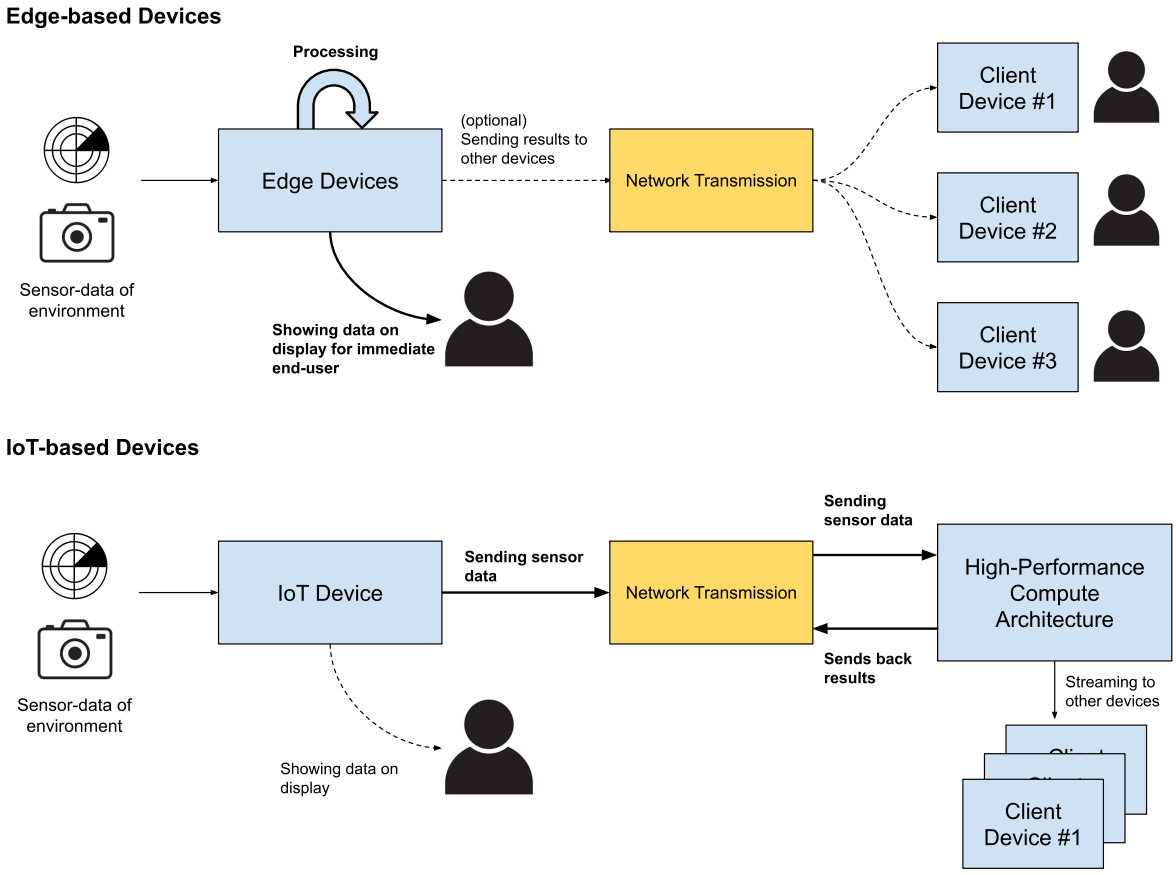
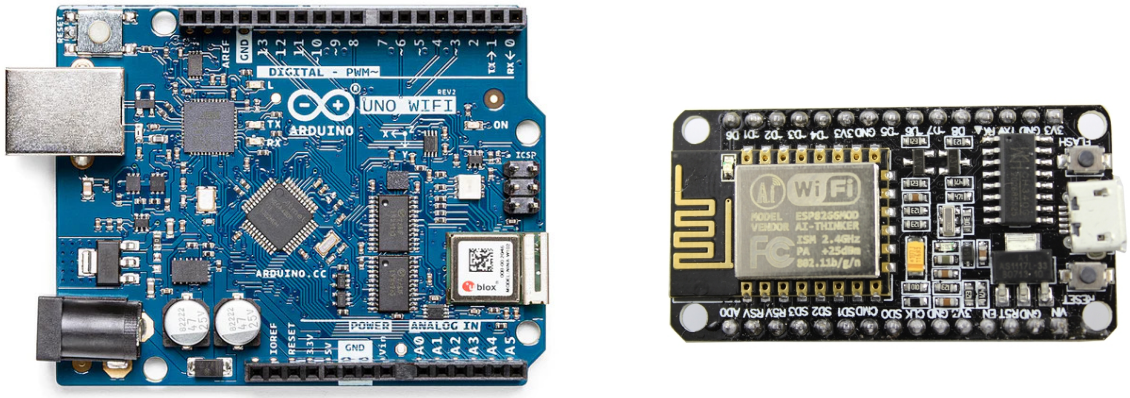
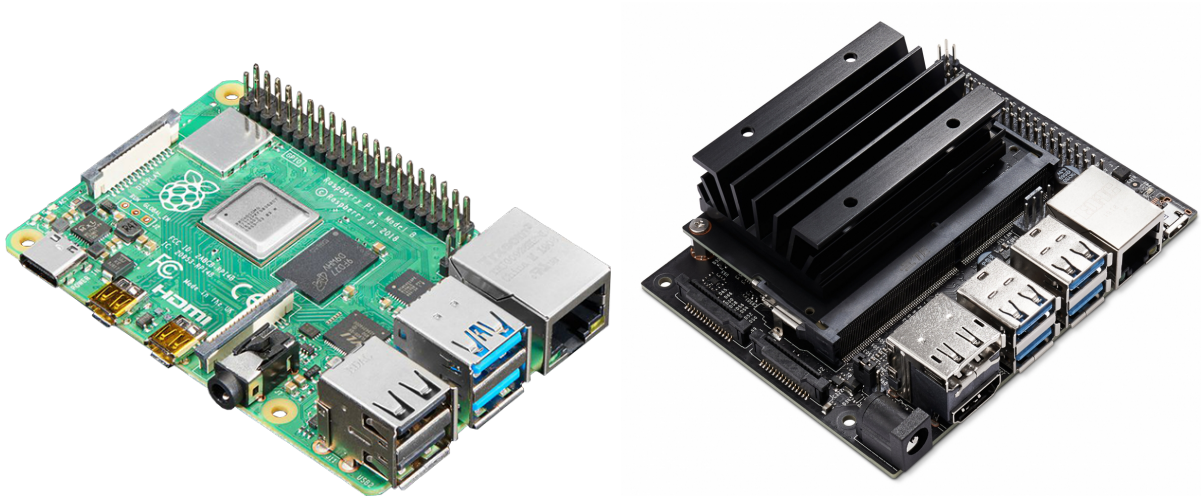


Figure 3.1: Illustration of the typical flow of data and processing edge-based devices and IoT-based devices. *For edge-based devices, the sensor data is processed on the device itself. Edge devices can also display the data to an immediate end-user as well as streaming data to a network of other end-users. IoT-based devices collected sensor data and transmit that data to a compute architecture. This architecture typically contains more compute power and processes the data quickly. The results can be distributed to other users and the IoT device itself.*



(a) Examples of IoT Devices - Arduino Uno Wifi (left) and NodeMCU (right)



(b) Example of Edge Devices - Raspberry Pi 4 (left) and Jetson Nano (right)

Figure 3.2: Examples of IoT and Edge Devices. *With IoT devices such as the Aurdino Uno Wifi and NodeMCU, the hardware consists of a main re-programmable microcontroller with low compute power. There are additional chips for connectivity, including low-power bluetooth and WiFi modules. Software-wise, they require a host computer to flash a program onto the microcontroller. IoT devices also very affordable, with both the Aurdino Uno WiFi and NodeMCU under $60 per unit. For edge devices, the Jetson Nano and Raspberry Pi 4 are traditional examples with far more capable and powerful ARM-based processors. They also have similar connectivity modules, but the Jetson devices can cost more than $200 per unit depending on the device type. The software development is similar to traditional development, where code can be assembled and executed on the device itself.*

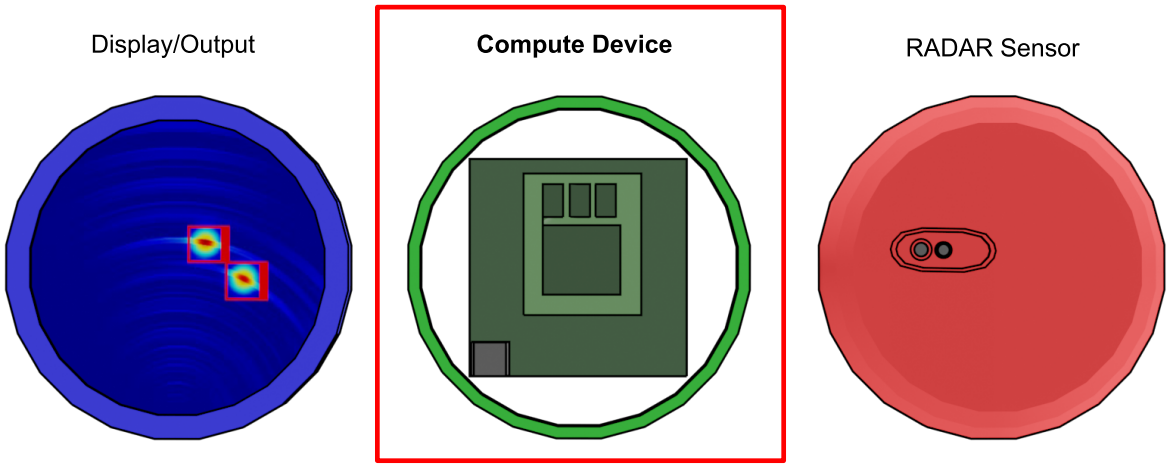


Figure 3.3: An initial sketch of the design of the integrated system for vital-sign system. *The compute device portion was the primary focus of this thesis as its purpose is to run sensor collection and processing code. Additionally, it was also responsible for tasks, such as detection and classification of individuals as well as regression for heart rate and respiratory rate. Similar to a typical smartphone design, the integrated system would consist of the compute device between the end-user display and the sensors.*

Hardware Platform	Size (w x l x h)	Weight (grams)	Maximum Power	Development Platform
NVIDIA 2070 Super	267mm x 116mm x 35mm	1100	215	Any (PyTorch, Tensorflow, etc.)
Trenz Electronic TE0807	72.2mm x 52mm x 3mm	42*	15**	Vitis AI
NVIDIA Jetson Xavier AGX	105mm x 105mm x 65mm	630	30	TensorRT
NVIDIA Jetson Xavier NX	45mm x 70mm x 65mm	172	20*	TensorRT

Table 3.1: Table comparison of different devices and their SWaP characteristics. *The requirements for the real-time system make a traditional desktop GPU infeasible due to its large size and extensive power draw. The same size requirements make the Jetson Xavier AGX impractical. The TE0807 and Jetson Xavier NX are better fits for this type of form factor.*

** - Weight does not include any sort of cooler.*

*** - Maximum power usage is dependent on the code running on the FPGA and the 15W is taken from [11].*

**** - 20W power options are available in JetPack 4.6 and newer. Older versions support 15W max.*

described in more detail in Figure 3.4. Table 3.1 details the SWaP characteristics of each specific device in Figure 3.4.

In terms of development cycles, FPGAs are known to be particularly difficult. Although, Xilinx's recent introduction of Vitis AI helps accelerate neural network deployment to their FPGAs by integrating existing DNN conversion toolkits, such as DNNDK (Deep Neural Network Development Kit), and FPGA runtime libraries, such as Xilinx Runtime (XRT). Furthermore, Video Vision Analytics SDK (VVAS) aims to provide easier integration between video I/O and Vitis AI.

Unfortunately, even with these SDKs, the development cycle with the Xilinx ZCU104 is still relatively longer than traditional ARM or x86-based devices. Additionally, the relatively smaller number of networks, layers, and operations supported by Vitis and difficulty of communicating with external devices makes an FPGA-based compute platform non-optimal. Both Jetson devices have a similar development cycle to traditional desktop systems, with a familiar Ubuntu-based operating system. Additionally, with a CUDA-capable GPU, Jetson devices can run the majority of neural networks (if the network can fit memory-wise) and can utilize a wider array of peripherals compared to the ZCU104. While the Jetson platform suffers in terms of power efficiency and speed (in some cases), their development flexibility and wider 1st and 3rd party support makes them the primary choice for this system. In terms of choosing between the Xavier AGX and Xavier NX, size constraints force the usage of the Xavier NX. In other circumstances, such as an IoT-based method, the Xavier AGX could serve as some kind of intermediate compute architecture receiving data from smaller, sensor devices.

For certain form factors, the Jetson Xavier NX "Devkit" (pictured in Figure 3.4) could be too big, so a potential workaround is utilizing the ConnectTech "Quark" NX. The "Quark" is a simple carrier board for the Xavier NX compute module and measures 82.6mm x 58.8mm [43]. Because the Quark is using the same module as the devkit, their performance and benchmarks are identical. The only difference is the change in I/O port availability. The Quark offers dual Giga-bit Ethernet, a GPIO interface, 2x MIPI CSI-2 lanes for camera input, and a USB 3.1 Type C interface for data communication[43]. Additionally, the Quark requires a BSP package from ConnectTech to be added onto the matching L4T installation. This makes the flashing and setup process more involved than the DevKit version.

There are additional carrier boards that have a similar footprint to the Quark. The NCB00 from DCDZ is a Jetson Xavier NX compatible carrier board aimed for UAV applications. Like the Quark, it is slightly bigger than the Xavier NX module itself. It measures in at 87mm x 50mm. Because of the UAV application, the NCB00 contains 5 separate CSI ports. Unlike the Quark, the NCB00 has a video output via a micro HDMI port. Additionally, the same Xavier NX module on the DevKit will work on the NCB00 and works without any BSP modification.

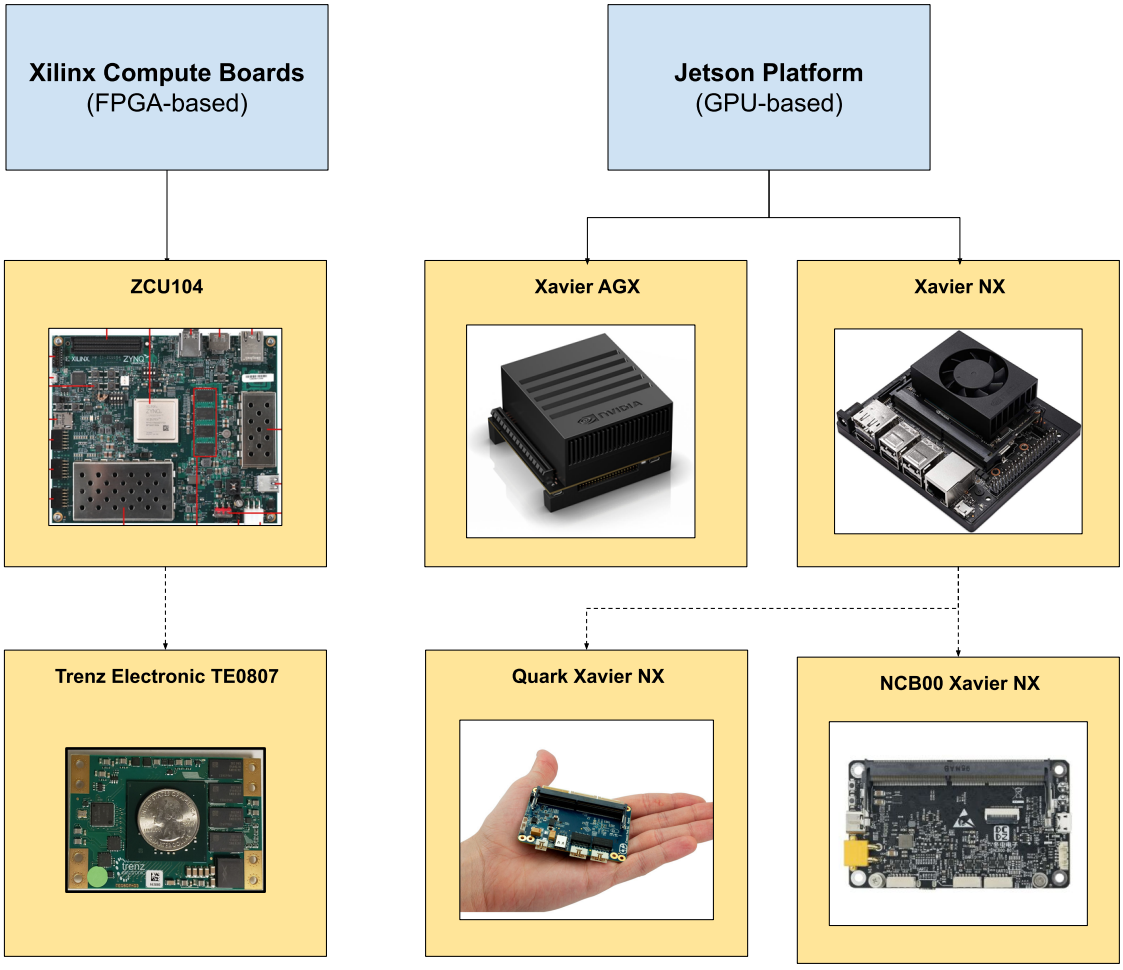


Figure 3.4: Three main hardware considerations - Xilinx UltraScale+ MPSoC ZCU104, Jetson Xavier AGX, and Jetson Xavier NX. *The ZCU104 provides a robust level of compute power as it contains media engines that can stream multimedia (such as 4k video input) as well as utilizing the DPU for accelerate neural network inferencing. Both Jetson Xavier devices also provided equivalent performance for their size and weight, with the AGX able to handle large neural networks with its 32GB of shared memory. The Xavier NX provides similar proportional performance when considering the size-to-compute ratio.*

3.3 Device Interconnections

The interconnection between the display, compute device, and the sensors are linear in nature. The RADAR device will record the scene and transmit the data to the compute device. The compute device will run the appropriate functions to transform the "raw" RADAR data to something usable for all the aforementioned features (localization, classification, etc.). Similarly, the compute device will take camera data and convert it into a familiar RGB matrix for detection and vital sign estimation. Finally, the display will receive the final image and render it. For the Xavier NX Devkit, the display can connect via HDMI and receive touch input via micro-USB. For the Quark, there must be another, smaller processor that can take the data stream from the Quark and render it on the display via the same sort of I/O.

The main connection method between the Xavier NX and the VMD3 is Ethernet. This Ethernet connection is used to send commands to the VMD3, as well as receiving a stream of recorded RADAR data. This functionality is described in more detailed in Chapter 6. The connection method between the camera and Xavier NX is the standard CSI-2 ports, and we utilize NVIDIA's Arguscam for connection and fast decoding via GPU. For Waveshare 5.5inch AMOLED display, the pre-built display board interfaces via HDMI for display output. Additionally, the touchscreen display is interfaced with a micro USB port. This port can also power the display if the port on the compute platform has a 5 volt power line.

Chapter 4

Real Time Detection and Tracking

4.1 Optical Data

For optical object detection, two popular datasets are utilized - Visual Object Context (VOC) and Common Objects in Context (COCO). Both datasets contain multiple iterations and these iterations are typically designated by a corresponding challenge year. In regards to Multi-object Tracking, the MOT17 and MOT20 datasets were used for evaluation and analysis of tracking methods in the optical domain.

4.1.1 VOC2007 & VOC2012 Datasets

The VOC datasets were a popular set of generic object detection datasets, with an iteration for every year between 2005 and 2012. For our purposes, we are utilizing VOC2007 and VOC2012 for their substantive additions to the VOC family of datasets. VOC2007 expanded the class number from VOC2006's 10 to 20 classes. VOC2007 was also the last year of the VOC challenges that the test set was released publicly [15]. Additionally, VOC2012 was chosen because it contained additional training and validation images. Training with VOC data typically involves merging the training and validation images of VOC2007 and VOC2012 and then evaluating on the VOC2007 test set. In terms of dataset sizes, VOC2007 contained a total of 9,963 images with 2,501 for training, 2,510 for validation, and 4,952 for testing. Additionally, VOC2012 contained a total of 11,540 images with 5,717 for training and 5,823 for validation.

4.1.2 COCO 2017 Dataset

The COCO 2017 dataset is a re-partition of the original COCO 2014 dataset and an extension of the respective COCO challenge year. Similar to VOC datasets, COCO datasets are composed of a training, validation, and testing set. The training and validation sets contain publicly available annotations while the test set annotations are only available via COCO's evaluation server. Because of this lack of publicly available test set annotations, this work utilizes the validation set and abbreviates "COCO 2017 Validation set" to "COCO val2017". COCO val2017 is composed of 5,000 and 80 object classes. It is important to mention that the class number within the annotations indicate 91 classes, but there is a distinction made

between "object" and "stuff" labels. The main distinction between the two labels is that a "stuff" label is applied to items in an image that do not have a clearly defined boundary [31].

4.1.3 Multi-Object Tracking Datasets

The Multi-Object Tracking (MOT) datasets are specifically used to train and evaluate MOT tracking methods. These datasets are composed of video footage from various perspectives including CCTV surveillance cameras as well as handheld footage with the camera person moving. There a set of classes in MOT that differentiate pedestrians from people on bikes/cars/trucks and large "obstructive" objects [37]. For this work's purpose, all classes except "pedestrian" are filtered out. Similar to VOC and COCO datasets, the MOT family of datasets have multiple iterations based on the year their respective MOT challenge events. In this work, MOT17 and MOT20 were utilized based on their unique contributions to the MOT dataset family. MOT17 is a reiteration of a previous MOT dataset (MOT16) but with corrected labeling and pre-computed bounding boxes from additional CNN-based detectors. MOT20 introduces new video footage but with a focus on higher density, crowd environments [13]. This introduces additional difficulty in both the association and localization tasks.

4.2 Techniques for Optical Object Detection

Since this thesis focuses on real-time inferencing, utilizing one-stage detectors is crucial. A select number of YOLO-based networks were examined for their mAP performance and robustness across different modalities. Because of its literature impact on one-stage object detection, this works utilizes YOLOv3 as a baseline network. Other networks were chosen based on their reported architecture improvements over the baseline, real-time speeds, and their memory footprint. All of the networks are initially evaluated under COCO val2017. Later, they will be trained on RADAR data and evaluated on both Range-Velocity and MVDR RADAR datasets. The evaluation of these networks consists of running each network at a confidence threshold of 0.001 to include as much detections as possible. Following COCO procedures, detections for each image are compiled into a single JSON file and evaluated with the official COCO evaluation tool for Python.
The evaluation results for COCO val2017 can be seen in Figure 4.2. YOLOv4 scored the highest mAP@0.5 of 68.21%, but all other similarly-sized YOLO-based networks score comparable mAP@0.5 scores. The mAP scores for the sub-10 million parameter (or miniaturized) networks show a greater mAP delta than the larger YOLO networks. YOLOv4 CSP Small scored the highest mAP@0.5 of 57.66%. Although, when analyzing from a parameter-size basis, there are two important findings. The first one is that the replacement of ResNet-like residual layers with CSP layers allows YOLOv4 CSP Small to outperform YOLOv3 Tiny by a mAP delta of 20.93% **while having ∼400,000 less parameters**. Similarly, YOLOv5n achieves competitive mAP results to YOLOv4 CSP and beats out YOLOv3 Tiny and YOLOv4 Tiny while being 4.9x and 3.4x smaller respectively.

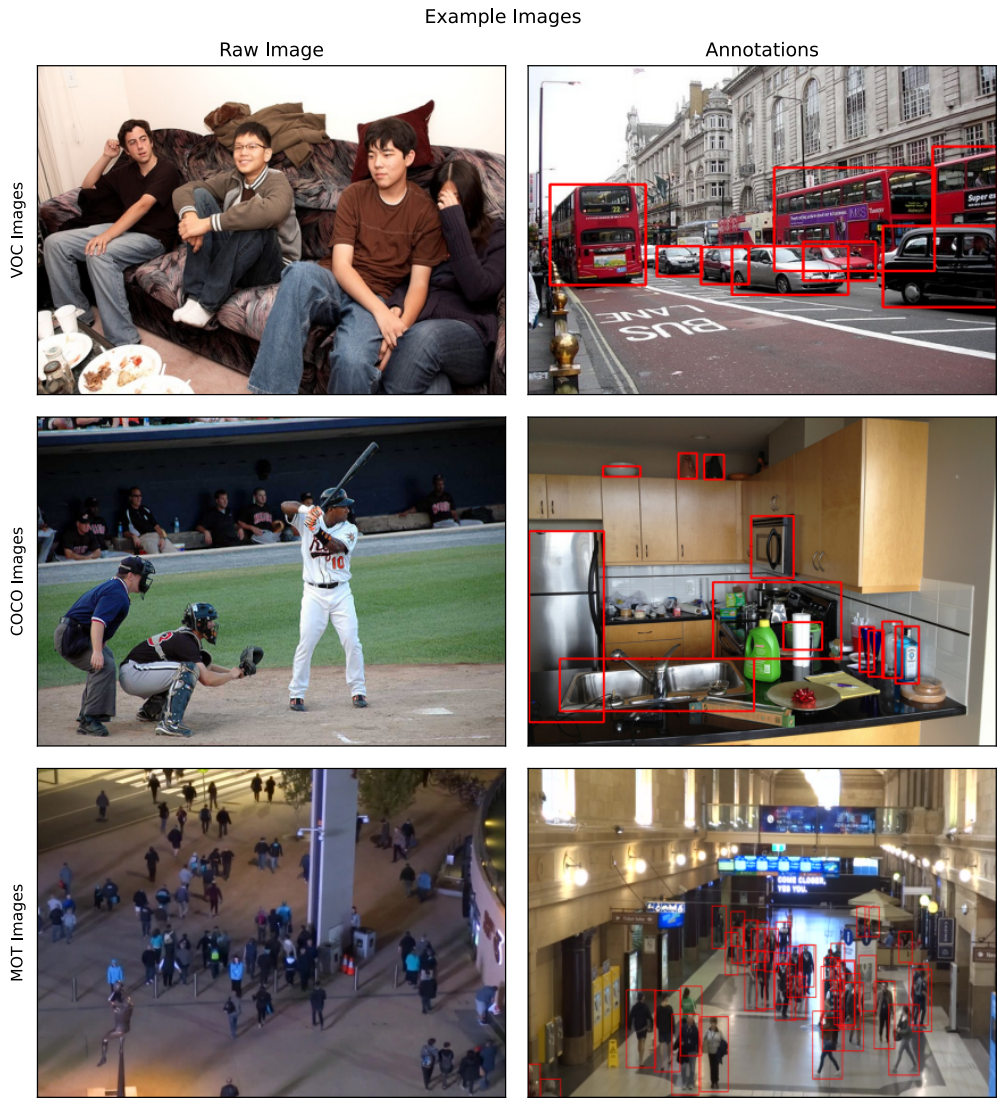


Figure 4.1: Image examples from the VOC datasets, COCO 2017 validation dataset, and MOT20 dataset. *All of the aforementioned optical datasets contain a diverse set of images and a relatively large number of classes. In terms of total dataset sizes, VOC2007 contains 9,963 images, VOC2012 contains 11,540 images, and COCO 2017 contains 123,287 (not including the testset) images. The VOC family of datasets contain 20 generic classes (persons, bottle, cars, etc.) while the COCO dataset contains 91 classes with similar generic labels. The important distinctive for class labels in COCO val2017 is between "object" and "stuff" categories. "Stuff" class labels are applied to things in an image that do not have clearly defined boundaries. For our purposes, we do not include the "stuff" category classes, which sets the number of usable classes to 80. The MOT data is composed of video folders with each image being an individual frame.*

Table 4.1: Table showcasing different YOLO-based networks and their parameter sizes. *Networks were chosen based on their notable improvements upon YOLOv3's architecture and training procedures. Networks, such as YOLOv5 and YOLOv4 CSP, are "scalable" networks, which give a large range of network subtypes. Such examples are YOLO5L (large), YOLOv5M (medium), etc. The network sizes for these scaled networks were chosen based on their approximate size to YOLOv3. YOLOv3 is considered a baseline network due to its significant literature impact. The "miniaturized" versions of the original networks (denotated by "Tiny" or "Nano" or "Small") are chosen for their deployability to embedded devices.*

Network Name	**Parameter Size**
YOLOv3	**61,497,430**
YOLOv3 Tiny	8,666,692
YOLOv4	**64,363,101**
YOLOv4 Tiny	6,056,606
YOLOv4 CSP	**52,921,437**
YOLOv4 CSP Small	8,270,621
YOLOv5L	**46,533,693**
YOLOv5n	1,760,518

Table 4.2: Inference FPS of all YOLO-based networks across discrete GPU, Jetson Xavier AGX, and Jetson Xavier NX. *Newer networks, such as YOLOv5L and YOLOv4 CSP, are more efficient than previous networks due to architectural and training advancements. Sub 10-million parameter networks are all above 30 FPS across all hardware platforms and are prime choices for implementation into the final real-time system. The performance magnitude between the discrete GPU and Jetson Xavier AGX and NX are dependent on network size. For networks with sub-10 million parameters, 6.79x and 7.86x speed-up between the discrete GPU and the AGX and NX respectively is seen. For bigger networks, the magnitude drops to 3.95x and 6.94x for the AGX and NX respectively. Between the AGX and NX, the AGX runs the networks an average of 1.48x faster.*

Hardware / **Networks**	**NVIDIA 2070 Super**	**Jetson Xavier AGX**	**Jetson Xavier NX**
YOLOv3	113.41 FPS	29.75 FPS	17.32 FPS
YOLOv3 Tiny	457.62 FPS	69.31 FPS	53.24 FPS
YOLOv4	72.09 FPS	17.38 FPS	9.87 FPS
YOLOv4 Tiny	463.05 FPS	66.17 PFS	54.77 FPS
YOLOv4 CSP	76.23 FPS	19.48 FPS	10.86 FPS
YOLOv4 CSP Small	280.30 FPS	53.32 FPS	47.32 FPS
YOLOv5L	156.32 FPS	39.87 FPS	22.61 FPS
YOLOv5n	998.60 FPS	120.34 FPS	105.34 FPS

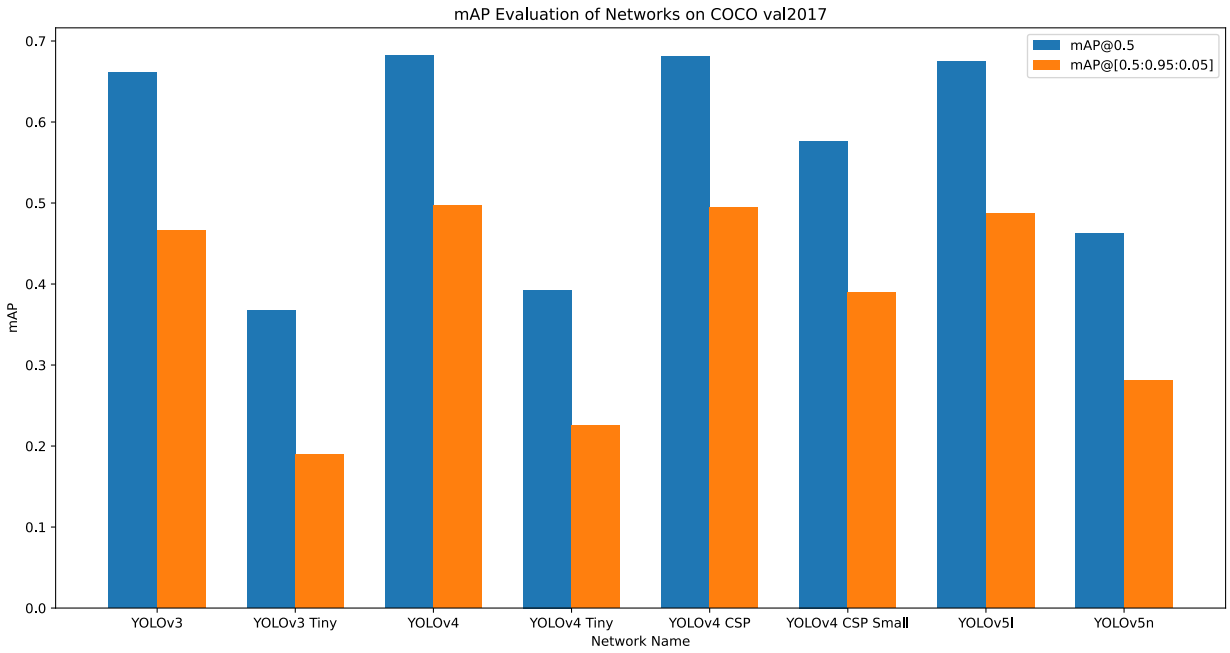


Figure 4.2: Graph showing the mAP results of various YOLO-based networks on COCO val2017. *All networks were evaluated directly from the Python COCO evaluation tools, and the mAP results are broken into mAP@0.5 (mAP at an IoU threshold = 0.5) and mAP@[0.5:0.95:0.05] (mAP at a range of IoU thresholds from 0.5 to 0.95 in intervals of 0.05). The latter enforces a* ***stricter*** *mAP computation, which is believed to have a closer association with real-life detection performance. From the evaluation, YOLOv4 achieves the highest mAP@0.5 and mAP@[0.5:0.95:0.05] with 68.21% and 49.41%. Although, all other similarly-sized YOLO networks achieve very similar mAP scores. Between the miniaturized models (sub-10 million parameter networks), YOLOv4 CSP Small achieves the best mAP scores while containing* ***less*** *parameters than YOLOv3 Tiny. YOLOv5n is also very competitive as it does better than YOLOv4 Tiny and YOLOv3 Tiny while being 3 to 4x smaller. Observations of the two mAP metrics show that there is just a static offset, and there is no change in network rankings between the two metrics.*

Inference times were also measured on all networks on different hardware platforms. These hardware platforms are a discrete GPU, the Jetson Xavier AGX, and the Jetson Xavier NX. The discrete GPU in this case is a mid-range Turing-based NVIDIA 2070 Super. Since it is discrete, the GPU is supported by a discrete CPU - an AMD Ryzen 2700X. It should be noted that the speeds recorded here are only inference speeds and do not include time for pre and post-processing. The post-processing operations, which can include algorithms like non-maximum suppression, can decrease total run time speeds and can involve other hardware components, such as the CPU.

Analysis of these run times shows a couple of outstanding points. Firstly, the mAP performance between YOLOv4, YOLOv4 CSP, and YOLOv5L in Figure 4.2 shows lack of substantive mAP improvements from YOLOv4 CSP and YOLOv5L when compared to YOLOv4. Figure 4.2 shows that the improvements brought by YOLOv4 CSP and YOLOv5L are related to computational efficiency. We get competitive mAP performance while running faster than YOLOv4. Additionally, all sub 10-million parameter networks were comfortably above 30 FPS. This gives a strong reason to utilize in the real-time system, especially considering some of their mAP performances.

Finally, we see the magnitude of improvement between the discrete GPU and the Jetsons correlating with the size of the networks. Sub 10-million networks are much faster on the discrete GPU than they are on the Jetsons. More specifically, we see a 6.79x and 7.86x speed up on the discrete GPU when compared to the AGX and NX respectively. For networks with over 10 million parameters, we only see a 3.94x and 6.94x speed up on the discrete GPU when compared to the AGX and NX respectively. This trend is not seen to this extent when comparing the AGX to the NX. The sub-10 million networks were on average 1.20x times faster on the AGX, and for bigger networks, were on average 1.76x times faster on the AGX.

4.3 RADAR Data

In regards to RADAR-based data, we utilize both real and synthetic-based RADAR data for testing and training our localization and tracking methods respectively. For real RADAR data, we capture data from different RADAR devices and process them with two different direction-of-arrival (DoA) methods. For the synthetic data, a program is created to generate images that are similar in appearance with the aforementioned real RADAR data. Details of the program and the output dataset are described in Section 4.5 ("Training with Synthetic RADAR Data").

4.3.1 Synthetic RADAR Data

Limited data is a major problem area for any kind of neural network, but this is an acute issue for CNNs. Two main approaches for this situation is to develop synthetic data, such as MOTSynth and MOTX, or enacting architectural changes that can increase learning ability with the existing, limited data. For this thesis, the initial steps towards RADAR localization

was focused on developing synthetic RADAR data since the collection of real RADAR data was initially difficult to produce. Some of these difficulties were the processing time of "raw" RADAR data to image-based outputs as well as the time it took to do each recording. These were all pipeline-related difficulties that were alleviated or solved later on when the real-time prototype was functional.

Creation of synthetic RADAR mainly relied on visual similarities to a small set of existing real RADAR data. Albeit, this visual reference was not directly applicable to the RADAR output from the prototype. This appearance-based synthetic data route was chosen over approaches related to electromagnetic simulation. Even though simulations would have produced more accurate synthetic RADAR data, the appropriate tools required can take a massive amount of compute resources and time.

This first set of synthetic RADAR data is detailed in Figure 4.3. It draws a simple "blob" that has a circular curvature through it to add more irregularity to the overall signature appearance. This first dataset contained 10,000 images and was split into a 70/30 train and test split. Furthermore, YOLOv3 was trained on this dataset and evaluated on both the Range-Velocity and MVDR datasets (Figure 4.4 and 4.5 respectively). From both of these figures, it can be seen that the network has a high average mAP on the Range-Velocity data versus a very low and inconsistent mAP on the MVDR-based RADAR. This suggests that the dataset is not very generalizable and cannot be reliably used in RADAR data that has been processed in different methods.

To address the ungeneralizable nature of the Synthetic RADAR Data v1, another appearance-based synthetic RADAR dataset was created. This dataset (called Synthetic RADAR Data v2 from now on) is a program that allows users to define more variables regarding the canvas, the types of signatures, and the overall diversity present in the dataset. For example, a user can define a range of image canvas sizes, the size of signatures, the shape of the signatures, as well as the intensity of background stationary signatures. Additionally, the program produces a user-defined number of "clips" with all of the aforementioned variables. Each clip is a randomly generated linear path, and signature-related and background-related perturbations are generated based on a seeded pseudo-random number generator. This program created the initial set for Synthetic RADAR Data v2, which consisted of 5 different clips with a total composition of 8,518 images. These images were then partitioned into a 80/20 split of training and validation set images. The visualization of some of the output images can be seen in Figure 4.6. Similarly, some statistical information related to the generate dataset is shown in Figure 4.6.

4.3.2 Real RADAR Data

Range-Velocity RADAR Data

The first dataset is provided by Yao Yu from MaXentric. The dataset is collected with a Sensorlogic UWB RADAR device. A clutter suppression algorithm is applied for each range bin, and an FFT is applied along the slow-time direction. The method outputs a 2D matrix,

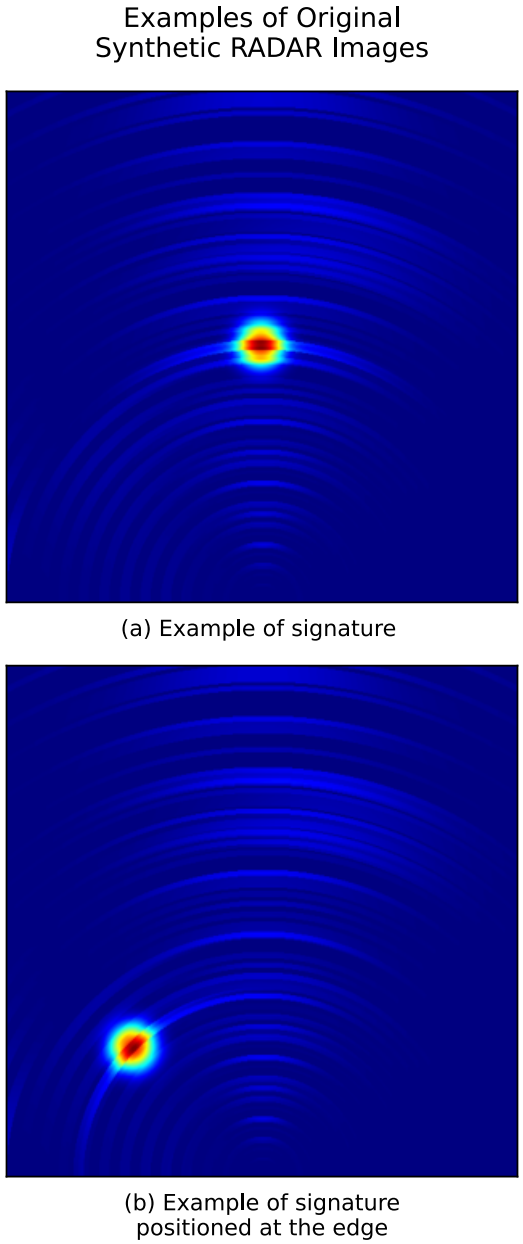


Figure 4.3: Example outputs from Synthetic RADAR Data v1. *Because no prior RADAR data directly applicable to the vital-sign product was available, the first attempt of creating synthetic was very simple. There would be a single signature with a set of static, curved lines to produce some perturbations. Additionally, a similar curve goes through the signature to add more irregularity.*

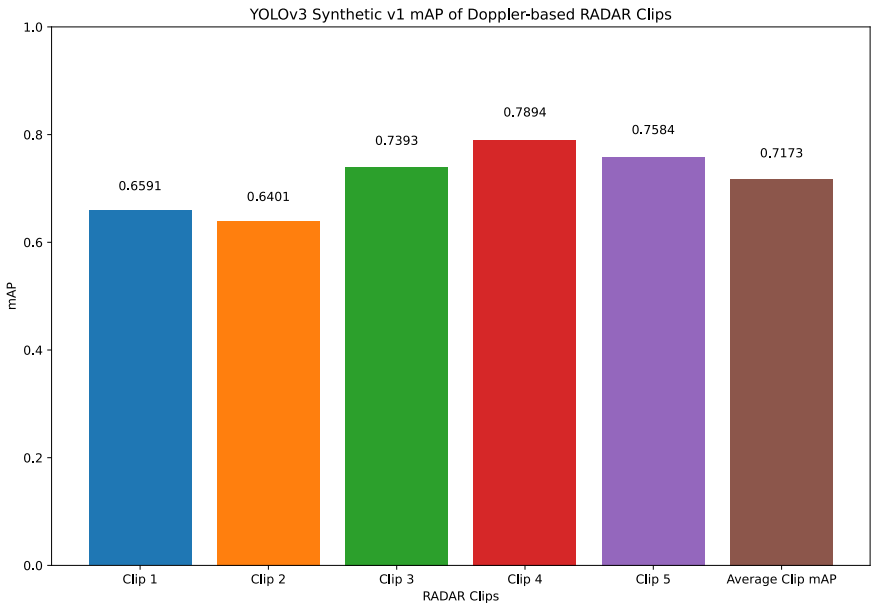


Figure 4.4: YOLOv3 trained on Synthetic RADAR Data v1 tested on the Range-Velocity test data. *The mAP results on the Range-Velocity dataset average out to 71%. This can be attributed to the Range-Velocity data having similar visual simplicity compared to Synthetic RADAR Data v1.*

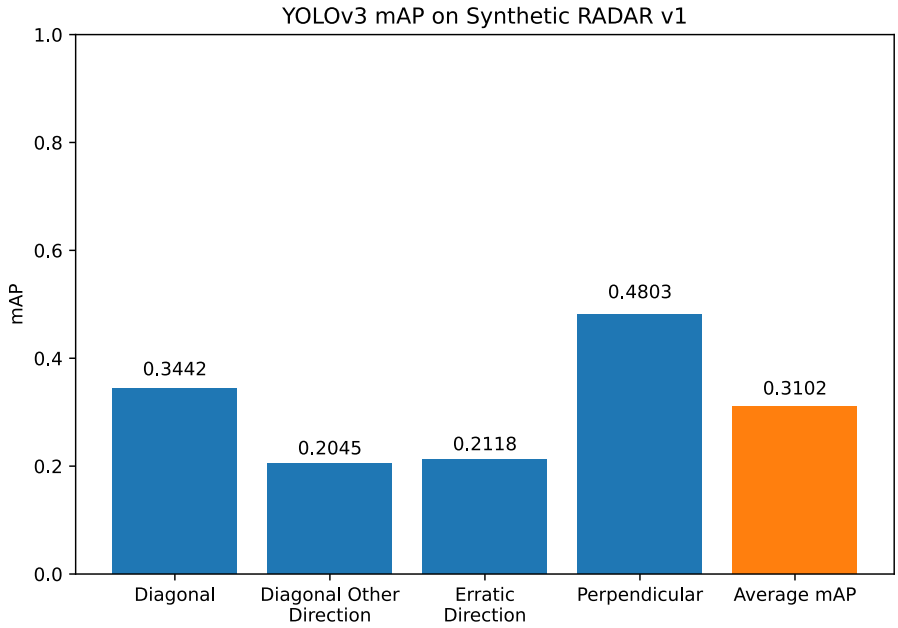


Figure 4.5: YOLOv3 trained on Synthetic RADAR Data v1 tested on MVDR-based test data. *Unlike Figure 4.4, YOLOv3 trained on Synthetic RADAR Data v1 scores an average mAP of 31%. The inconsistent performance between the two real RADAR data indicates that the Synthetic RADAR Data v1 does not generalize well.*

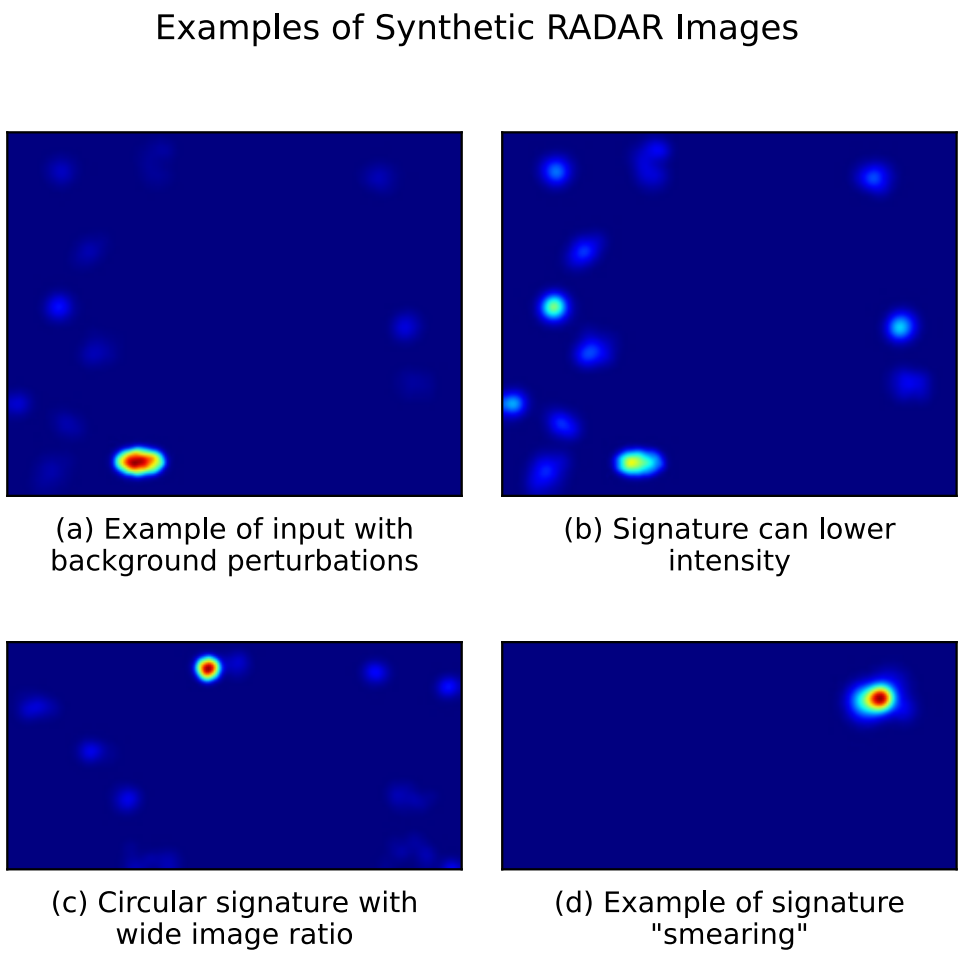


Figure 4.6: Example outputs from the appearance-based synthetic RADAR. *Unlike Synthetic RADAR Data v1, this dataset utilizes a range of different canvas scale. Additionally, the radial line is removed and replaced with stochastic perturbations that can "smear" the primary, circular signature. Furthermore, there are additional degradation methods including static, background perturbations as well as signatures with variable intensity during a certain path.*

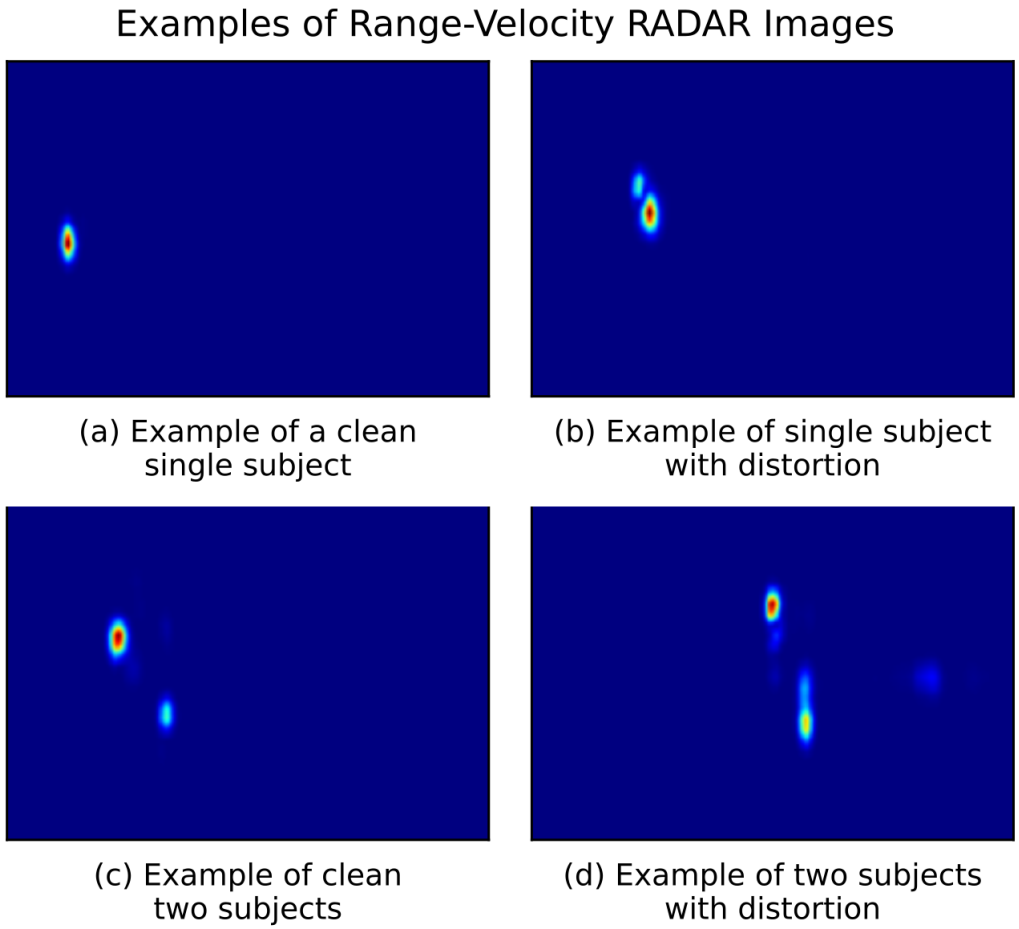


Figure 4.7: Examples of Range-Velocity RADAR data. *The data was collected by Yao Yu with a Sensorlogic UWB RADAR. Fast Fourier Transform (FFT) and a clutter suppression algorithm is applied to produce a 2D matrix, where the x-axis represents "range" and the y-axis representing "speed".*

Table 4.3: Table of different clip descriptions from the Range-Velocity dataset.

Clip #	Clip Name	Description
1	familyRoom_fs200_twoSubjects_cross WalkBackForth_freeAir_noUser	Two subjects walking back and forth.
2	familyRoom_fs200_twoSubjects_cross WalkBackForth_throughWall_noUser_1	Clip #1 but the RADAR device is behind drywall.
3	familyRoom_fs200_twoSubjects_ crossWalkBackForth_throughWall_noUser	Same as Clip #2
4	familyRoom_fs200_twoSubjects_crossWalk-BackForthSideToSide_freeAir_noUser	Two subjects walking in a circular motion.
5	familyRoom_fs200_twoSubjects_crossWalk-BackForthSideToSide_throughWall_noUser	Clip #5 but the RADAR device is behind drywall.

where the x-axis represents "range" and the y-axis representing "speed" (Figure 4.7). The dataset contains five different clips which are around two minutes long. Table 4.3 describes each clip's name, duration, and movement description.

MVDR-Based RADAR Data

The second dataset is provided by Marvin Joshi and Abdel-Kareem Moadi from the University of Tennessee. This dataset is composed of a singular subject walking in four different directions: Diagonal, Diagonal Other Direction, Erratic Direction, Perpendicular (in respect to the RADAR). The data was captured from a FMCW RADAR device and the processing method used was MVDR. The MVDR output is a 2D matrix where the x-axis is the angles (from -90 to 90) and the y-axis is the range (Figure 4.8).

4.4 Training on Synthetic RADAR Data

An analysis of mAP performance was done with the Synthetic RADAR Data v2 on different types of YOLO networks. The mAP of the synthetic validation is shown in the Appendix, but the main focus of this analysis to see the mAP performance of these networks against the Range-Velocity and MVDR RADAR data. If the synthetic data is a generalizable set, then comparable performance will be seen across both types of datasets.

From both Figure 4.9 and 4.10, it is clear that the Synthetic RADAR Data v2 is empirically more generalizable than its previous iteration. YOLOv3 trained on version 2 scores an mAP of 84.15%, which is 12.42% higher compared to the same model trained on the first version. This increase in mAP is shown to a greater contrast when looking at the MVDR performance. YOLOv3 trained on Synthetic RADAR version 2 is on average 41.2% higher than its counterpart trained on the first iteration. Other YOLO-based models retrained on Synthetic RADAR Dataset v2 also performed with similar mAPs for both MVDR and Range-Velocity datasets.

Comparisons between the different YOLO-based networks show that YOLOv5L performs the best on both MVDR and Range-Velocity testsets with an average mAP of 78.1% and 87.11% respectively. The mAP delta between YOLOv5L and the 2nd place network, YOLOv4 CSP, is 2.29%. This delta is relatively high when compared to the same mAP delta of 0.52% on the MVDR-based test. Amongst sub-10 million parameter networks, the top performer depends on the dataset. YOLOv3 Tiny yielded an mAP of 84.15% on the Range-Velocity testset. On the MVDR-based dataset, YOLOv4 Tiny performed the best with an mAP of 67.73%. Furthermore, the 2.77% mAP delta between YOLOv4 Tiny and YOLOv4 CSP Small is more notable than the 0.41% mAP delta between YOLOv3 and YOLOv4 CSP Small on the Range-Velocity testset.

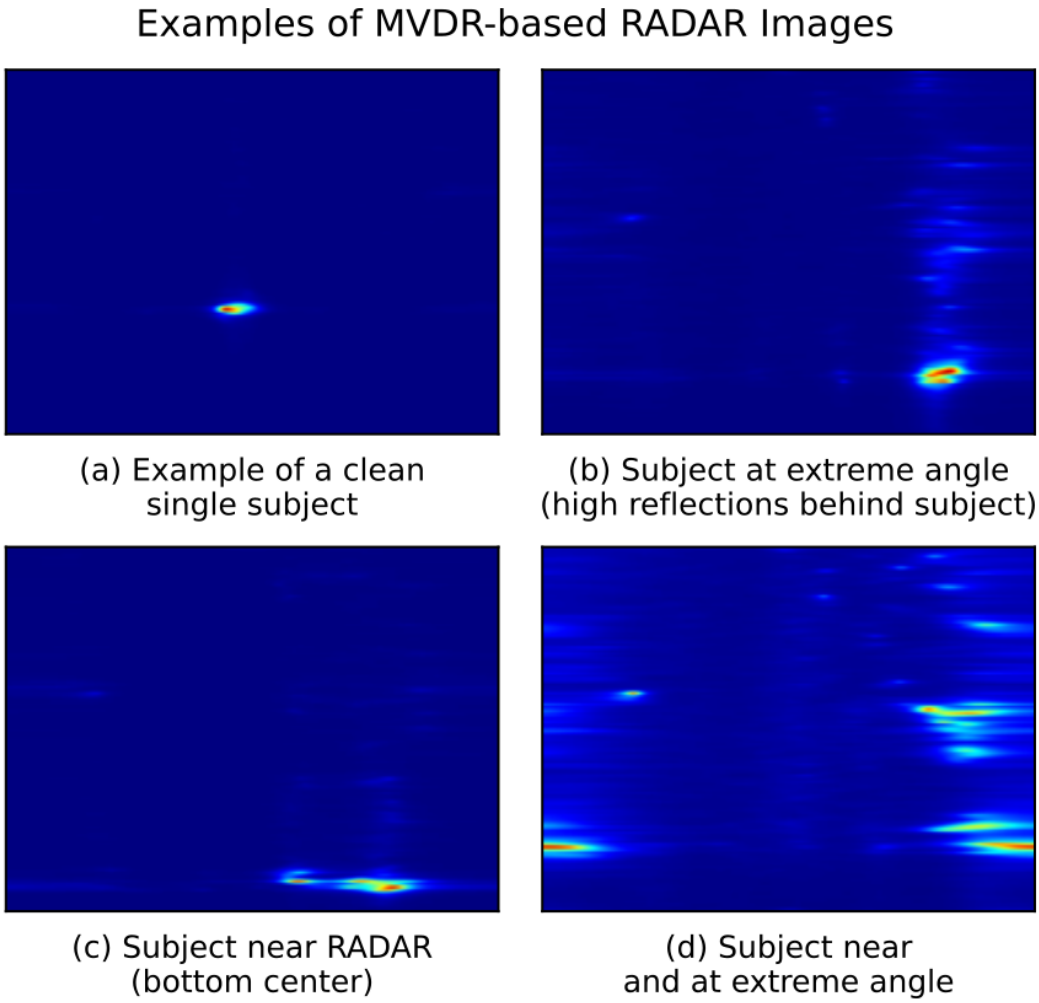


Figure 4.8: Examples of MVDR-based RADAR data. *With MVDR, positional data is more intuitive compared to the Range-Velocity approach. The x-axis is the angle from the center (where the RADAR is located) while the y-axis is the physical range from the RADAR. For the FMCW RADAR device, the maximum range is around 7 meters.*

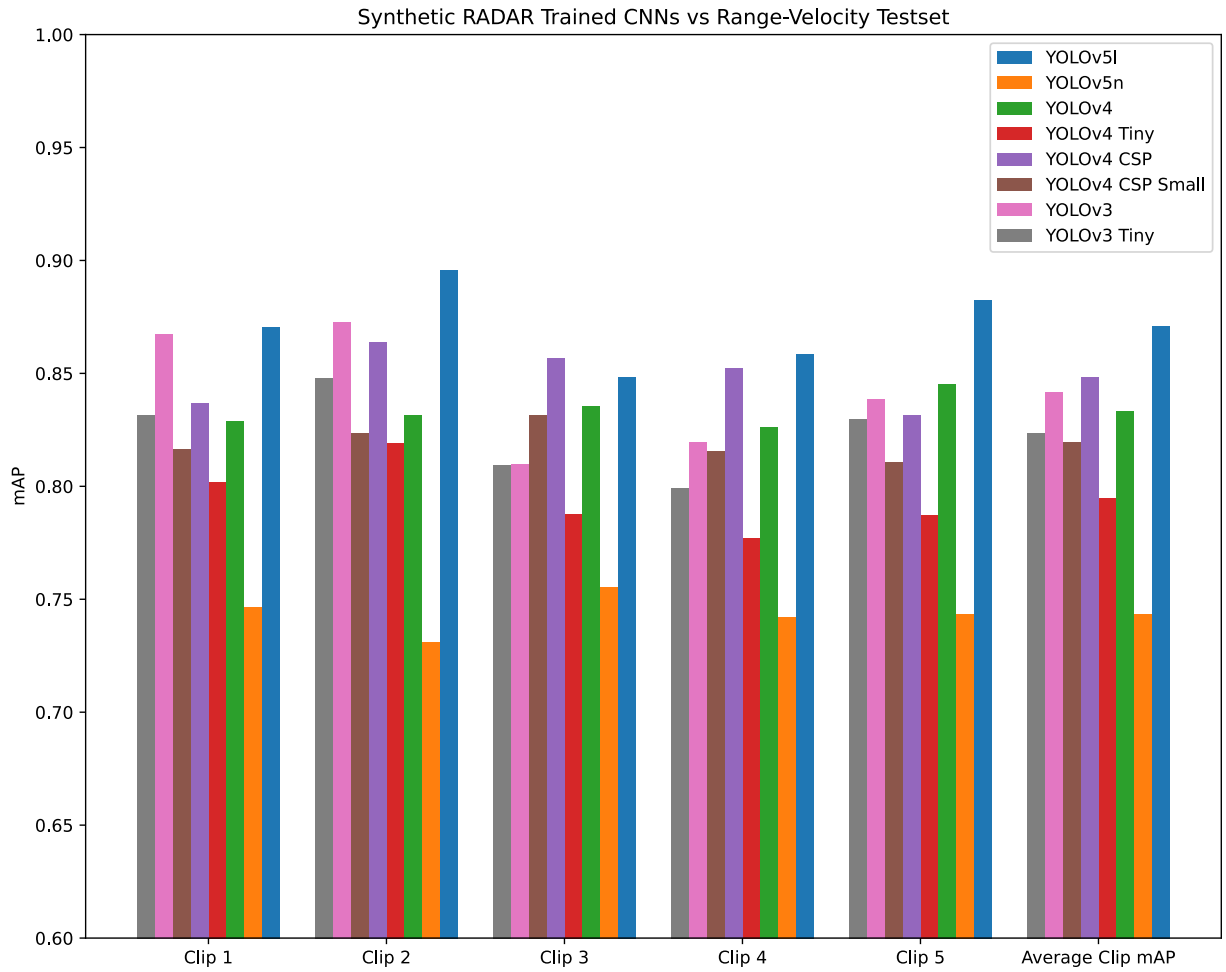


Figure 4.9: mAP comparison of multiple object detection networks on the Range-Velocity test set. *The dataset is composed of two subjects walking in five different methods and in two different environments - through wall and non-through wall (also annotated as "freeAir"). Unlike Figure 4.4, the YOLOv3 network trained on Synthetic RADAR Dataset v2 scores 9% higher mAP. All networks, even the networks with less than 10 millions parameter, achieve a higher average performance than the performance of YOLOv3 shown in Figure 4.4.*

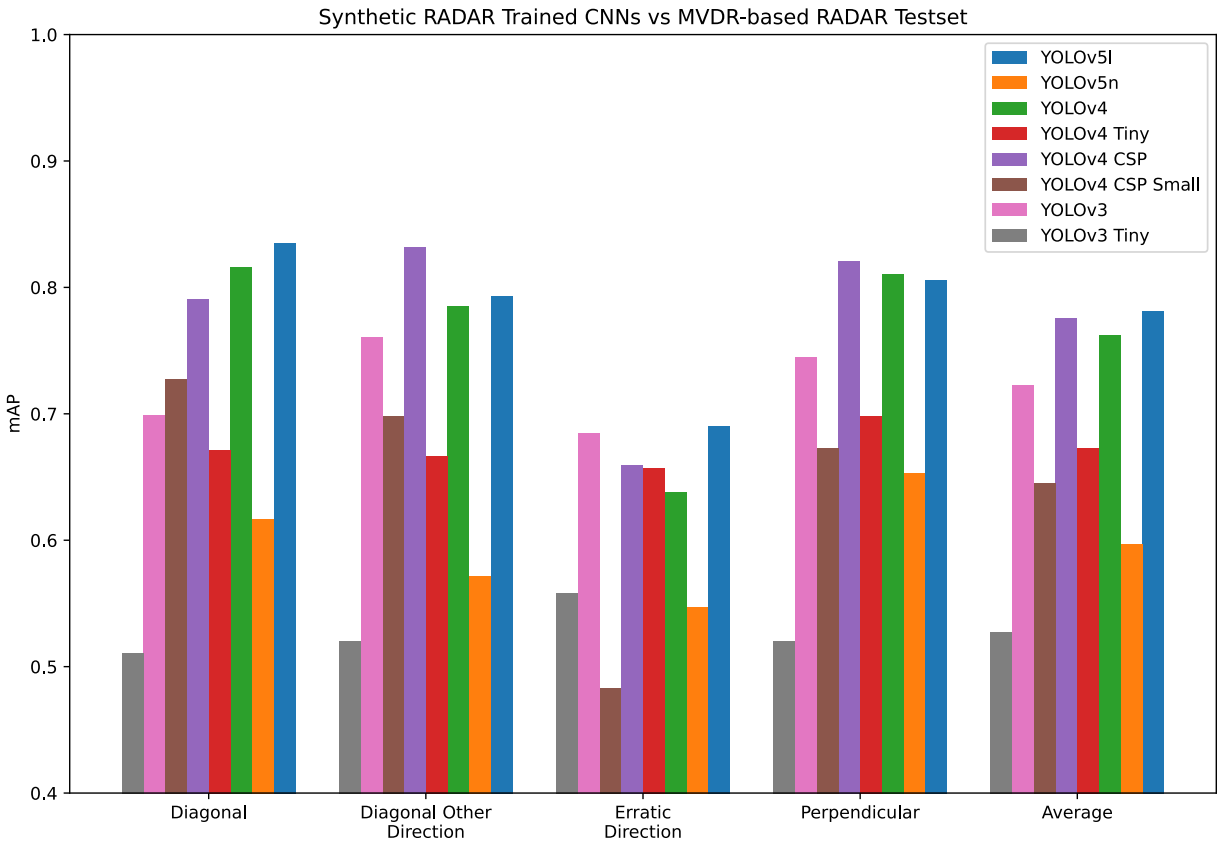


Figure 4.10: mAP comparison of multiple object detection networks on a MVDR-based RADAR test set. *The dataset is composed of four different clips of a single subject walking in a given direction or pattern. The average mAP score between the YOLOv3 shown in this figure is ∼40% higher than the YOLOv3 shown in Figure 4.4. This indicates that this dataset is more generalizable than the first version. Additionally, a comparison between the different YOLO-based networks show that YOLOv5L on average scores the highest mAP across both datasets. The performance is quite noticeable on the Range-Velocity data. Meanwhile, the mAP delta of YOLOv5L and YOLOv4 CSP on the MVDR-based is less notable. Sub-10 million parameter networks predictably have worse performance than their larger counterparts. Additionally, their performance rankings greatly differ amongst themselves between the two datasets.*

4.5 Training on Real RADAR Data

The real RADAR data collected in the previous section required a relatively long process of recording "raw" RADAR frames, processing them offline, and manually annotating all of them for a usable benchmark. With all of the optimizations and consolidation of different processes for the final real-time system, data collection became streamlined. With the assistance of Chandler Bauder and Peter Alley, an additional MVDR-based dataset is created. This dataset also captures simultaneous optical camera data from an IMX477 camera connected to the Xavier NX. The camera data is a future-proofing attempt for additional multi-modal tasks in this research project that extend beyond this thesis' contribution.

The dataset contains 13 recordings, with a range of one to three individuals walking in erratic movements for 300 "timesteps" (Figure 4.11). This makes this particular dataset larger than the other two datasets in terms of the number of images and the number of individuals present. For annotations, because there are networks available for signature detection (from Chapter 4) and face detection, auto-labeling is applied. This is a process of utilizing an existing detector that is relatively accurate and having it annotate unlabeled data. Afterwards, a person cleans up any mistakes made by the network. For this dataset, the two networks were YOLOv5L from Chapter 4 and TinaFace, which is a SOTA face detector [63].

Apart from just detection boxes, additional association labels are also created between the RADAR and optical modalities. With this annotation, it is possible to tackle cross-modal association tasks for vital-sign estimation synchronization and robustifying existing tracking methods. Since this dataset gives additional MVDR data, we train the sub-10 million parameter networks defined in the Synthetic RADAR Data section on this dataset's MVDR data and test them against the MVDR-based dataset created by Marvin Joshi and Abdel-Kareem Moadi. We also train the same networks against different, smaller input sizes to see the performance of mAP while increasing overall speed due to less preprocessing and smaller network size. The hyperparameters for these training sessions are identical to the synthetic training excluding epoch number.

Training with real MVDR data produced better mAP results than utilizing the synthetic, generic RADAR dataset. The real data networks improved their mAP on average 19.25% compared to their synthetic counterparts. From both the synthetic and real data training sessions, YOLOv4 CSP Small proved to be the most competitive or best performing network amongst the sub 10-million parameter networks. Both types of analysis would point to utilizing this network for the RADAR portion of the real time system, but looking at performance across input sizes shows that YOLOv5n can perform at the same level as YOLOv4 CSP Small at the smallest image input size. Also, from Figure 4.2, YOLOv5n is 2.24x faster than YOLOv4 CSP Small at a largest image size. For these reasons, utilizing YOLOv5n at an image input size of 128 would be the optimal choice for RADAR localization in the real-time system.

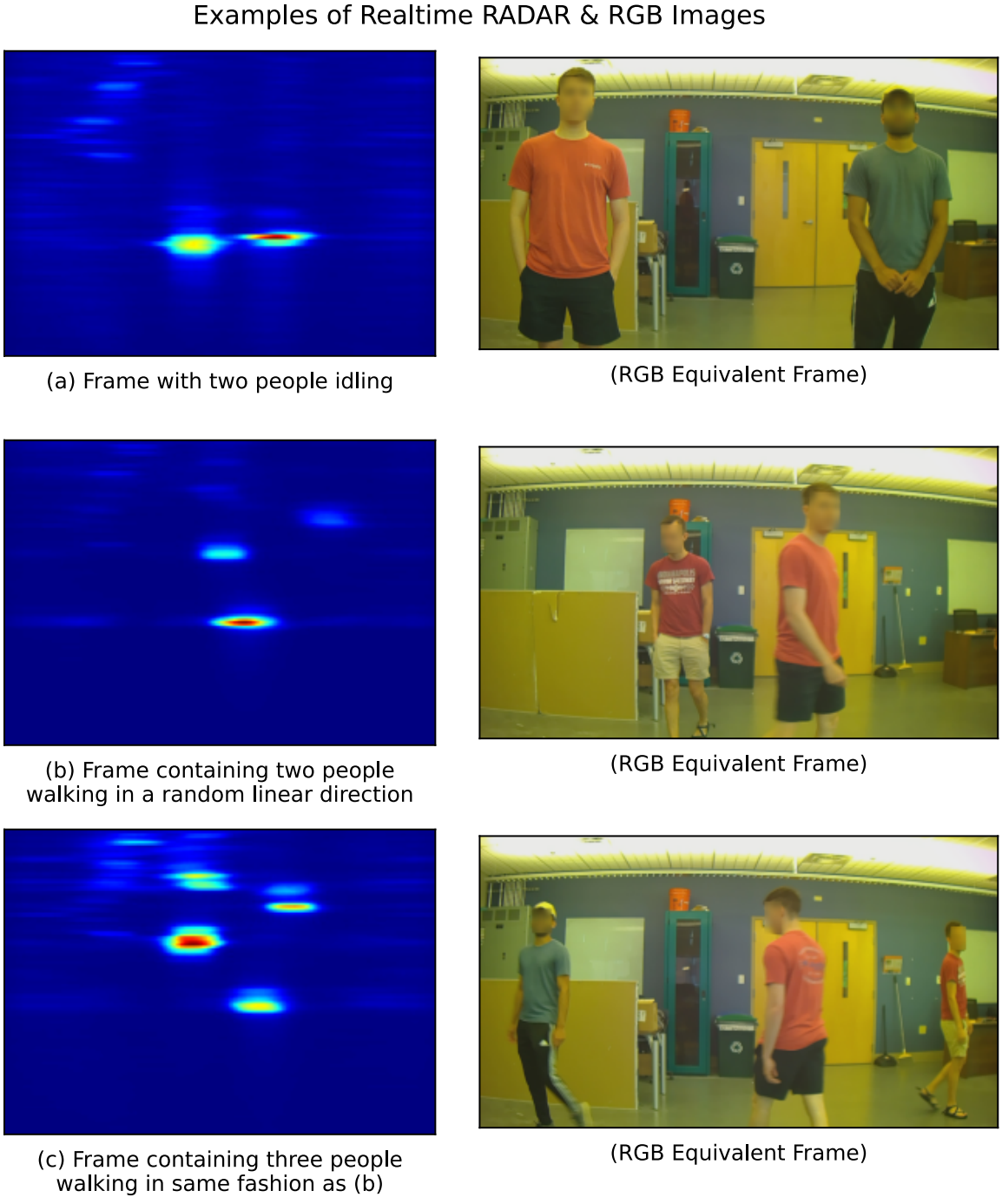


Figure 4.11: Example of dataset collected using real-time data collection. *This dataset collected clips of one, two, and three subjects. Each clip was 300 frames long and simultaneously captured RADAR and optical data. The RADAR data was processed using MVDR, similar to the MVDR-based dataset in Chapter 4.*

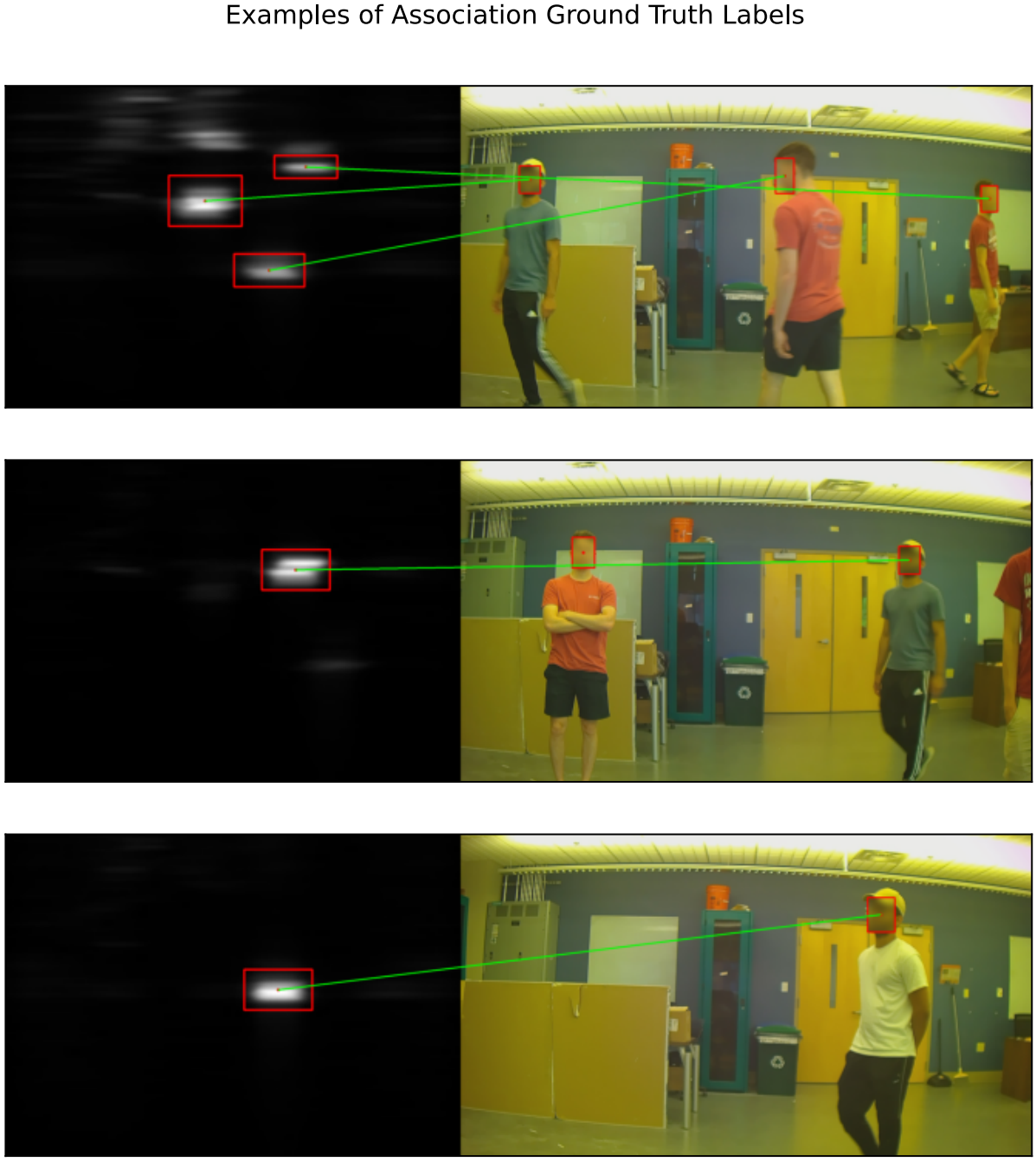


Figure 4.12: Examples of dataset annotations. *Similar to the datasets detailed in Chapter 4, this dataset provides bounding box annotations for RADAR signatures. To expand further than the other two datasets, faces in the optical dataset are also annotated. For future research objectives, "association" annotations are also created to help map detections across modalities.*

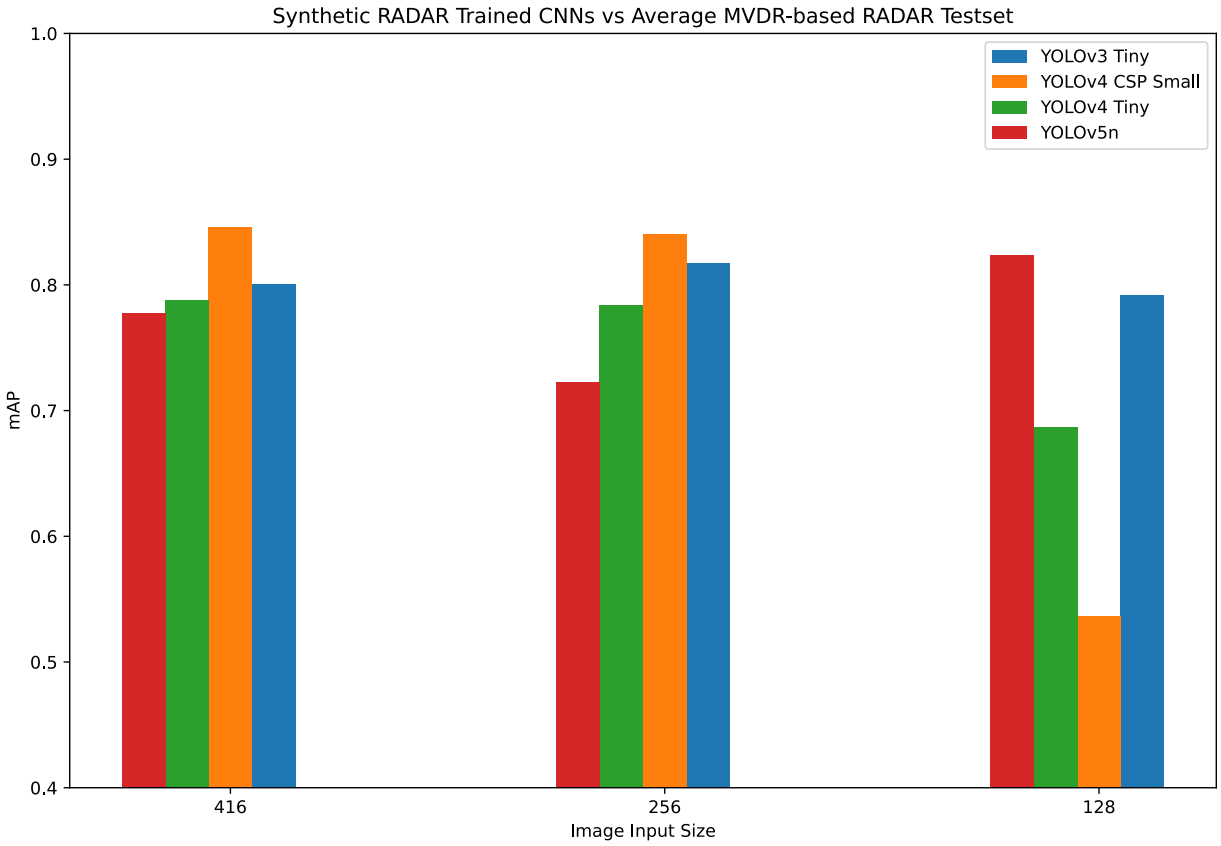


Figure 4.13: Training results with sub-10 million parameter networks on the real-time MVDR data. *There is no consistent trend regarding network performance across image sizes, but YOLOv4 CSP Small still produced the best results with an image size of 416, and it out-performed its synthetic counterpart with an mAP of 84.61% vs 64.52%.*

4.6 Cross-Modal Association

As stated before, the real-time MVDR dataset contains a new type of annotation. This annotation is an association index that connects a given detection in RADAR to a detection in optical. Practical usage for this kind of association would be tie together both modalities to produce better, more stable vital sign estimations. Additionally, the association could help with combining tracking outputs from both modalities. This thesis will *introduce* a method to tackle this task.

This association task can be tackled naively by using a machine learning method to learn an estimated projection between the two modalities. We utilize a Support Vector Machine (SVM) to do this estimated projection. A SVM is a classification method that utilizes a hyperplane to create proper class boundaries for a given training set [60]. The training set filters out empty samples as well as samples with no detections in one modality. Furthermore, we also filter out any duplicate positions in either modality. This leaves 2,900 training samples for the SVM to be fitted against and 500 testing samples for evaluation.

Each sample consists of detections from both RADAR and optical data. Similar to ViFi's baseline, this problem is reinterpreted as a complete bipartite graph, where the two sets are the corresponding modalities' detections. This means that each node (detection) in one set is connected by an edge to all other nodes in the other set. The task then becomes "edge prediction", which is essentially predicting if a given edge exists within a graph. The SVM is doing binary classification on each edge of the graph, which consists of a given RADAR-and-optical detection pair. The pair contains a vector representing normalized coordinates of the center of each detection. For inference, The SVM outputs a boolean vector and the pairs labeled false are filtered out. The remaining pairs are then visualized with an edge (Figure 4.14).

Cross validation is performed during SVM training with a fold of 10, which means 10% of the training data was reserved for a validation set. The 10% partition was shifted each fold as to reduce the possible bias from the data. Additionally, there is a limitation with the dataset itself, where there is only one main perspective where all the face detections were processed from. To alleviate this, random numbers between 0 and 1 are generated from a Gaussian distribution and added to the y-component of the face detections. For every fold, the validation and test accuracy are roughly the same with some small variation. The average test accuracy is 94.25% and the average validation accuracy is 96.26%. The common error in the association is typically when RADAR detections are matched to the same face detection (Figure 4.16). Since the SVM scored relatively high accuracy, can we say that the cross-modality association task has been sufficiently solved? Unfortunately, shifting on the y-axis does not account for many other perspectives that the configuration can be in, so a development of a much larger dataset is needed. What if the y-component was not added to the fitting test? The SVM accuracy actually falls to a 86.89%. This accuracy is likely much lower when applied to other perspectives.

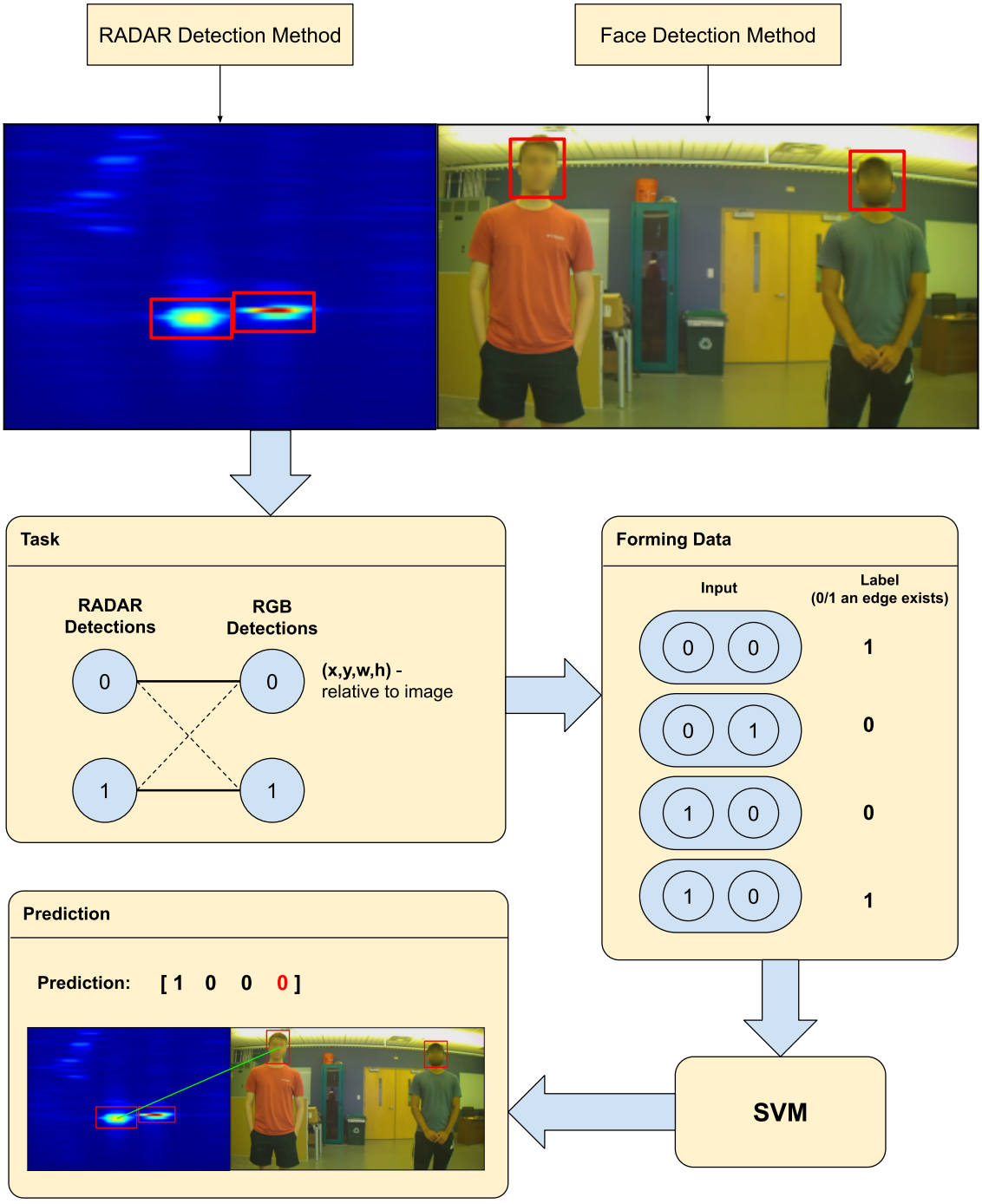


Figure 4.14: Diagram showing the reframing of association task as a graph-like problem. *A SVM-based binary classifier determines whether or not a given RADAR and Optical pair exists. The pairs represent an edge in a graph, and a complete bipartite graph is created during training and inferencing. During inferencing, the predictions classified as false are removed.*

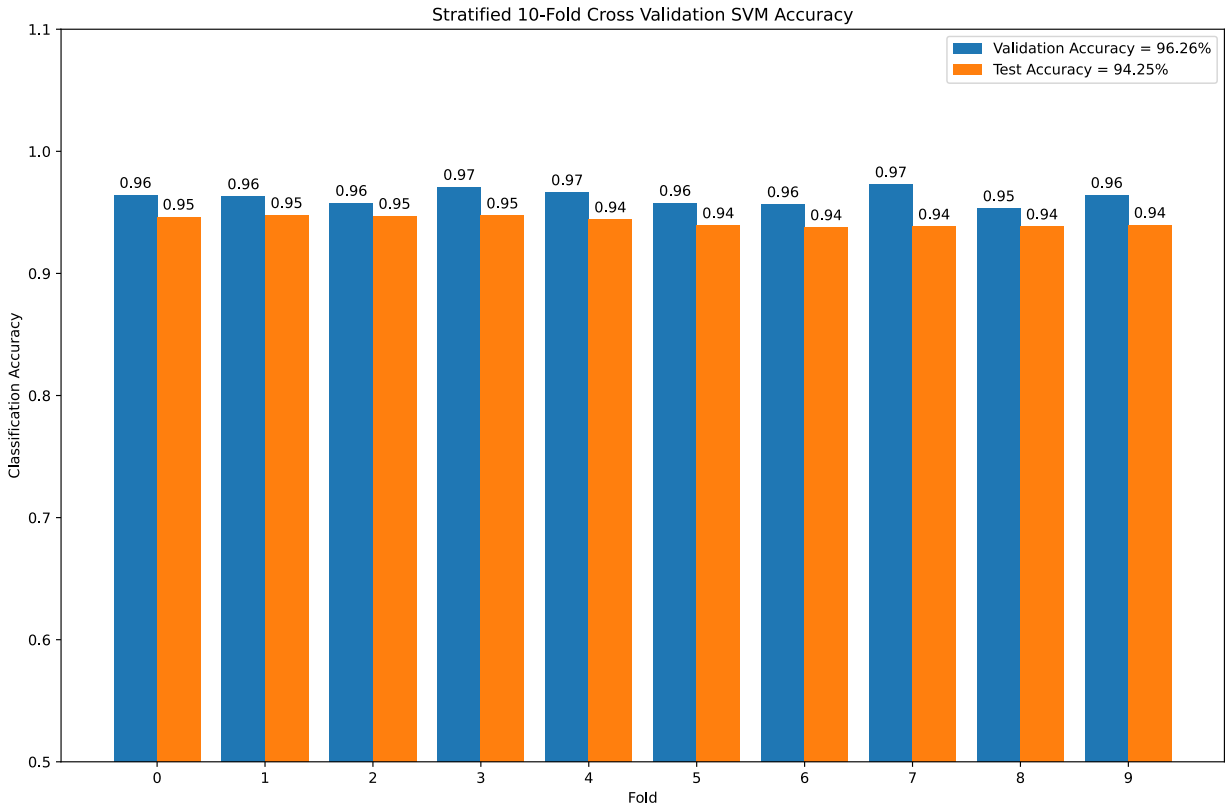


Figure 4.15: Classification accuracy of SVM training for cross-modal associations. *10-folder cross validation is done on the training set on a seperate validation set for each fold. The average validation accuracy 96.26%. After each fold, we also run a seperate test set on the SVM, and the average test accuracy is 94.25%. The SVM fitted on a certain fold with the highest test accuracy is saved.*

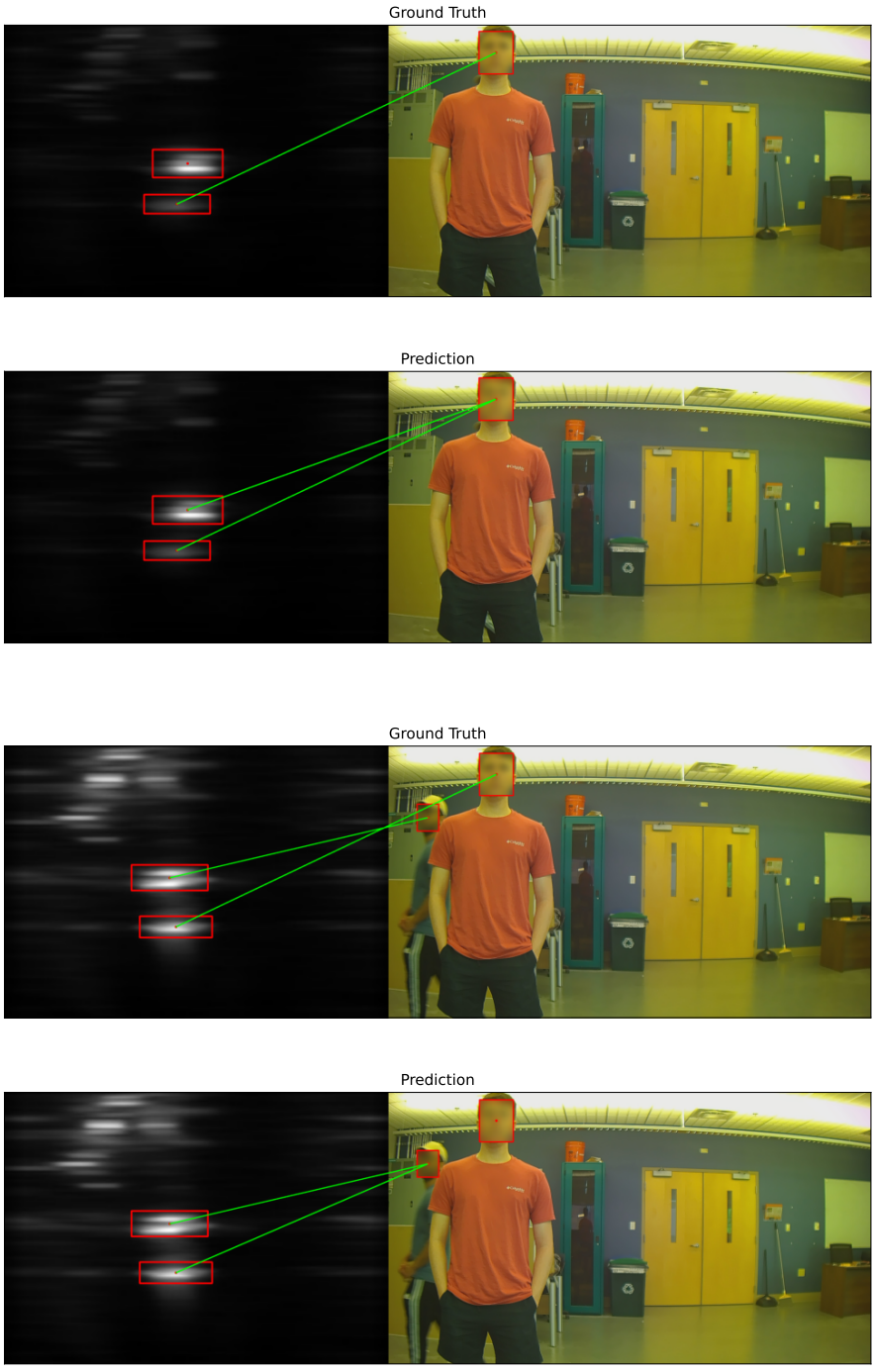


Figure 4.16: Erroneous predictions produced by SVM-based association technique on testset. The majority of errors occur when detections in either modalities are relatively close to one another. This causes the SVM to produce a false positive association.

4.7 Techniques for Multi-Object Tracking

The detection methods that were previously mentioned lacked any sort of association within a stream. This is vital for live optical video surveillance and for this system. MOT methods described in Chapter 2 are utilized in this work's prototype to help with subject tracking as well as alleviate potential false positives from the object detector. These trackers were benchmarked against the MOT17 and MOT20 training sets (optical). This benchmarking is defined in Figures 4.17 and 4.18. A similar tracker analysis is also done on RADAR data.

For the detection input for the tracking methods, this analysis utilized YOLOX-X, a fully-convolutional CNN defined in [20] and trained by [58], for all trackers. Additionally, evaluation was done with TrackEval for the HOTA evaluation [26]. From this benchmark, the clear leader is BYTE with a HOTA of 0.75 and 0.76 for MOT17 and MOT20 respectively. This is not very surprising as [58] showed BYTE outperforming SORT and DeepSORT at equal values for both the MOT17 and MOT20 training and testing sets. What is surprising is SORT having a significantly worse performance under MOT20 compared to other trackers and its own performance in MOT17. A possible reason is that MOT20's higher density environment increases difficulty of the Kalman filter matching based solely on position. DeepSORT utilizes the same kind of position-based matching but also incorporates CNN-based appearance vectors as another matching criteria. This allows DeepSORT to score 19% higher on MOT20 compared to SORT.

HOTA values for IoUTracker are relatively competitive to the other, more advanced methods. This can be explained by the fact that a lot of subjects in both MOT17 and MOT20 are consisted detected by YOLOX-X. The consistency of detection is partly due to the performance of the CNN but also the fact that individuals in these videos often than not stay in a predictable course, do not change speeds, and are typically not obstructed for long periods of time. Because of these video attributes, utilizing a simple matching metric like IoU is sufficient.

The run time was also calculated for each tracking method on multiple platforms in Figure 4.19. Since all of the tracking methods - excluding DeepSORT - perform tracking updates using state prediction, they all have real-time capabilities. But, for analysis-sake, the IoU Tracker is the fastest method and by a large margin when compared to the other methods. This is because it lacks any sophisticated state estimation functions, such as the Kalman Filter that's present in SORT and BYTE.

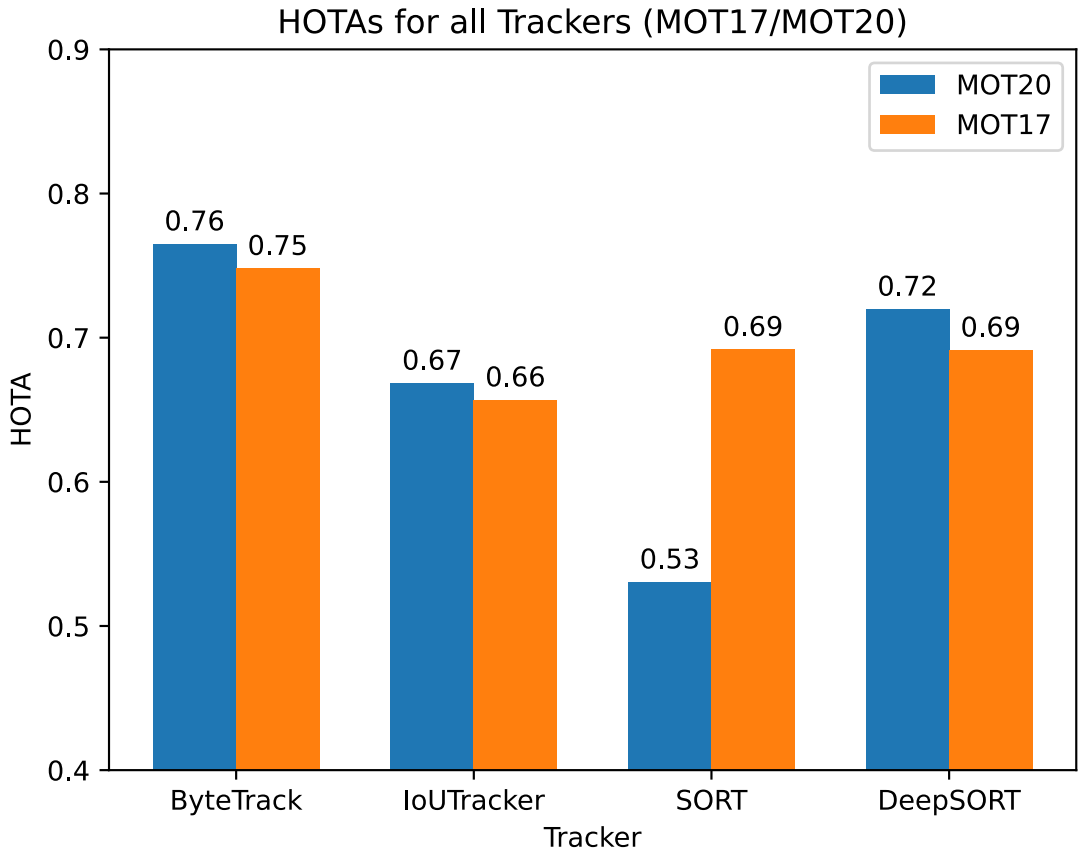
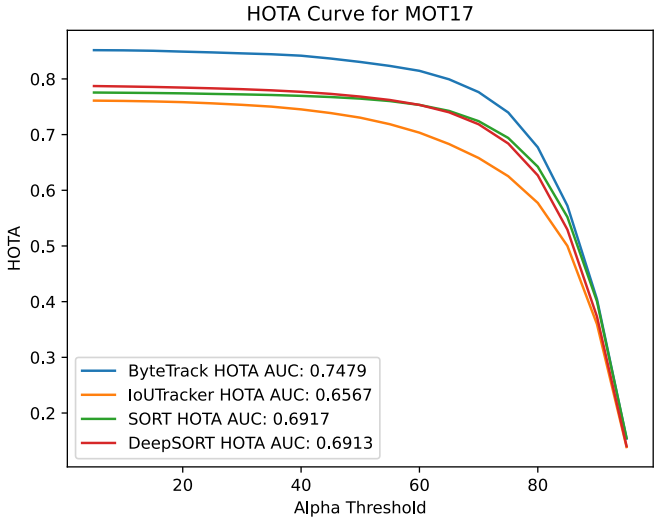
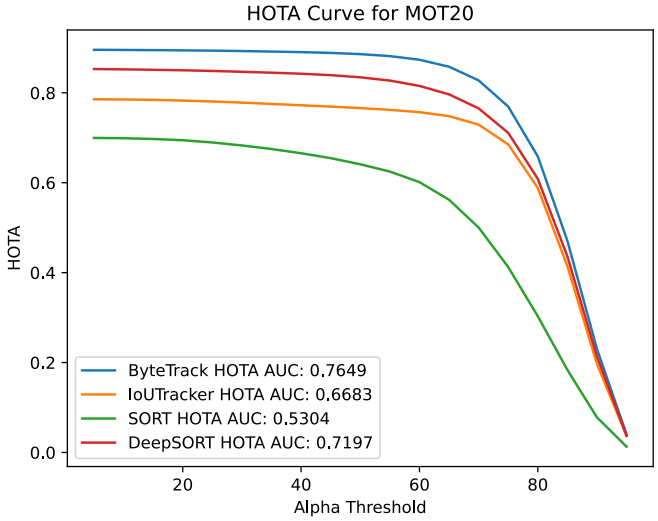


Figure 4.17: Bar chart showing the HOTA scores for all trackers for both MOT20 and MOT17. *From this evaluation, we can see that BYTE outperforms all other trackers in both datasets. It can also be seen that SORT has a significantly worse performance on MOT20 compared to other trackers. Additionally, the delta between SORT's performance on MOT20 and MOT17 is much higher than the delta of other trackers. This can attributed to MOT20's higher density environments along with SORT restricting itself to only analyzing locations. DeepSORT utilizes the same location analysis but also incorporates appearance vectors, which allows it to have a notable 19% higher HOTA on MOT20 compared to SORT. IoU Tracking scores a competitive HOTA score compared to other methods. This is due to the predictable nature of the majority of subjects in the video clips.*



(a) $HOTA_\alpha$ Curve for MOT17



(b) $HOTA_\alpha$ Curve for MOT20

Figure 4.18: $HOTA_\alpha$ **curves for both MOT17 and MOT20 training sets.** *The curves are an intermediate step for the final HOTA values shown in Figure 4.17. But analyzing these curves can give an idea how well the tracker is able to maintain its HOTA value as* α *increases. In this case,* α *is just the IoU threshold for the detection matching phase defined in Chapter 2. This essentially means the trackers are offered less and less boxes to do detection and association matching. This typically means the HOTA at higher* α *thresholds are lower. From Figure b, we can see that SORT suffers HOTA loss at much earlier* α *value compared to the rest of the trackers. This can be because of the higher pedestrian density in MOT20 compared to MOT17.*

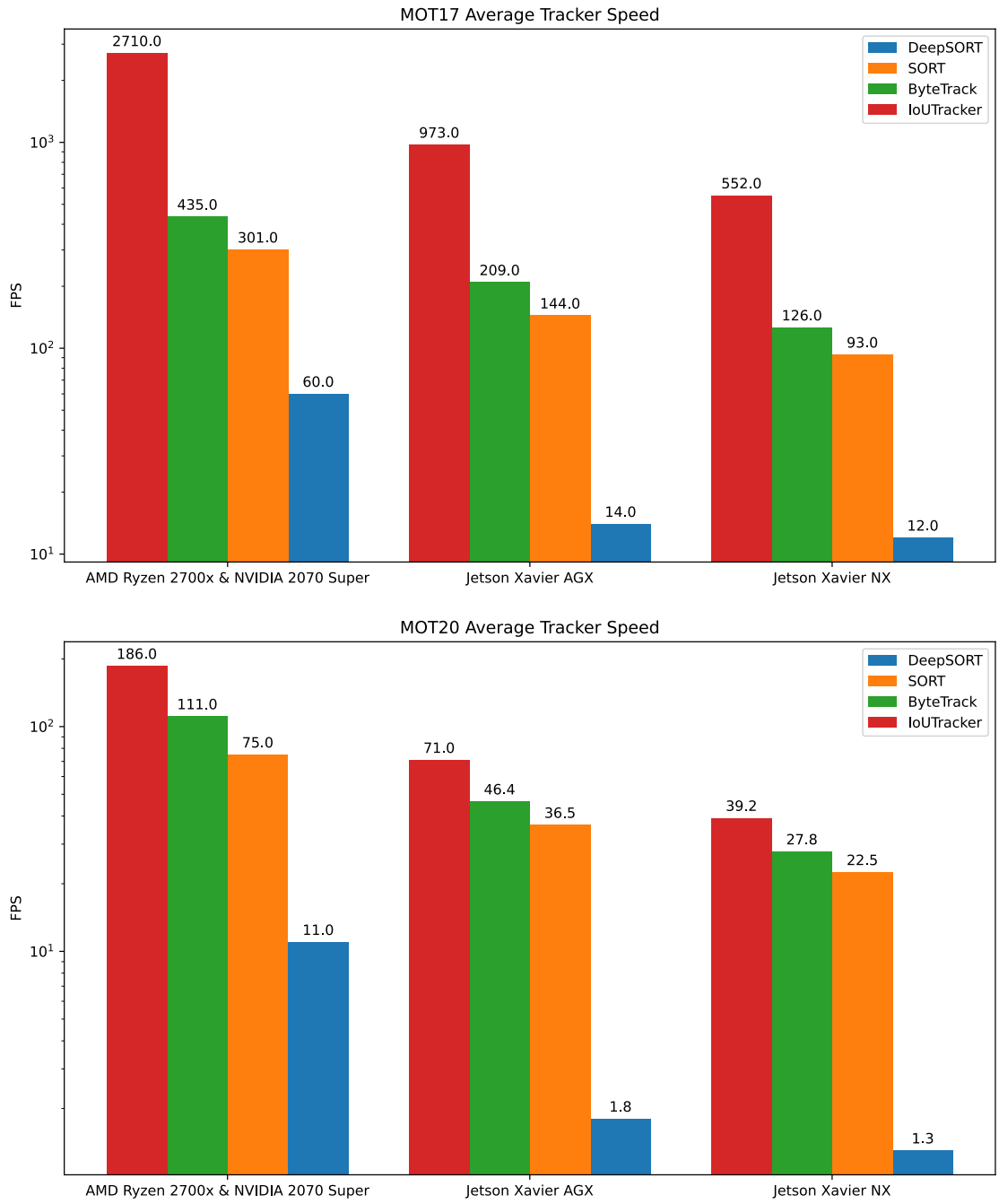


Figure 4.19: Comparison of average tracking method speeds on Desktop CPU and GPU, AGX, and NX. *For all compute platforms, all methods ran on the CPU (AMD Ryzen 2700X) and DeepSORT ran on the Desktop GPU (NVIDIA 2070 Super). The fastest method was IoU Tracker, and this is due to the lack of advanced state estimation or neural networks, which are present in the other methods. Compared to the desktop, AGX and NX run times were 2.3x and 3.9x slower for BYTE, SORT, and IoUTracker on both datasets, respectively. DeepSORT fails to keep real-time speeds on MOT20 due to its high density environments. All MOT17 and MOT20 runs for each platform and for each tracker can be found in Figures 6, 7, and 8 in the Appendix.*

Chapter 5

Data Augmentation Analysis

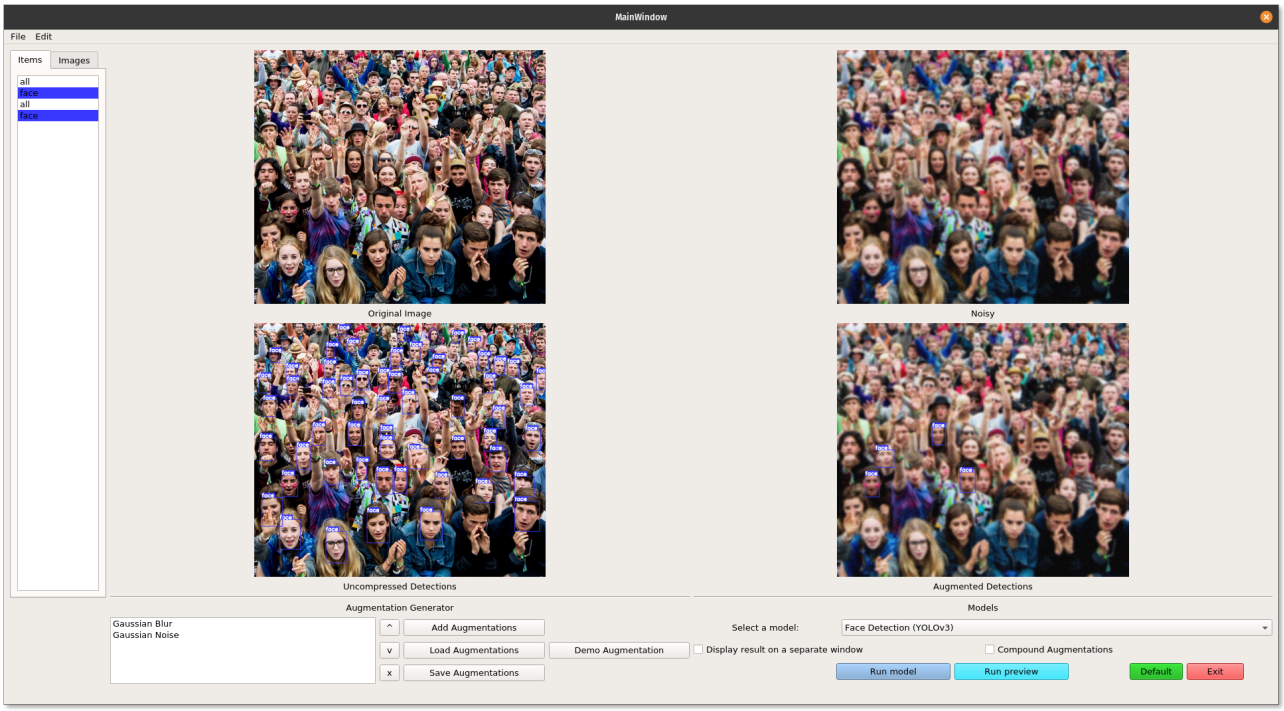
Work done by previous IRIS students and other researchers in the area of neural network sensitivity and robustness was outlined in Chapter 2. They have shown in breadth that CNN-based detectors still do not have comparable perception abilities in degraded environments compared to human perception. Advancements in training procedures and network architecture improves baseline performance as well as robustness, but there is still a notable gap between CNNs and human perception. This analysis attempts further expand this body of knowledge by analyzing more recent compression methods as well as newer networks and architectures.

An additional reason is based on previous discussion on Edge vs IoT device paradigms for a real-time system. Although it was decided to pursue an edge-based paradigm, future changes to user requirements could warrant a change to an IoT-based system. This warrants an analysis on compression since it is the primary mechanism that allows IoT devices to stream sensor data in real-time. Additionally, based on the analysis, an optimal set of compression parameters and methods can be proposed for different situations.

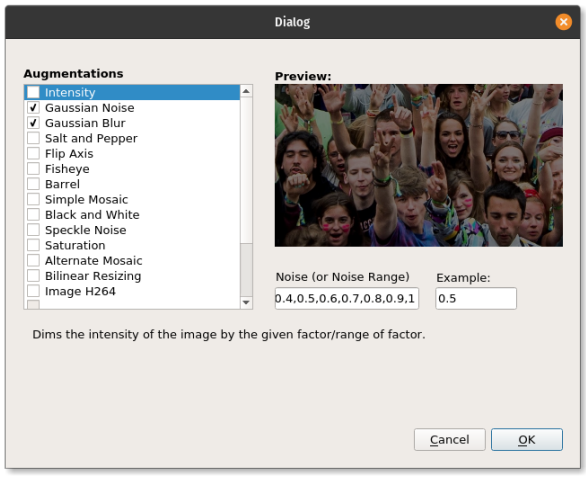
5.1 GUI Application for Augmentation Analysis

To support current and future work in data augmentation, a GUI application is created to assist in loading in images, augmentations and their appropriate parameters, and a given neural network for evaluation. The application and the main dialog windows are pictured in Figure 5.1. It was developed with PyQt5, and it utilizes PyTorch for neural network execution and OpenCV2 for image processing operations. The structure of this application was created to be as flexible as possible in terms of adding new augmentations, image datasets, and neural networks (Figure 5.2). Rus Refati as well as teams from COSC401 and 402 helped tremendously in terms of adding new augmentations and neural networks as well as refining the user experience.

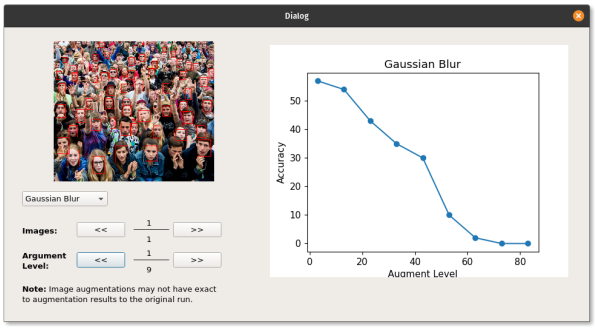
An end-user would select their choice in augmentation functions and set range of parameter values for each active augmentation in a separate Augmentation Selection window. The selection of neural network is chosen from a drop-down list in the bottom-right corner.



(a) Main window that would be presented to the user



(b) Dialog box for adding/editing/removing active augmentations



(c) Dialog box for the experiment results

Figure 5.1: Screenshot of the main window of the GUI Application for augmentation analysis. *To add or remove or edit augmentations, clicking the "Add Augmentations" button in the bottom-left corner will present the dialog window shown in (b). After adjusting augmentations, parameters, neural networks, and images, clicking the "Run Experiments" button in the bottom-right corner will run the experiment. The results are shown in a separate experiment dialog box (c).*

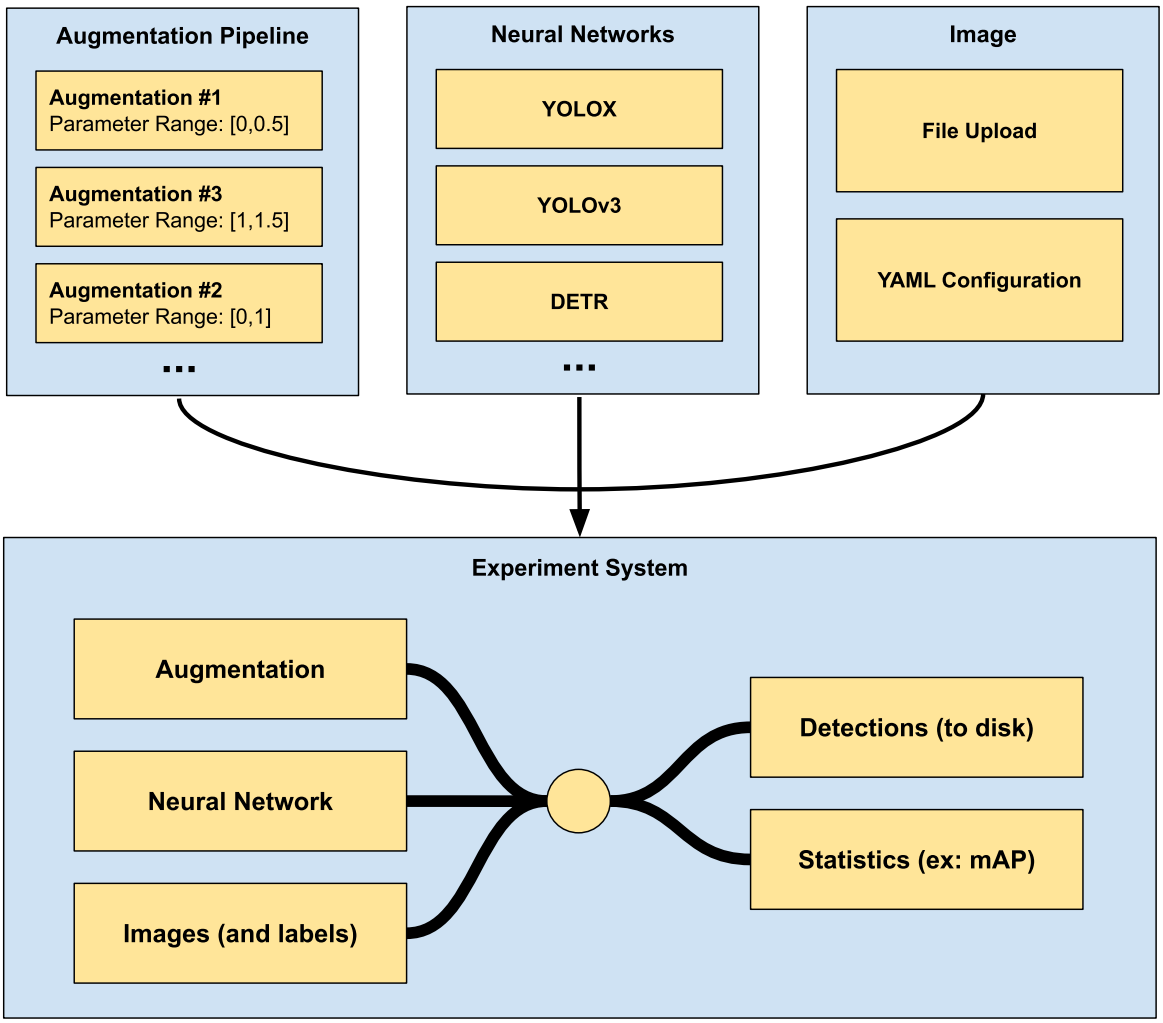


Figure 5.2: Software flow of GUI Application for augmentation analysis. *The GUI reads from three main data sources. The first source is an augmentation file that contains all the low-level functions to transform an image or frame. Each one of these functions are wrapped around an abstract class called an "Augmentation". This abstract class allows all functions to integrate into the "AugmentationPipeline" class, which controls the execution and order of all chosen augmentations. Similarly, neural networks are managed via a dictionary of network objects. Each network object is a custom class that inherits another abstract class called a "Model". This abstract class requires an inherited class to implement functions such as initialization, inference, and drawing. This is so that calls and output to each network are standardized and makes developing and process later elements much more efficient.*

Finally, the images to run these augmentations and network over can be uploaded individually via context menu on the left-hand side of the GUI or by uploading an appropriate YAML configuration file. This configuration file can allow for entire datasets to be uploaded into the GUI as well as ground truth labels. There are preset YAML files for popular datasets, such as VOC2007 and COCO val2017. Once all these experiment variables have been set, we can preview the augmentations on a given image as well as the network performance on the image with no augmentation and the same image with the augmentations on an example parameter. The core of the application is the experiments, which can triggered via the "Run Experiment" button. This runs all the experiment variables against all the loaded images. The output of the neural network is saved to that experiment's specific folder, where it can be further processed later. Once the experiment runs have been done, the GUI displays a dialog box that contains the detections at a specific augmentation level and the performance of the network on that augmentation.

For this specific compression analysis, the GUI was used to run all chosen neural networks through all chosen image augmentations on COCO val2017. The GUI saved all raw detection output for each image in the dataset as well as the mAP output from COCO's evaluation tool. Both types of items were processed with another, separate Python script to generate graphs and examples seen in the later subsections.

5.2 Compression Experiment Setup

Similar to [45], this work's compression analysis looks at JPEG and H264 compression. The analysis also focuses on mAP performance changes of object detectors. To extend the analysis done by [45], this analysis looks at additional compression methods, including WebP image compression, Bilinear Resizing, H265 video encoding, and a small neural network-based compression method. In terms of networks, the analysis utilizes SOTA and near-SOTA single stage object detectors trained on the COCO 2017 training set. Similar to how [45] did their mAP analysis on VOC2007, this analysis will evaluate the detectors on the COCO val2017 dataset and display the mAP @ IoU=0.5 as the main metric. Other similar mAP metrics, such as mAP @ [0.5:0.95:0.05], were calculated but the resulting graph trends were very similar to the former mAP metric.

5.2.1 Compression Method Selection

All compression methods were chosen for different reasons. JPEG and H264 were chosen due to their balance between file (or video) size reduction and processing time. They are also one of the most popular used methods. WebP, which was introduced by Google in the early 2010s, is intended to be a direct competitor to JPEG. A cursory study done by Google showed that WebP image files, on average, were 25-34% smaller than their JPEG counterparts with similar SSIM scores [21]. Similar to WebP, H265 is an enhancement over H264. It employs a more advanced method of analyzing video movement in order to save additional data. According to Banitalebi-Dehkordi et al., H265 outperforms H264 in terms of bitrate savings by 25.08% while maintaining comparable video quality [3]. Bilinear resizing

was chosen to act as a baseline. It is a simple method of resizing an image to a smaller size (hence the smaller the file size). The resizing method was specifically bilinear because of it provides the best balance between speed and preserving major image features [22].

Finally, the selection of a neural network-based compression method comes from the recent advancements in that sub-field. The latest compression networks attempt to learn what portions of an image to remove based on the portion's impact on image distortion [9]. Furthermore, for additional size savings, the networks also learn how to "quantize" the transmittable feature maps that represent its compressed image [9]. The "Gaussian Entropy Network" from Cheng et al. has a relatively high PSNR vs Bit-rate performance on the Kodak dataset [7]. It is also a relatively small network, with a maximum network parameter of 26,598,965, which is similar to the size of ResNet50 (25,557,032). In terms of implementation, CompressionAI, a model zoo for popular compression networks, was utilized [7].

Transmission between neural network-based, video-based, and image-based methods are detailed in Figure 5.7. For image-based algorithmic compression, the streaming-based device will send a whole compressed image over the network. A video-based algorithmic compression method will have a stream of bits representing the encoder output sent over the transmission network. This encoding needs to have some sort of decoder (also called playback) on the other device to reconstruct into video frames for display. Finally, neural network-based methods will transmit feature maps from the encoder part of a neural network. On the compute architecture, a decoder of the same network reconstructs the image for display.

5.2.2 Neural Network Selection

The networks selected are YOLOv3 Ultraltyics, YOLOX, and DETR. All three are near-SOTA, real-time object detectors trained on the COCO dataset. All three are also chosen due to their distinctive architectural composition. YOLOv3 Ultralytics is a deviation from the original YOLOv3 network but with a Sigmoid Linear Unit (SiLU) activation and an updated set of training augmentations from YOLOv5 [25]. YOLOX, as mentioned in Chapter 2, directly predicts bounding box localization rather than processing the output through anchor-matching - which YOLOv3 and other YOLO iterations have used. DETR is a real-time, transformer-based network that utilizes a typical CNN for feature extraction. It accomplishes bounding box regression and classification with a traditional transformer network. The transformer architecture's attention mechanism has been noted to be more resilient to augmentations that traditional CNNs are [44].

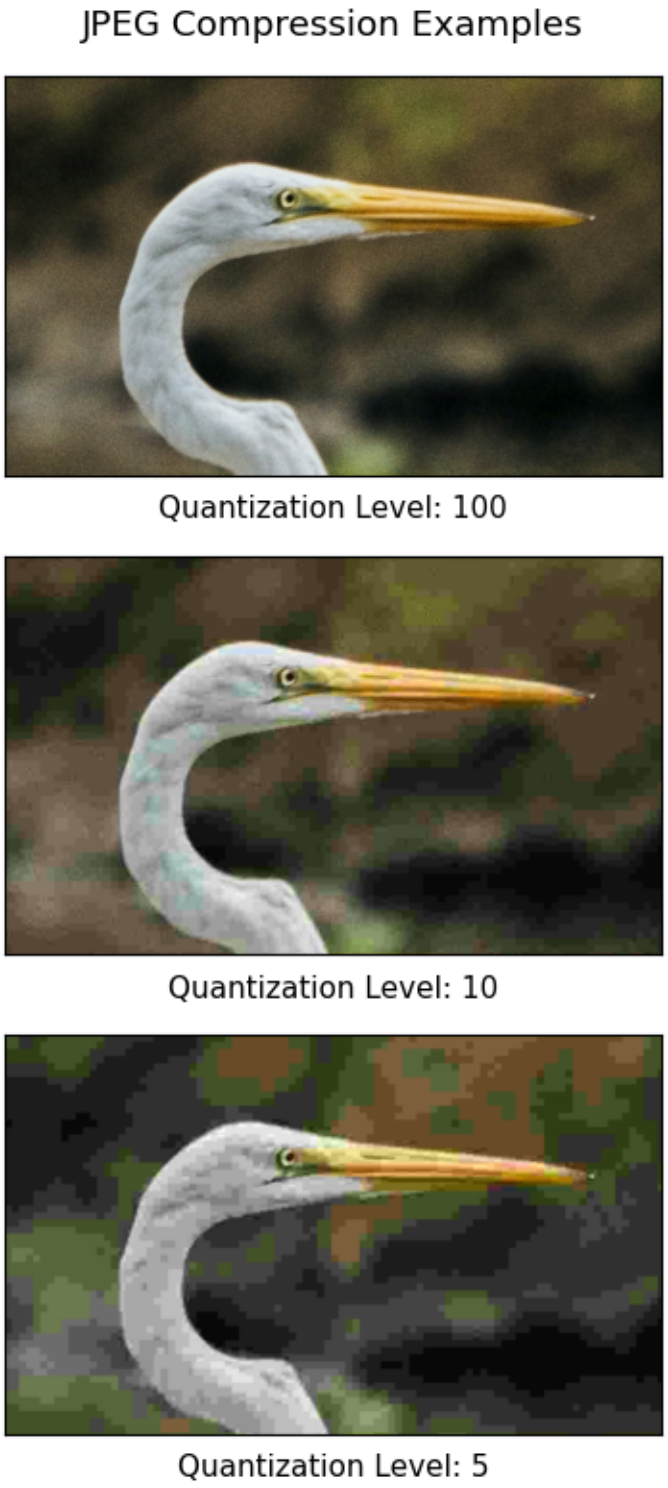


Figure 5.3: Example of JPEG compression applied to an image. *The parameter being changed within JPEG is the "Quantization Level", which is one of the final steps in the algorithm and is what decides which image features are kept.*

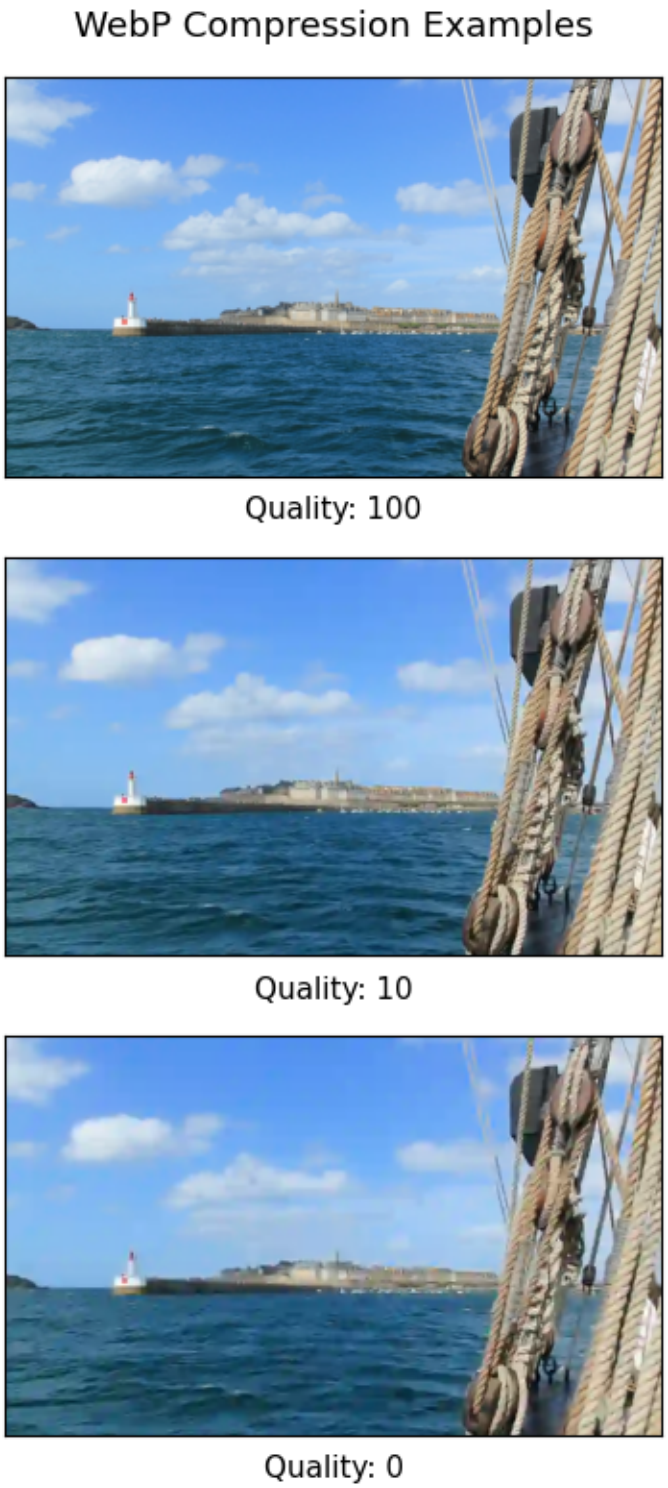


Figure 5.4: Example of WebP compression applied to an image. *The variable is the "Quality", which is similar to JPEG compression's "Quantization Level".*

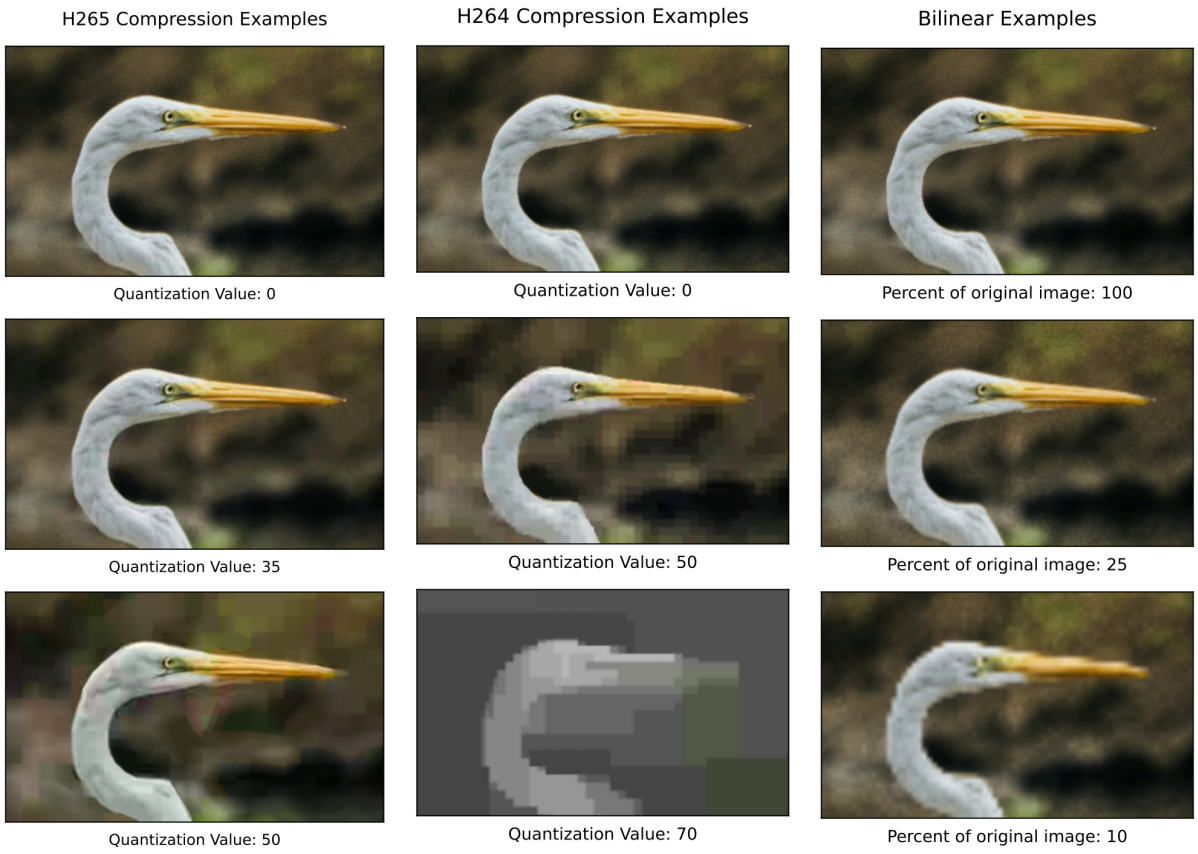


Figure 5.5: Examples of Bilinear Resizing, H264, and H265 compression to a given image. The variable for bilinear resizing is the percentage of the original width and height of the image. For H264 and H265, the typical parameter to adjust would have been the "Constant Rate Factor" (CRF). Rather than using CRF, this analysis uses the quantization parameter in order to be consistent in terms of benchmarking on COCO val2017.



Figure 5.6: Example of Gaussian Entropy Network from [9]. *The network can provide different levels of compression, but it is incorporated into the training process and network structure. Training the network would take a prohibitively long time, so the variable is the different "quality" models that have been pretrained by CompressionAI. This quality ranges from 1 to 6, where 1 produces the most compressed image.*

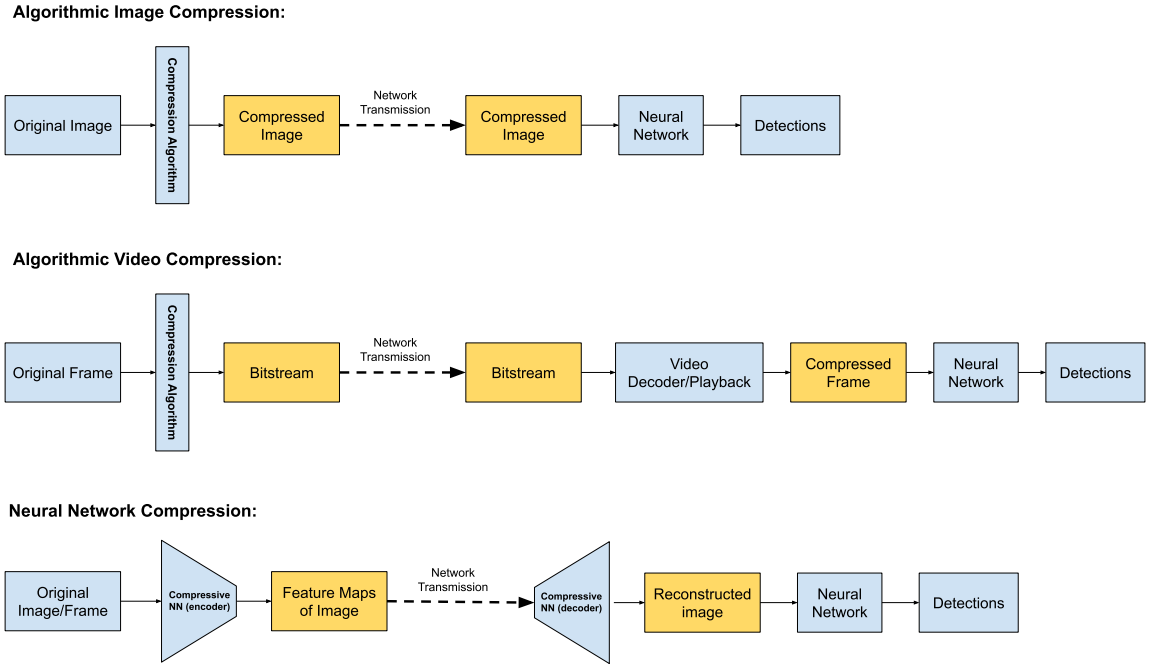


Figure 5.7: Diagram illustrating two main compression methods. *In a streaming-based scenario, the streaming device would have to compress the sensor input and send it over the network to a larger compute architecture for processing. The analysis's compression methods are broken into three different categories and concepts. Image, video, and neural network-based compression are different in terms of resulting data that is transmitted. For image-based compression, the ultimate output is a compressed image. That image itself is sent over the network. For video-based compression, the typical data that is transmitted is a stream of encoded video rather than a discrete frame. The receiving client needs to have a playback or decoder of some kind to reinterpret the H264 encoding into a frame. The Gaussian Entropy Network is made up of an encoder and decoder. The encoder network turns an image to quantized feature maps that describe the image. Those feature maps are transmitted to the decompression network, which reconstructs the image.*

5.3 Image Compression Results

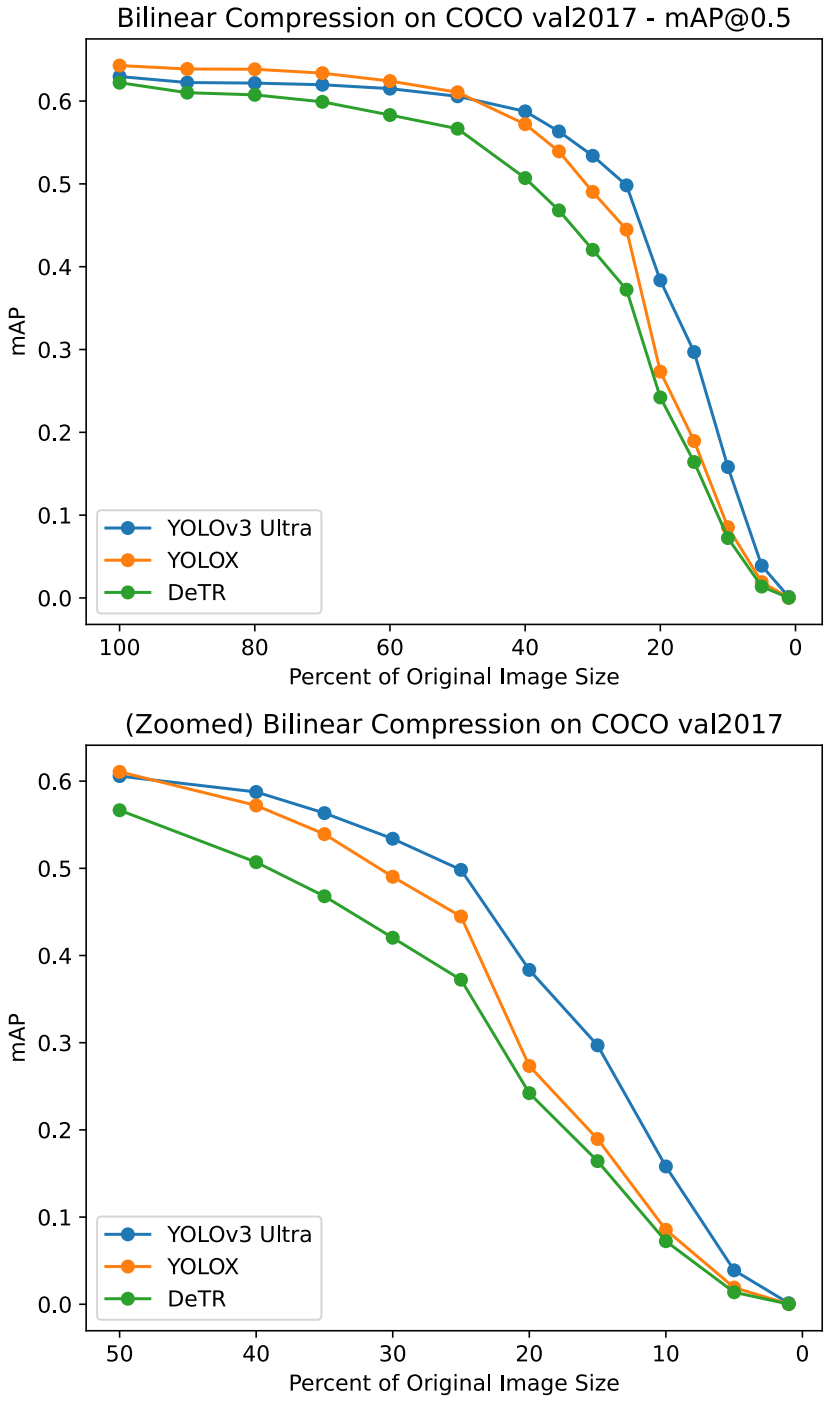


Figure 5.8: Bilinear Compression of COCO val2017 and evaluation of YOLOv3, YOLOX, and DETR. *Performance across all networks start to decrease significantly around 40% of the original image size.*



Figure 5.9: Detection examples from DETR, YOLOv3 Ultra, and YOLOX on Bilinear Resizing. *DETR is observed to produce more false positives at levels below 40% when compared to YOLOv3 Ultra and YOLOX. For this particular augmentation, this might be due to DETR being trained at large image sizes (minimum width of 800) compared to the YOLO networks (a ratio-preserved 640x640 image).*

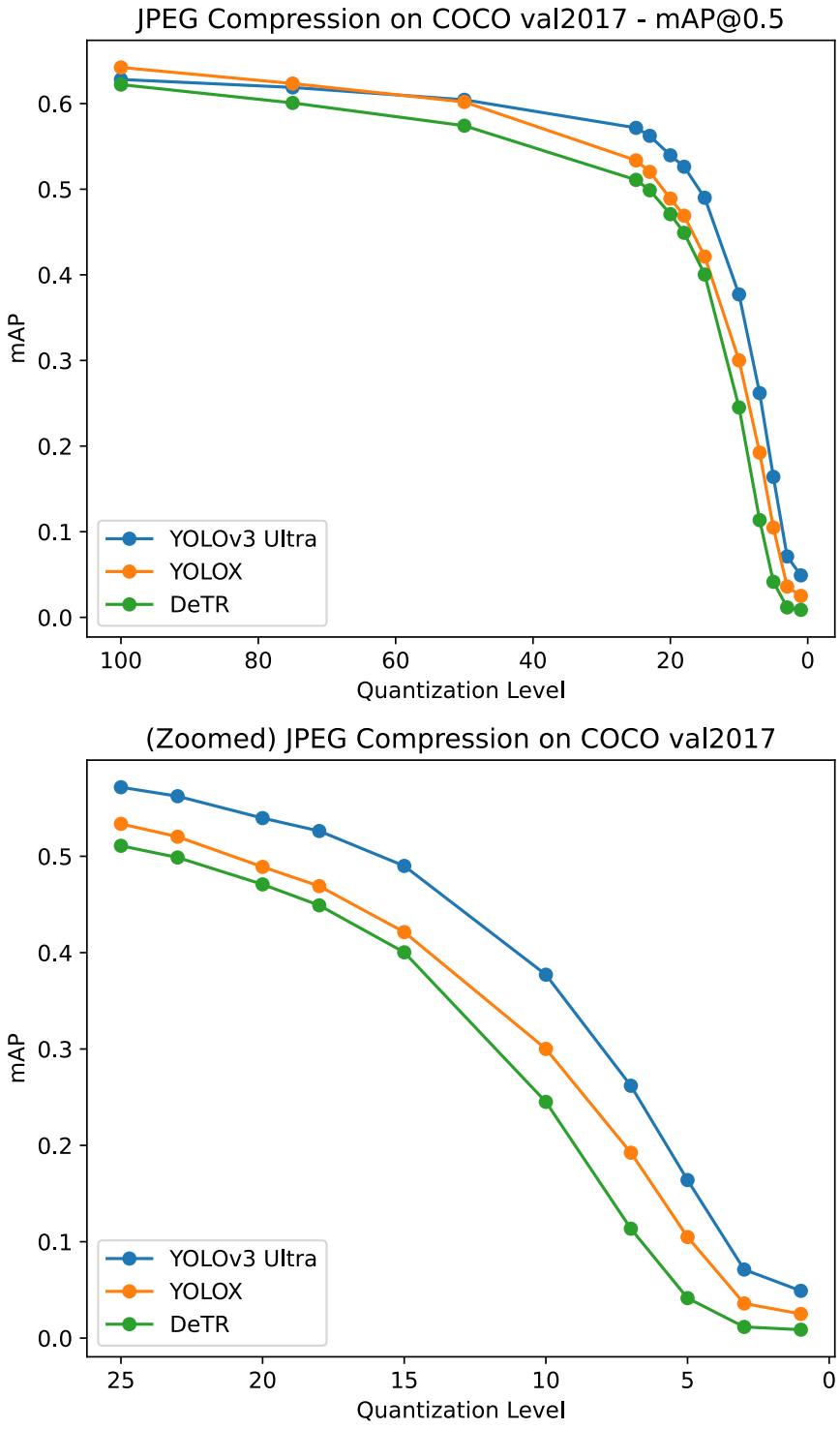


Figure 5.10: JPEG Compression of YOLOv3, YOLOX, and DETR. *The general trend is similar to the observations of [45], where the mAP@0.5 for all networks is within 10% of their original score between quantization level of 100 to 40. Once quantization level of 20 is reached, all networks see a precipitous drop in mAP. At the highest quantization level, all networks are very close to 0% mAP. This is reasonable since the compression has removed many features to an unrecognizable point.*



Figure 5.11: Detection comparisons between DETR, YOLOv3 Ultra, YOLOX on JPEG Compression. *All detections were filtered at a confidence threshold of 35%. All networks decline in localization accuracy as the quantization level decreases. Similar to the bilinear resizing, DETR produces a higher rate of false positive detections up until a quantization level of 0. At this level, all networks report little to no high confidence detections even though the batter, catcher, and umpire are recognizable to a human.*

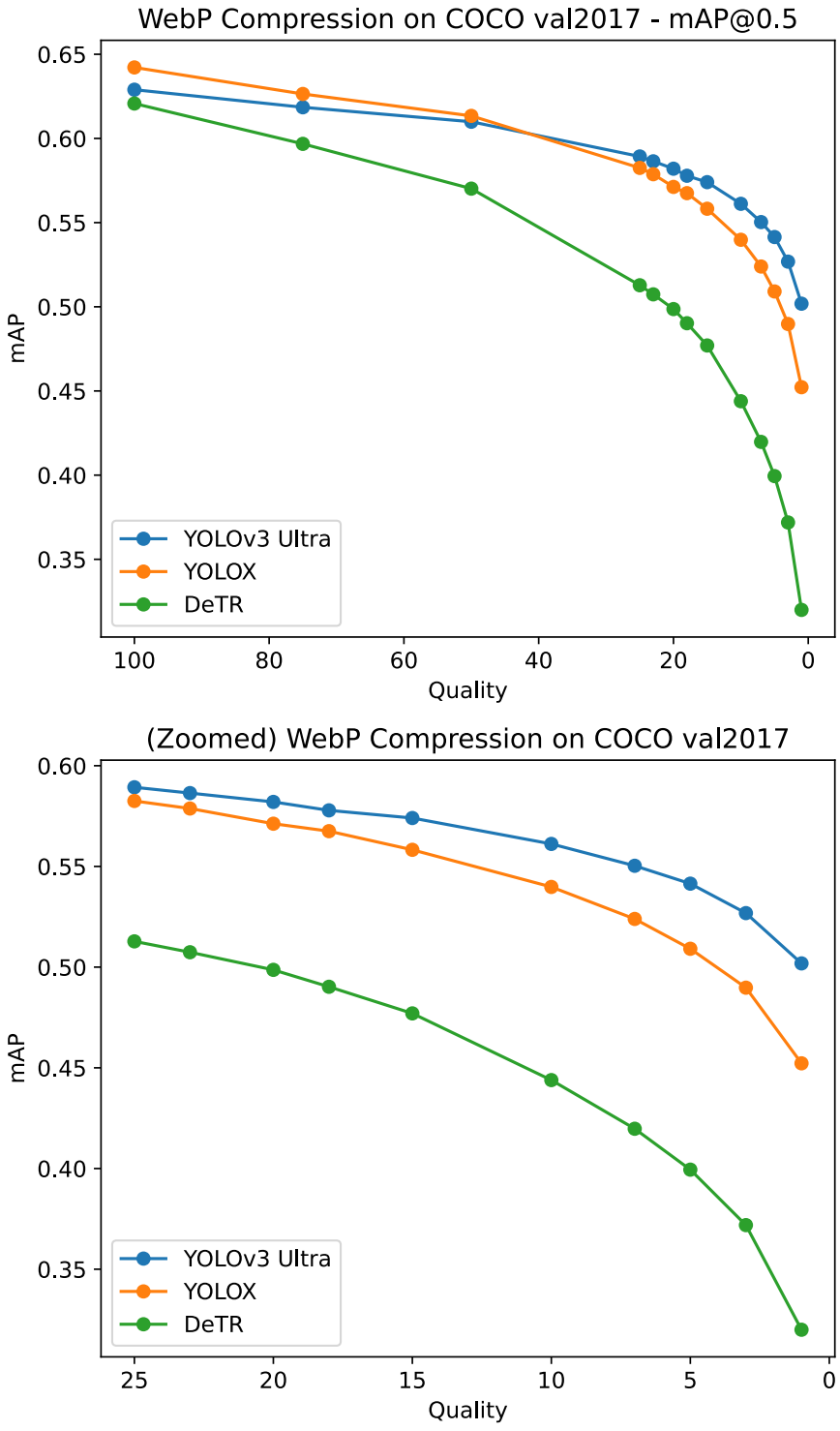


Figure 5.12: WebP Compression of YOLOv3, YOLOX, and DETR. *WebP retains more mAP performance at more aggressive compression parameters compared to JPEG. Although, the dropoff in performance for DETR and YOLOX occurs sooner than their performance on JPEG.*



Figure 5.13: Detection examples of all networks on WebP Compression. *All networks produce stable bounding boxes across all of the WebP parameters.*

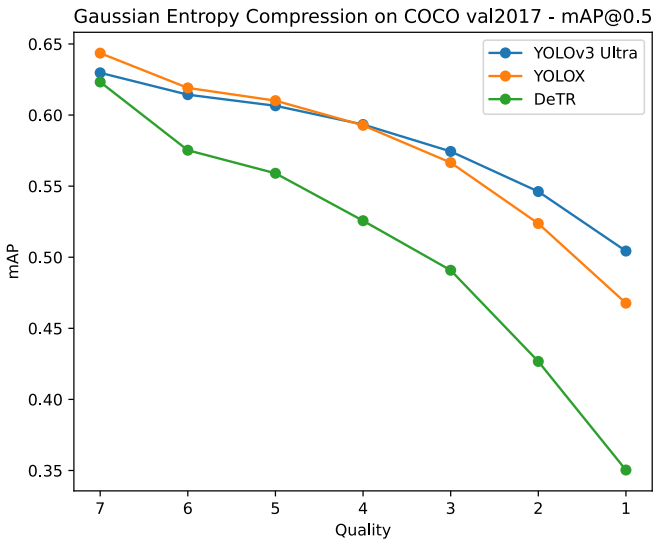


Figure 5.14: Gaussian Entropy Network Compression of YOLOv3, YOLOX, and DETR. *The Gaussian Entropy Network's mAP decline is more linear and less impactful than other image compressions.* ***Note:*** *parameter "7" is the image with no compression.*

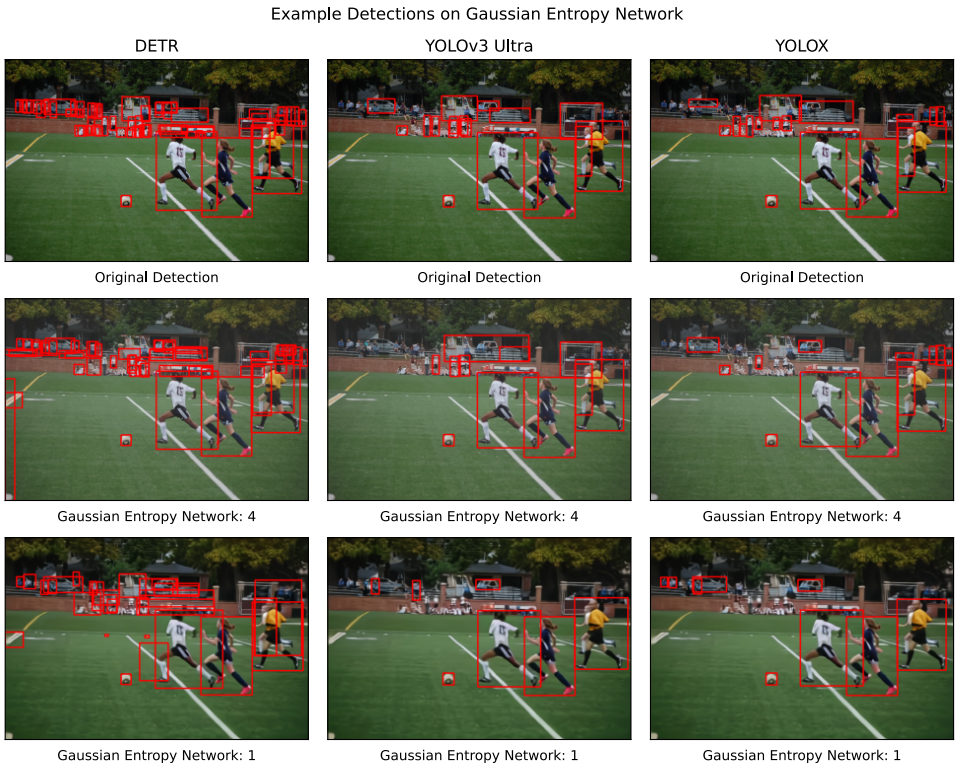


Figure 5.15: Detection examples of all networks on the Gaussian Entropy Network. *YOLO-based networks lose smaller objects, such as the soccer audience, at lower quality levels. On the other hand, DETR produces more false positives, which follows observations in Bilinear and JPEG.*

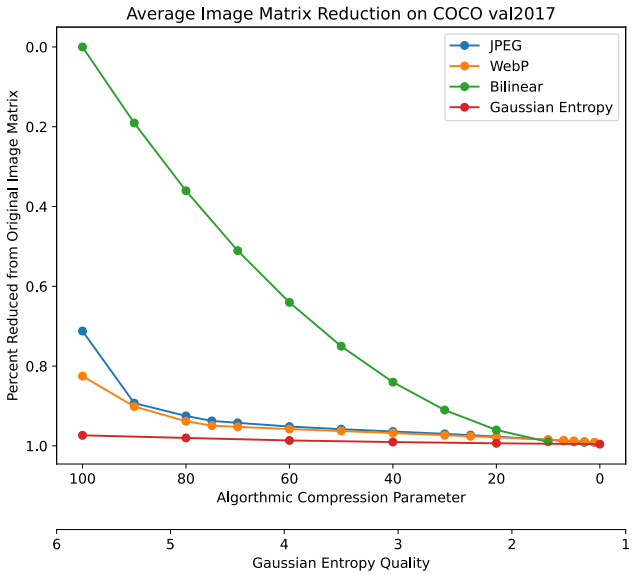


Figure 5.16: Average size reduction from image matrices in COCO val2017. *The Gaussian Entropy Network is able to reduce the original image matrix to the smallest amount of encoded data. This supports the findings in the original paper and CompressionAI's benchmarks [7] [9].*

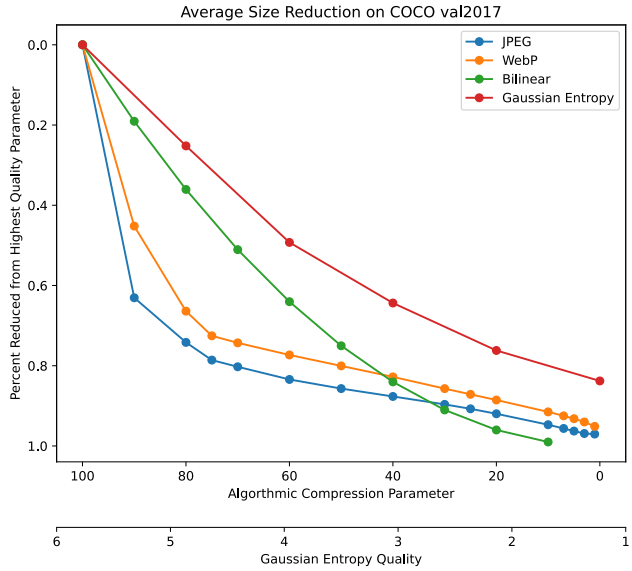


Figure 5.17: Average percent reduction of encoded image data in COCO val2017. *This graph complements Figure 5.16 by comparing the highest quality encoded data to data that is further compressed. For JPEG and WebP, the majority of reduction occurs within the lowest compression parameters. Bilinear resizing does not encode the image matrix so trend is identical to Figure 5.16 is seen here. With Gaussian Entropy Network, a quality parameter of 1 is 80% smaller than the highest quality parameter.*

5.4 Video Compression Results

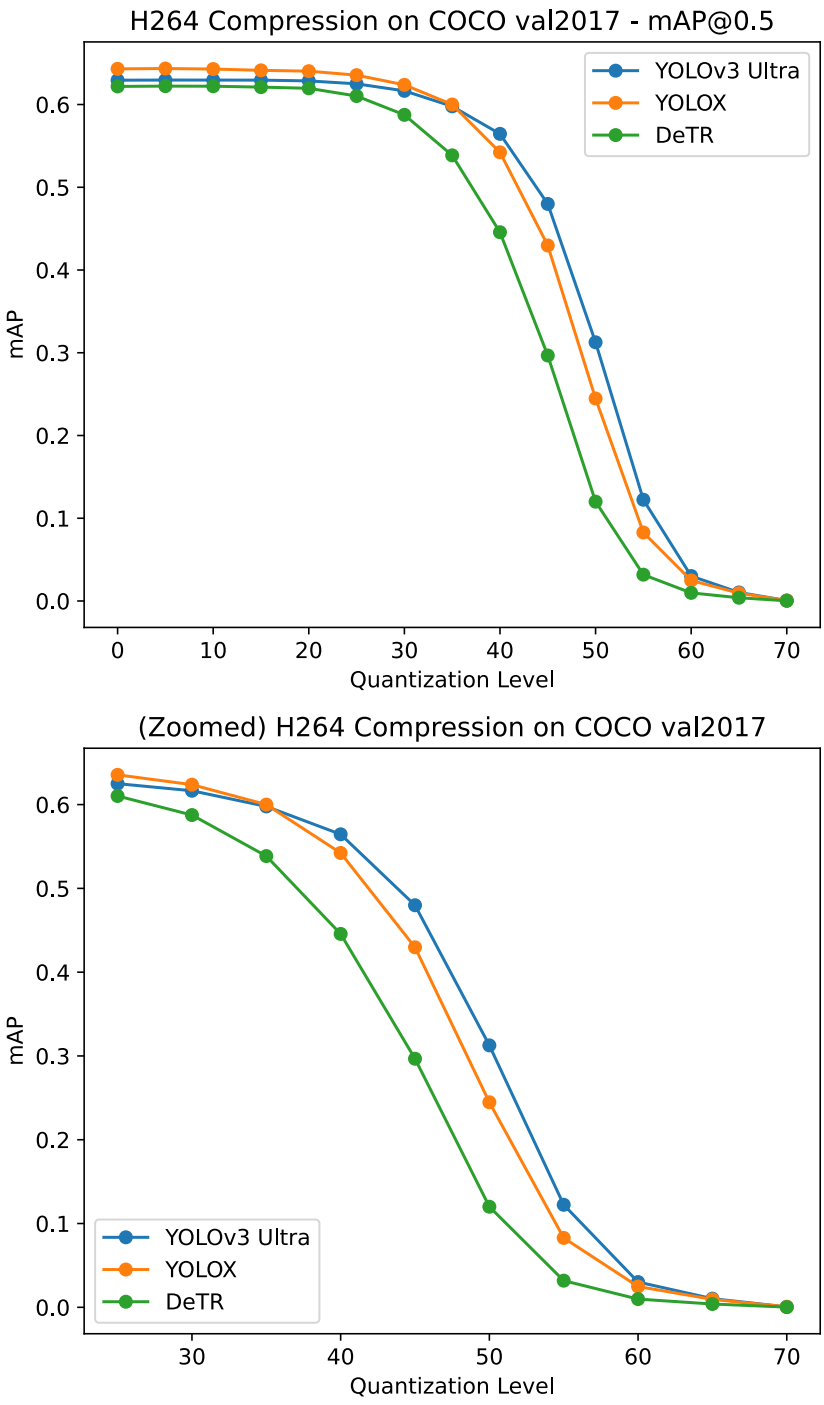


Figure 5.18: H264 Compression of YOLOv3, YOLOX, and DETR. *mAP performance degrades starting at a quantization level of 30%. This corresponds with image quality as it also degrades significantly after that quantization level (refer to Figure 5.19).*



Figure 5.19: Detection examples of all networks on H264 compression. *There is an intense decrease in image quality from quantization level of 35 to 60, which corresponds with the drop in mAP performance.*

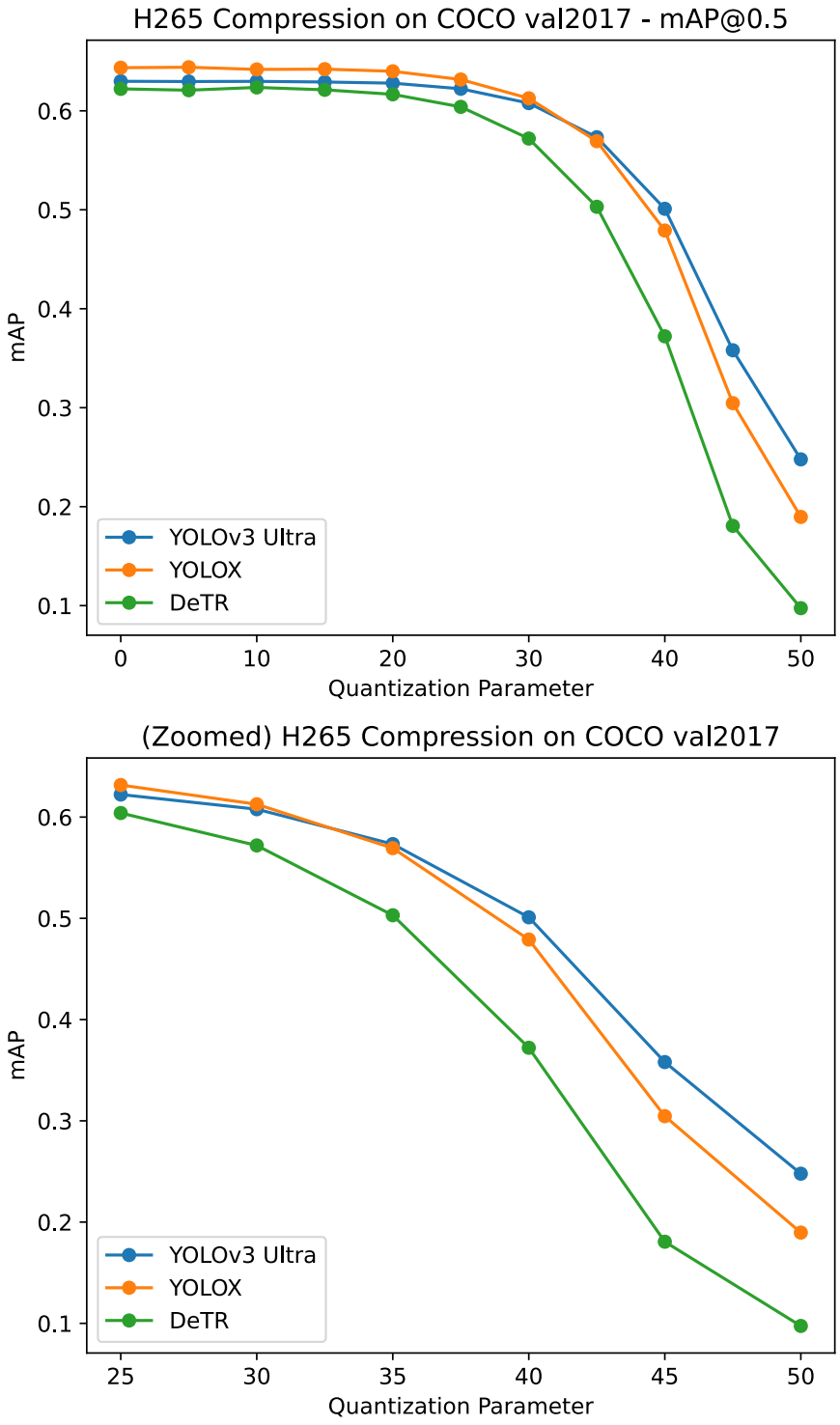


Figure 5.20: H265 Compression of YOLOv3, YOLOX, and DETR. *mAP performance is almost identical to trends seen in H264, but in the 30 to 50 range, all networks suffer worse mAP performance under H265.*



Figure 5.21: Detection examples of all networks on H265 compression. *All quantization parameters retained enough information for all networks to detect the major objects (batter, catcher, umpire, etc.).*

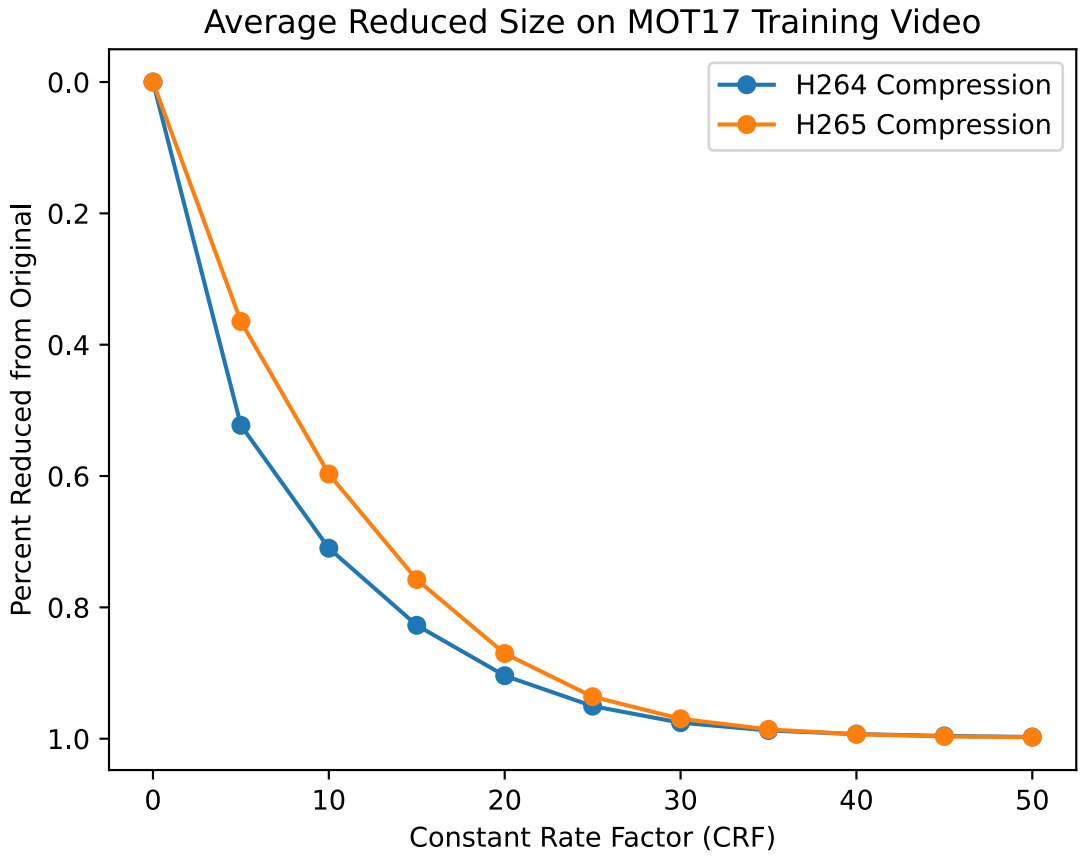


Figure 5.22: Average percent reduction of video clips in the MOT17 training set. *CRF was the variable since this is the processing of a video. But H264 dynamically adjusts the quantization parameter to be around the set CRF value, so there can be some correlation made between the reduction from the CRF and mAP performance of the quantization parameter. Similar to observations seen in Figure 5.22, H264 outperforms H265 in terms of video file size reduction. Both methods see a plateau of size reduction around a CRF of 30. Quantization levels around this level show stable mAP performance for all networks. This indicates an optimal balance of size reduction and mAP performance at a CRF of 30.*

5.5 Compression Conclusions

5.5.1 Observations

In terms of network performance, all networks showed a negative logarithmic trend for all compression methods except the Gaussian Entropy Network. This logarithmic trend can be described as the mAP performance being stable until a certain point where it would drop rapidly. DETR showed the worst mAP degradation across all types of augmentations. For example, under WebP compression, DETR ended with a 31.99% mAP@0.5 score, which is 13% and 18% lower than YOLOX and YOLOv3 Ultra respectively. Moreover, the decline trend starts sooner for DETR. Additionally, when examining example detections from Figures 5.11, 5.9, 5.19, 5.15 and 5.21, DETR created a higher number of false positive detections when compared to the non-transformer-based networks. YOLOX showed similar degradation trends to YOLOv3 Ultra, but it consistently achieved lower mAP scores at more extreme compression parameters.

In terms of image-based augmentations, bilinear resizing performs worse than all augmentations across all networks. In terms of actual image size savings, Figure 5.16 quantifies the average percent reduction of all COCO val2017 images loaded into memory with OpenCV2 at differing compression levels. Compared to bilinear resizing, both JPEG, WebP, and Gaussian Entropy reduced the average COCO val2017 image more effectively. Amongst JPEG, WebP, and Gaussian Entropy, the neural network had the highest percent of reduction across all quality ranges. The limiting factor is that this network has a slow post-processing quantization method. This is only noticeable during compression, where it runs around 1-2 FPS on a NVIDIA 2070 Super. The end-to-end compression and decompression (no quantization) is 15 FPS on a NVIDIA 2070 Super. Between JPEG and WebP, Figure 5.16 shows WebP outperforming JPEG on almost every parameter with maximum reduction delta of 11.3%. Although, it is also apparent that the size reduction difference is negligible after a quantization and quality parameter of 90. Other factors such as image quality and mAP performance must be considered. The majority of networks experience less mAP degradation on WebP compared to JPEG Compression. Qualitatively, WebP shows higher image quality compared to a JPEG image at a similar compression level (Figure 5.4 and Figure 5.3).

For video-based augmentations, H264 and H265 exhibited very similar mAP behaviors across all three detectors. The mAP scores were stable up until a quantization parameter of 30, but the mAP drops to near 0% by quantization parameter of 50. Qualitatively, Figure 5.5 shows that H264 at a similar quantization parameter shows an extensively "blocky" image. Contrastly, H265 at the same quantization parameter can preserve more image quality.

5.5.2 Practical Applications

In terms of practical applications, the ultimate choice is highly dependent on the data being streamed and the streaming device's specifications. For example, image-based compression methods are useful for sending data that is not continuous or streams with unstable FPS. They also have less computational overhead compared to video-based methods if there are no

dedicated video encoders. Furthermore, specific image-based methods like bilinear resizing can be useful for microcontrollers (ex: Raspberry Pi Pico), where compute power might not meet the minimum for methods like WebP. Usage of neural network-based compression methods show promising compression ability while preserving image quality.

Unfortunately, the speed is not applicable for real-time applications unless another encoding method is utilized on the image output. **Between image-based methods, WebP at a quantization level of 50 will save you 95.7% of the original image matrix while mAP performance only degrades by a maximum of 5%.** If power management or hardware selection limits compute power, WebP might not be the optimal solution since the encoder has been shown to be 6x slower than JPEG [49]. In this case, using JPEG at a quality level of 50 could yield similar mAP performance and size reduction but worse image quality.

Video-based methods are necessary for sending continuous video data, and these methods are highly efficient when ran on dedicated encoders. For both H264 and H265, there is little to no advantage in a CRF higher than 30. Similarly, the quantization parameters around and before 30 show stable mAP performance across all networks. If we limit ourselves to a maximum CRF of 30 and an average quantization parameter near 30, there is no notable advantage between H264 and H265 in terms of network performance. In terms of subjective image quality, from Figure 5.19, 5.21, and other works, H265 produces better looking frames across almost all quantization ranges [3]. In terms of empirical image quality measurements (SSIM, PSNR, BRISQUE, etc.), H265 produces higher frame quality compared to H264 [4]. **Because of these image quality-based reasons, streaming-based systems should utilize H265 at a CRF of 30 for video-based streaming.**. There is a caveat to this conclusion. According to [4], computational complexity for H265 is on average 51% higher than H264. **If your streaming hardware does not have a dedicated encoder for H265, then utilizing H264 would be the optimal choice**.

Chapter 6

Prototype Development

6.1 Hardware Flow

As mentioned in Chapter 3, the hardware layout for this system consists of a compute device, display, and sensor inputs (Figure 6.1). The different iterations are shown in Figure 6.3. During the development of the major software components, the hardware flow was only present in the form of bare interconnections (Figure 6.3a). Once the software had come to a certain maturity level, we developed an enclosure as a step towards the goal of a hand-held form factor.

The enclosure was developed with a CAD software called Fusion360 and was split into two parts that snap fit together. Because the prototyping needed to be quick, components were 3D printed with standard PLA plastic. Additionally, the Jetson Xavier NX had the fan head removed and replaced with a large thermal pad. To transfer heat off the thermal pad, we have two low-profile fans. One fan is directly on top of the thermal pan and blows air outward across the thermal pad. Another fan is placed on top of the VMD3 and also blows directly across the thermal pad.

This current system iteration does not contain any battery power, so all power and connection wires are fed outwards from two port holes - one located at the bottom base and the other on the side. The bottom base port hole was designed for the VMD3's power and ethernet cables. The side port hole was designed to easily plugin power, ethernet, HDMI, and USB devices into the NX. The enclosure was also designed to account for both RADAR and camera sensors. There is a simple circular punch-out to account for the IMX519. Additionally, a 0.5mm well inside the enclosure was created for a better fit for the VMD3 and reduction in RADAR absorption by the case.

6.2 Software Flow

For the integrated system, many different processes are implemented to achieve detection, heartrate estimation, and respiratory estimation. Figure 6.5 shows a general and detailed

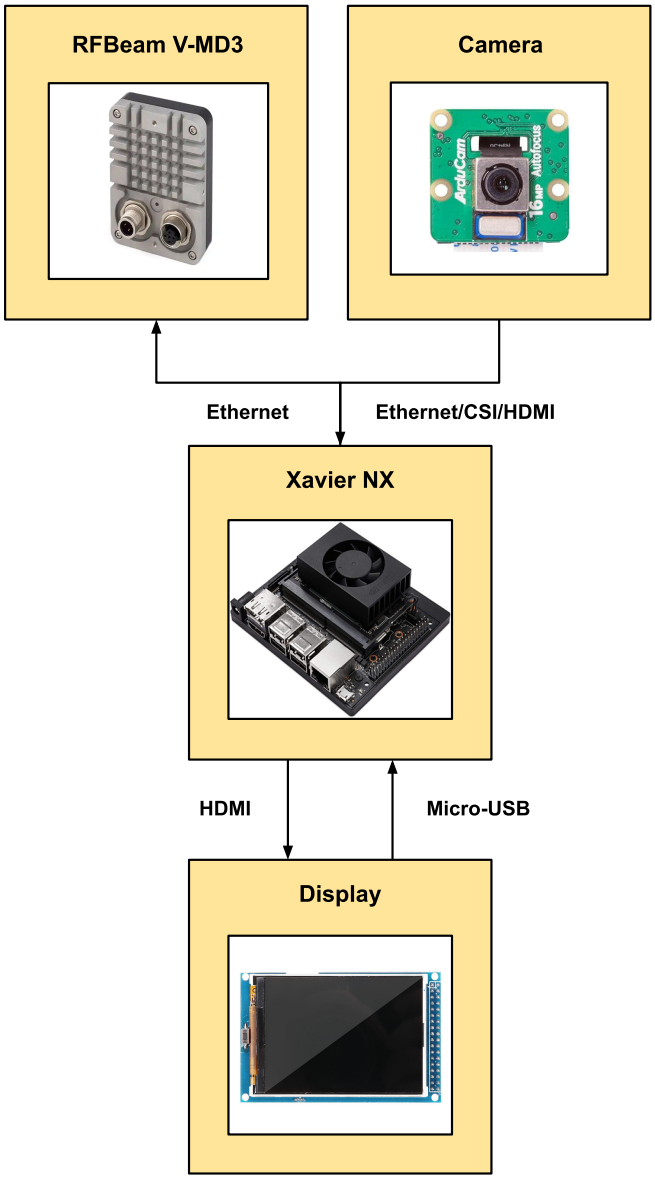
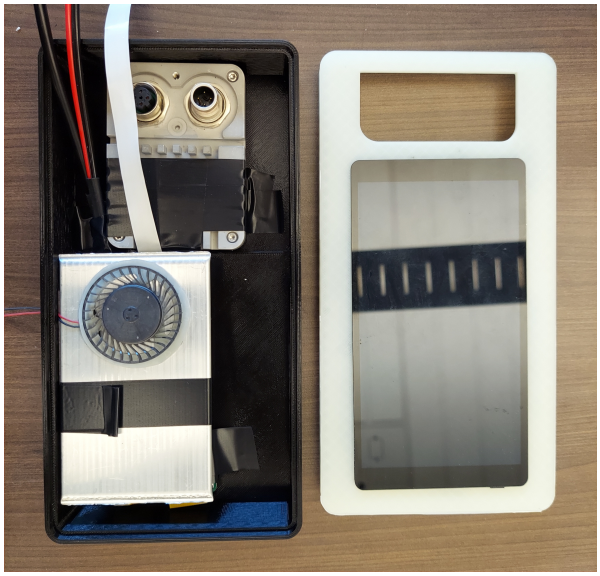


Figure 6.1: Hardware diagram showcasing the specific devices used in the final prototype. *Reiterating the decision made in Chapter 3, we utilize the Jetson Xavier NX module for the main compute device. It is connected via ethernet to the RFBeam VMD3 and via CSI to a IMX519 camera. A 5.5inch AMOLED display is also connected via HDMI to the Xavier NX.*

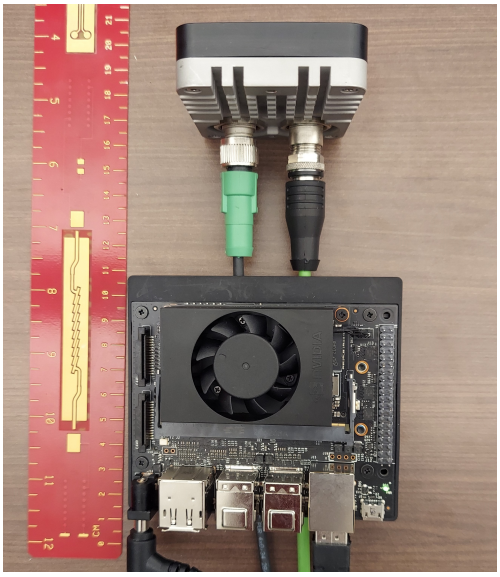


(a) Latest iteration design of prototype enclosure

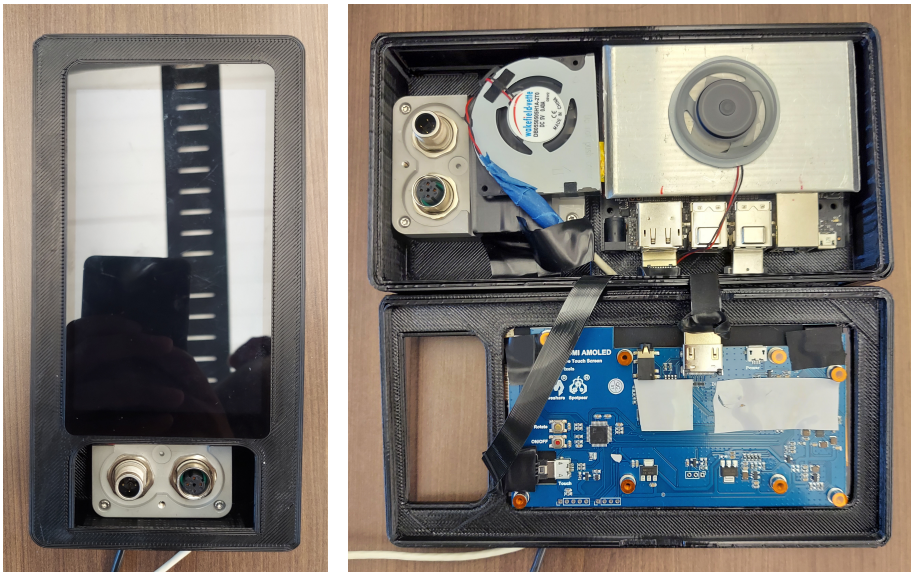


(b) Internals of the prototype enclosure

Figure 6.2: Images of the prototype enclosure for vital-sign estimation. *We replaced the NVIDIA Jetson DevKit shown in Figure 6.3 with the NCB00. The frame that holds the AMOLED screen was also heightened to give more room to the low-profile fan on the NCB00.*



(a) Initial prototype presenting with no display or enclosure.



(b) 2nd iteration with enclosure, headless Xavier NX Devkit, small cooling system, and display.

Figure 6.3: Diagram showing the different prototype iterations. *The first prototype was a simple connection of the different core devices. The second prototype had a rectangular enclosure that housed all the major components as well as a cooling system. Both iterations utilized a Jetson Xavier NX DevKit.*

breakdown of the data processing within the system. This section will break down each component and describe their implementations.

6.2.1 TCP/UDP Interface

But, before any of those processes can actual run, there must be an interface between the RADAR device and the compute device. As described in Chapter 3, the VMD3 provides a convenient ethernet-based system to send commands to and receive RADAR output from. The compute device sets up a connection with the VMD3 over TCP. Once the connection is established, the compute device can tell the VMD3 to start recording, stop recording, or change configuration settings. Once this TCP command is sent, the VMD3 responds back in TCP with a confirmation packet. Afterwards, the RADAR data is streamed via a UDP connection.

This communication interface was mainly written by Abdel-Kareem Moadi. Because this component can be initialized asynchronously to the rest of the component pipeline, we place the communication interface on a separate process. This process is spawned and executed at the beginning of the main process. Once TCP communication is established, the process waits for other processes to finish their initialization via a flag. Once the flag is set by the main process, the communication process will stream UDP data, preprocess them, and write them to a SharedArray for zero-copy transfer between it and the main process.

6.2.2 Minimum Variance Distortionless Response

A GPU-accelerated Minimum Variance Distortionless Response (MVDR) algorithm was ran on the main thread of the compute device. The shared data from the communication interface is copied and designated as GPU-bound memory. Although the copy operation is expensive, this is necessary because MVDR transforms the RADAR data matrix but the data needs to be preserved for later operations regarding the heartrate and respiration estimation. It should be noted that the transfer operation between the CPU and GPU is quickened with the shared memory pool. MVDR produces a 2D matrix that will be fed into the detection method as an image.

6.2.3 Detection Method

Based on the work in the previous chapters, we employ a neural network-based approach to localize subjects. More specifically, we utilize YOLOv5n because it provided competitive mAP performance and the lowest parameter size in Chapter 4. To get a neural network running on a Jetson platform, TensorRT is utilized to optimize the network structure to the specific hardware target. For this system implementation, we utilize a YOLOv5 TensorRT implementation from the TensorRTX Github repository. This project gives a network structure in C++ using pure TensorRT API calls. This can remove any incorrect in network structure caused by automatic conversion tools, such as the ONNX-to-TensorRT pipeline. The PyTorch YOLOv5n weights are loaded into the C++ network structure and processed through the TensorRT Builder class, which produces a serializable "engine" that

is saved to the disk. Because other components of this system run on Python, we utilize a shared object library to load some C++ functions into Python. TensorRT Python API and CuPy are used to load in the engine, preprocess the image, and retrieve the results from the engine.

6.2.4 Multi-Object Tracking

Similar to the HOTA analysis done on optical data, we also run the same HOTA analysis on the Range-Velocity and MVDR-based data. The detection boxes that were fed into the tracking methods were the ground truth boxes. This was done to focus on the association and localization ability of the tracking methods themselves rather than a given detector's localization ability.

Because of the nature of dropped signatures, the tracking performance of all trackers were lower than the optical data. Additionally, the appearances of RADAR signatures are essentially pointless since the appearance can change significantly based on object position and material. This is the primary reason why DeepSORT's performance is very similar to SORT. The method relied more on the SORT mechanism rather than the feature extractor. As seen on the optical HOTA analysis, BYTE was able to outperform the other methods. We integrate the predicted tracks into a class system that can maintain a persistent vital sign profile for a given person. In other words, we created a "Person" class that is initialized with a new track id. This "Person" object stores current and previous locations as well as other data related to vital sign estimations. Each object is stored in a list and is removed after its corresponding track has not been seen in a number of frames.

6.2.5 Phase Shift

Although this is not the focus of the thesis topic, for the sake of completeness, the pipeline utilized Online AMTC to calculate a subject's heart-rate and respiration-rate [62]. The algorithm allows the tracing and analysis of weak frequencies in two fashion - online and offline versions. The Online AMTC function requires the location of a given subject as well as their computed results from previous timesteps (if applicable). This function is run on a separate process and runs sequentially across all active Person objects that were sent into its process, so as there are more subjects, the slower the AMTC process can take.

6.2.6 Extended Functionality

GUI-Based Streaming Data

As discussed in Chapter 3, streaming data from the Xavier NX to other devices can enhance the functionality of the prototype. A first attempt at delivering real-time data, such as video streaming, was a desktop application that ran on another device. This device would be directly connected to the Xavier NX via ethernet connection. When the desktop application is launched, it will attempt to communicate with a small program on the Xavier NX that coordinates communication to the external device and other tasks, such as launching a neural

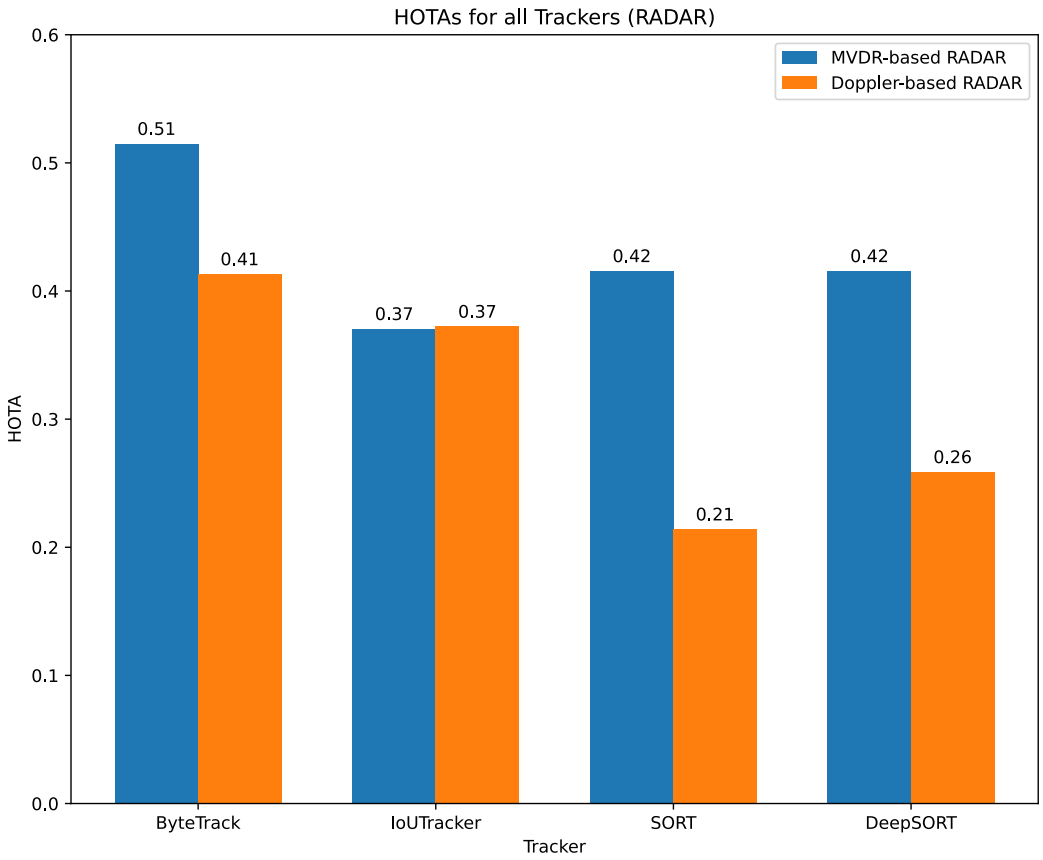
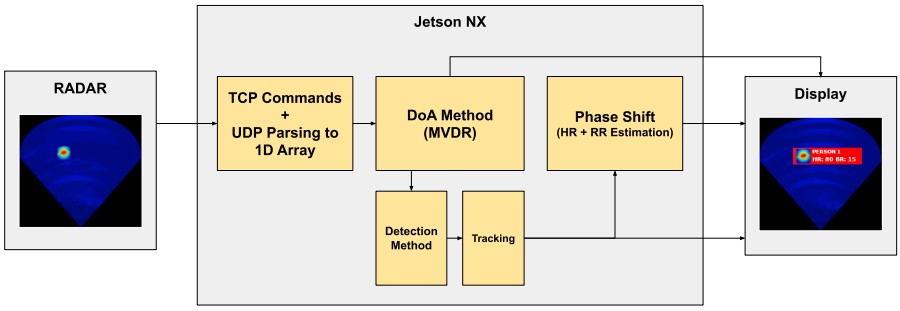
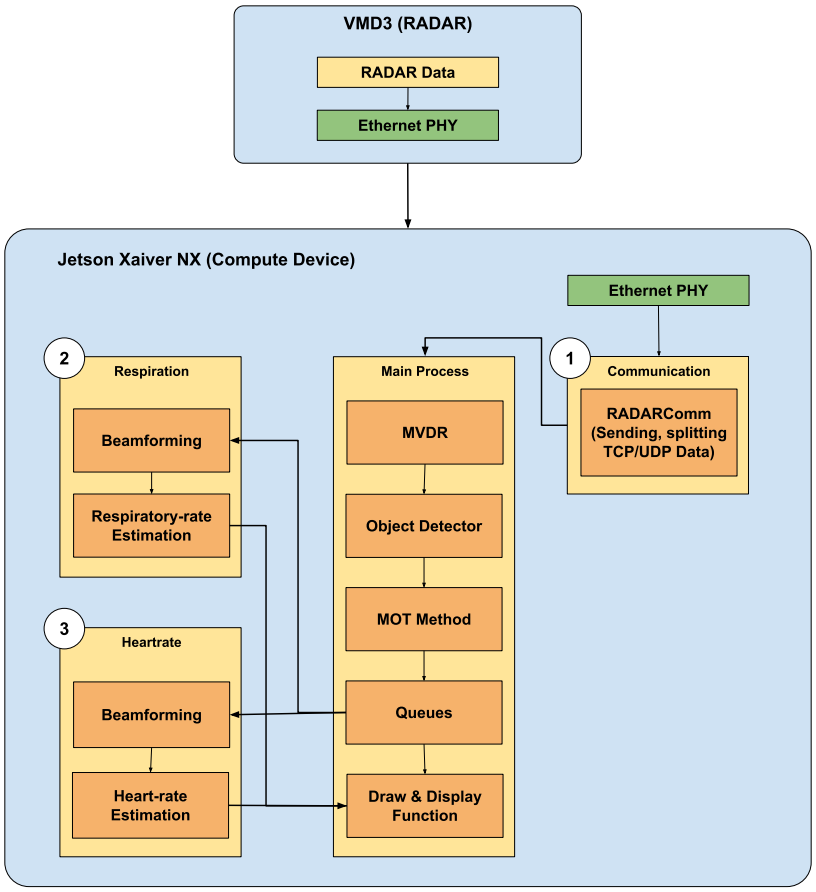


Figure 6.4: HOTA performance of tracking methods on Range-Velocity and MVDR-based RADAR datasets. *DeepSORT does not benefit from encoding any appearances of the RADAR signatures due to their consistent changes. BYTE still proves to be the most efficient at tracking MVDR videos. Although, the overall HOTA scores are significantly lower than in optical methods. This can be because of detections not be temporally stability, causing excessive track ID creation.*



(a) General pipeline of software components of RADAR system.



(b) Detailed diagram of software process for real-time processing.

Figure 6.5: High-level and low-level breakdown of system software components. *At the launch of the integrated system, the program starts with the "Main Process", which creates other asynchronous processes. Each asynchronous process is labeled with a number to represent which order they are started and a header for their general purpose. The "Communication" module receives the VMD3 data and writes to a pool of shared memory. Once a set number of frames have been collected by the VMD3, it is processed for targets in the main process. The "Respiration" and "Heartrate" modules take the angle and range locations to compute their respective outputs for each target. The outputs are sent back to the main process for visualization.*

network or streaming data out.

Once the desktop application knows that the Xavier NX is ready to communicate, any of the demo button presses will send a TCP packet to the coordinator program within the Xavier NX and execute the proper model. Both the host computer and the Xavier NX would create pipelines to efficiently feed camera input into the network, stream and receive image output via H264 UDP. The Xavier NX's pipeline was constructed with gstreamer since it was supported by NVIDIA with custom GPU-accelerated plugins for H264 encoding. On the host computer side, gstreamer was difficult to install on operating systems like Windows. So, as an alternative, we utilize FFMPEG and Pythonic bindings via ffmpeg-python to receive and decode the UDP stream.

Web-Based Streaming Data

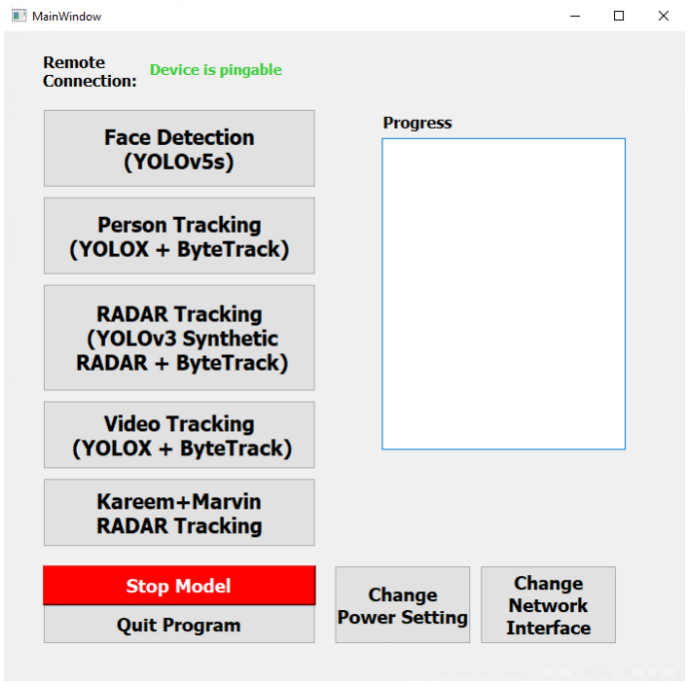
Although the "GUI-based Streaming" method provides a direct way of other devices communicating with the Xavier NX, a major downside is the difficulty that comes with desktop applications. This includes dealing with inconsistent support for libraries across different operating systems. Additionally, the PyQt5 base that powers the GUI does not support mobile devices.

To work around this and add additional functionality, a webpage is created that can stream data (video, text, etc.) coming from the Jetson Xavier NX to another device's browser. This is an internally-served webpage that is served from a small webserver running on the Jetson. The streaming protocol uses WebRTC, which is an API that enables browser-to-browser communication [18]. In order to keep the communication "real-time", the protocol enables browsers to talk directly to one another ("Peer-to-Peer" communication). To get this web-based streaming system working in a local system, VidGear, a flexible streaming library for Python, was used [52]. Specifically, a VidGear class called WebGearRTC was modified to change the default webpage styling and content as well as add multiple, simultaneous RTC streams.

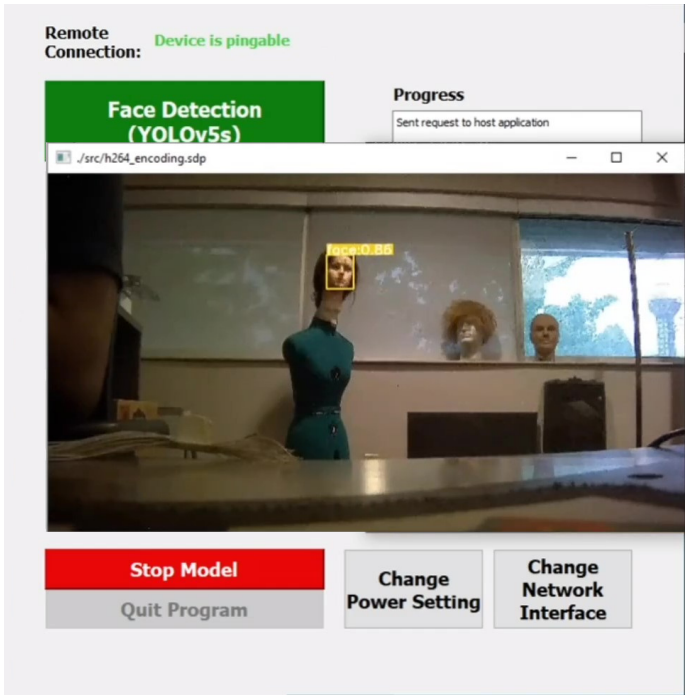
6.3 CNN Quantization (TensorRT)

6.3.1 General Quantization Process

TensorRT supports Post-Training Quantization (PTQ) from FLOAT32 to FLOAT16 and INT8. The typical flow for PTQ to INT8 under TensorRT starts with creating a definition of a given network in TensorRT. The network definition can be done in various ways, such as rebuilding with TensorRT C++ API calls or utilizing automatic conversion tools for PyTorch, TensorFlow, or ONNX platforms. The next step is to create a calibration dataset. This dataset typically contains a reasonable representation of all classes, objects, and scenes that the network was trained on. Once the calibration dataset has been created, TensorRT can quantize the network definition properly to INT8.



(a) Main GUI when device is connected to a device



(b) Main GUI with FFMPEG streaming

Figure 6.6: Illustration of GUI-based streaming method. *The GUI consisted of a series of demos that could run on the Quark NX, and the GUI communicated with the Quark via an Ethernet system. This same connection system would later be employed with a web-based streaming system.*

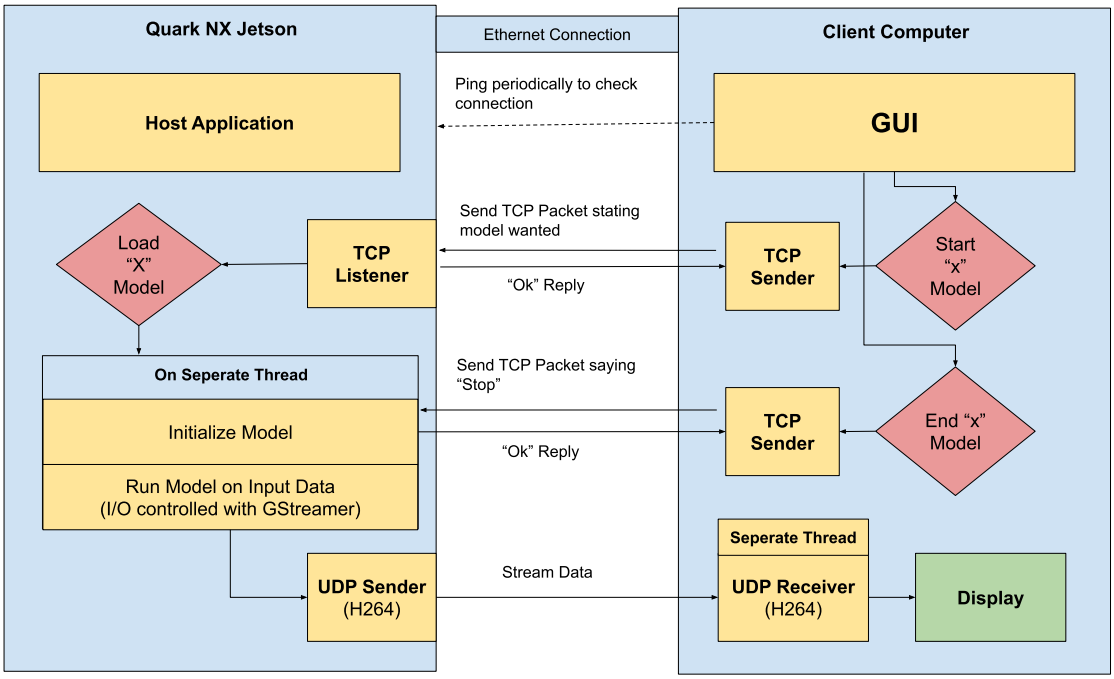


Figure 6.7: Diagram flow of streaming via ethernet between a Jetson device and a "Client" Computer. *The Quark NX was the main use case for this since it is "headless" (i.e. no display outputs). The streaming, inferencing, and command-listening components on the Quark side happen on their own separate threads. Streaming is enabled via GStreamer and NVIDIA-accelerated H264 encoding. Receiving the stream uses FFMPEG's RTP streaming protocol.*

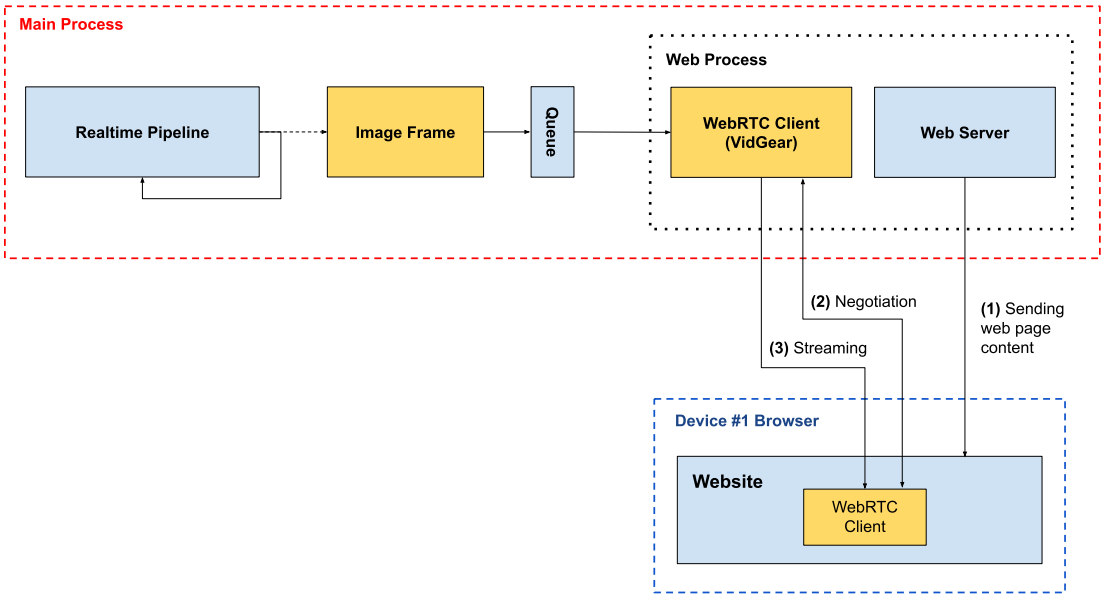


Figure 6.8: Diagram showing current web-based streaming system. *A web process handles two things - the streaming protocol (WebRTC) and handling web traffic (AGSI web server). A client device can connect and access the stream via a simple website. WebRTC on both the NX and a client device establish connection and start an RTP stream that is similar to "GUI-Based Streaming Data".*

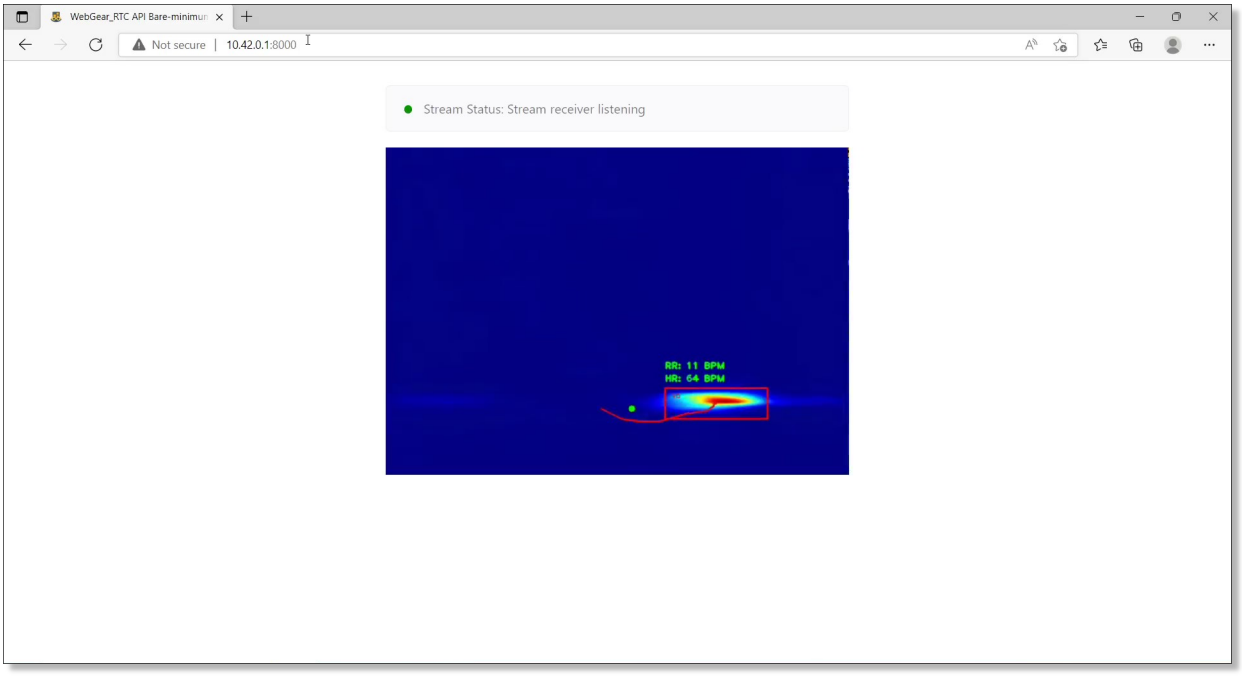


Figure 6.9: Screenshot of the website a client device would visit in order to see the Web-RTC-enabled video stream. *The website is a simple static website with very a local javascript file that does the WebRTC negotiation*

For quantization to FLOAT16, there is no need for calibration dataset since the precision difference of weights between FLOAT16 and FLOAT32 does not impact overall accuracy. Once quantization is done, the network definition, network weights, and other TensorRT related optimizations are contained in an "engine" (also called an "engine plan") that can serialized to the disk. After being serialized, they can be loaded in and ran similar to PyTorch and TensorFlow models.

6.3.2 Quantization Analysis

For analysis, models from the degradation analysis in Chapter 3 were chosen to be quantized from FP32 to FP16 and INT8. Each network's quantized model will then be evaluated against COCO val2017 to quantify possible accuracy drops. The COCO calibration dataset is 1,000 randomly chosen validation images and were provided by the TensorRTX repository [57].

6.3.3 Quantization Conclusions

Applying quantization on YOLOv3 Ultra, YOLOX, and DETR has yielded similar results. For FLOAT16 quantization, all networks suffered no mAP degradation and experienced significant run time speed improvements. On the Xavier NX, FLOAT16 models increase the average inference speeds by 2.8x. For Xavier AGX, we see an average increase of inference speed by 3.27x. For INT8 quantization, all networks suffered minimal mAP loss, but shows further speed improvements compared to the FLOAT32 models, with an average speedup of 3.8x on the AGX and 4.88x on the NX. Between FLOAT16 and INT8 speeds, INT8 is on average 1.35x faster on the AGX and 1.48x faster on the NX. INT8 PTQ comes with a certain level of mAP loss, and the loss extent can differ based on network architecture or calibration dataset. From Figure 6.10, YOLOv3 lost the most mAP with a reduction of 7%, while YOLOX and DETR lost 2% and 3% respectively. This loss in performance can be visualized in Figure 6.13, where detections between FLOAT16 and INT8 are shown for different networks.

In terms of applying quantization to a given real-time system, it is almost required to reduce your model precision to at least FLOAT16. Based on Figure 6.10 and the Xavier AGX and Xavier NX run times in Figure 6.12 and 6.11, there is no effect on accuracy and greatly increases model inference speeds. The harder decision is whether or not quantizing to INT8 is worth it. A mAP performance drop is hard to classify as "negligible" since it is subjective. For the case of COCO 2017, since the dataset consists of many classes and high object density in each image, the maximum of drop 7% by YOLOv3 is an acceptable loss of performance especially when considering a minimum speed increase of 1.5x.

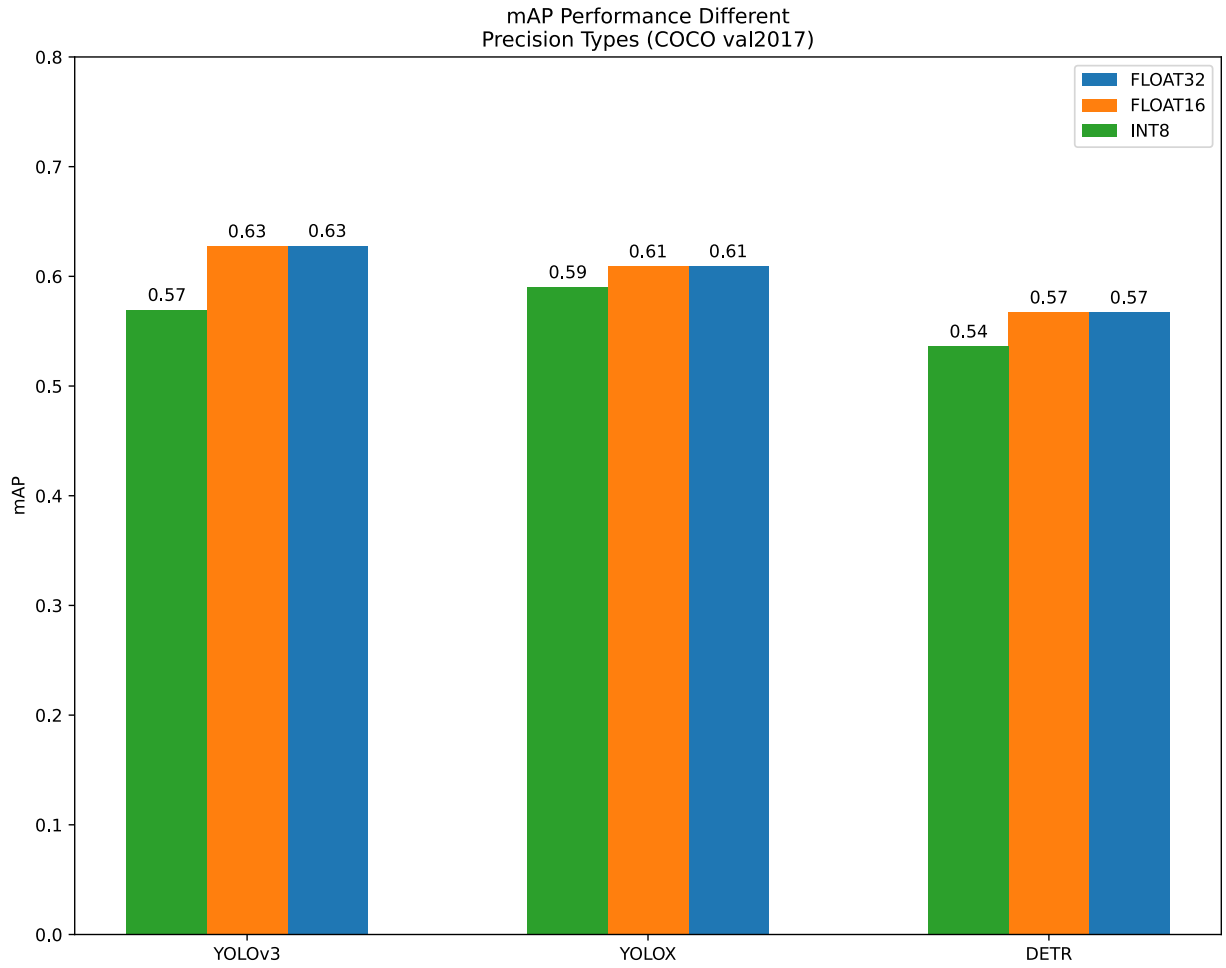


Figure 6.10: mAP @ IoU=0.5 performance of YOLOv3, YOLOX, and DETR on COCO val2017. *mAP performance between FP32 and FP16 models are identical because the range of values between the two data types are approximately equal. When it comes to INT8 quantization, this is not the case and network performance can drastically drop if the weights are not rescaled to INT8 properly. With the previously mentioned TensorRT quantization process, all networks lost less than 8% of their original mAP performance. All networks used the same quantization algorithm and the same calibration dataset. YOLOX fared the best in terms of mAP delta between INT8 and FP32.*

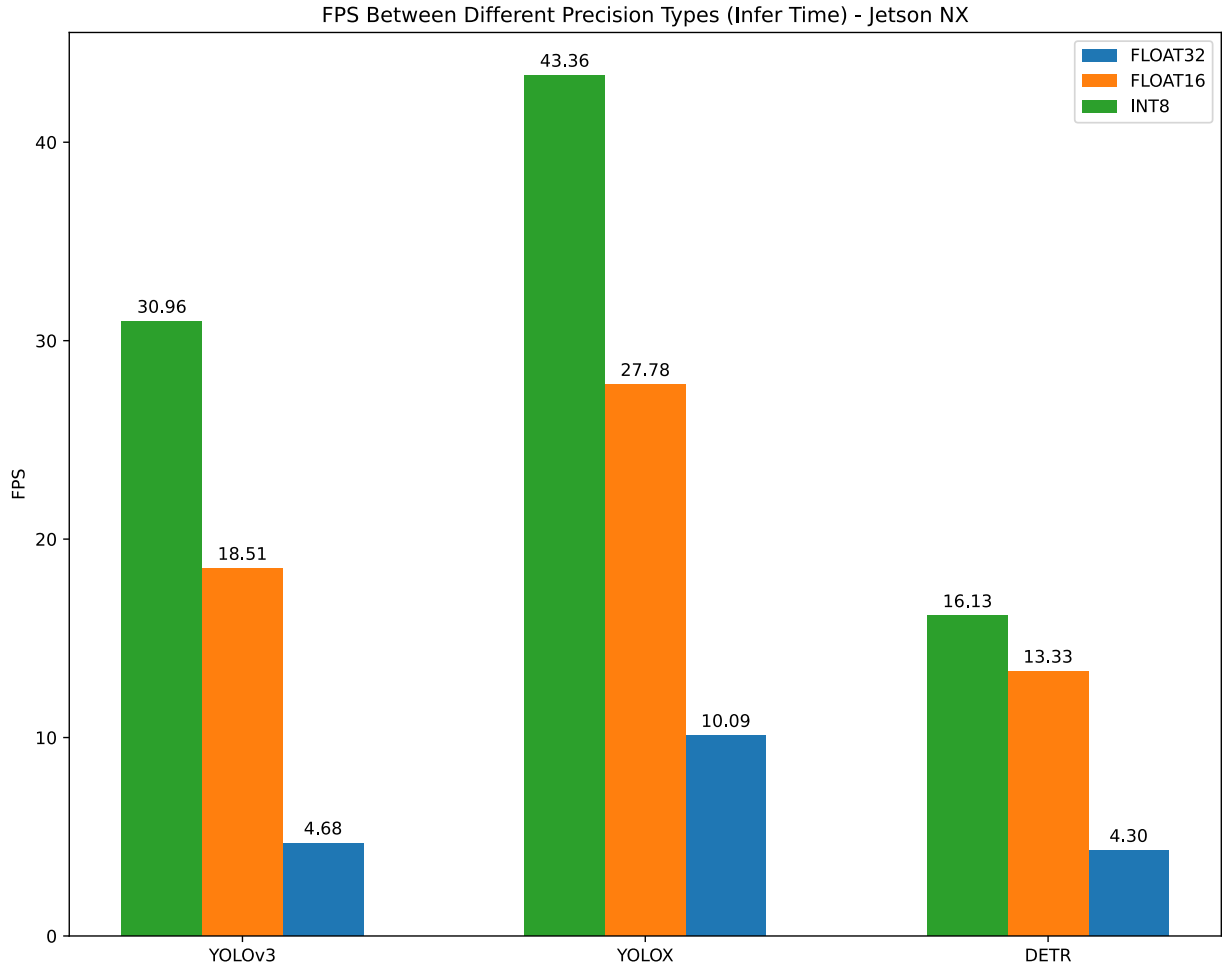


Figure 6.11: Inference speed for YOLOv3, YOLOX, and DETR at different precision types on the Jetson Xavier NX. *YOLOv3 inference speed increased the most across FLOAT16 and INT8 precision types. There was a 3.95x increase in speeds when comparing the FLOAT16 model to the FLOAT32 model. There is a 6.61x increase between INT8 and FLOAT32 and a 1.67x increase between INT8 and FLOAT16 respectively. Meanwhile, YOLOX runs the fastest overall with an INT8 model reaching an FPS of 43.36. This can be attributed to the lower parameter count when compared to YOLOv3 and DETR. DETR, even at the lowest precision type, does not run as fast as the CNN-based detectors. Since DETR's feature extractor is ResNet50, it is probable that the bottleneck is the traditional transformer network.*

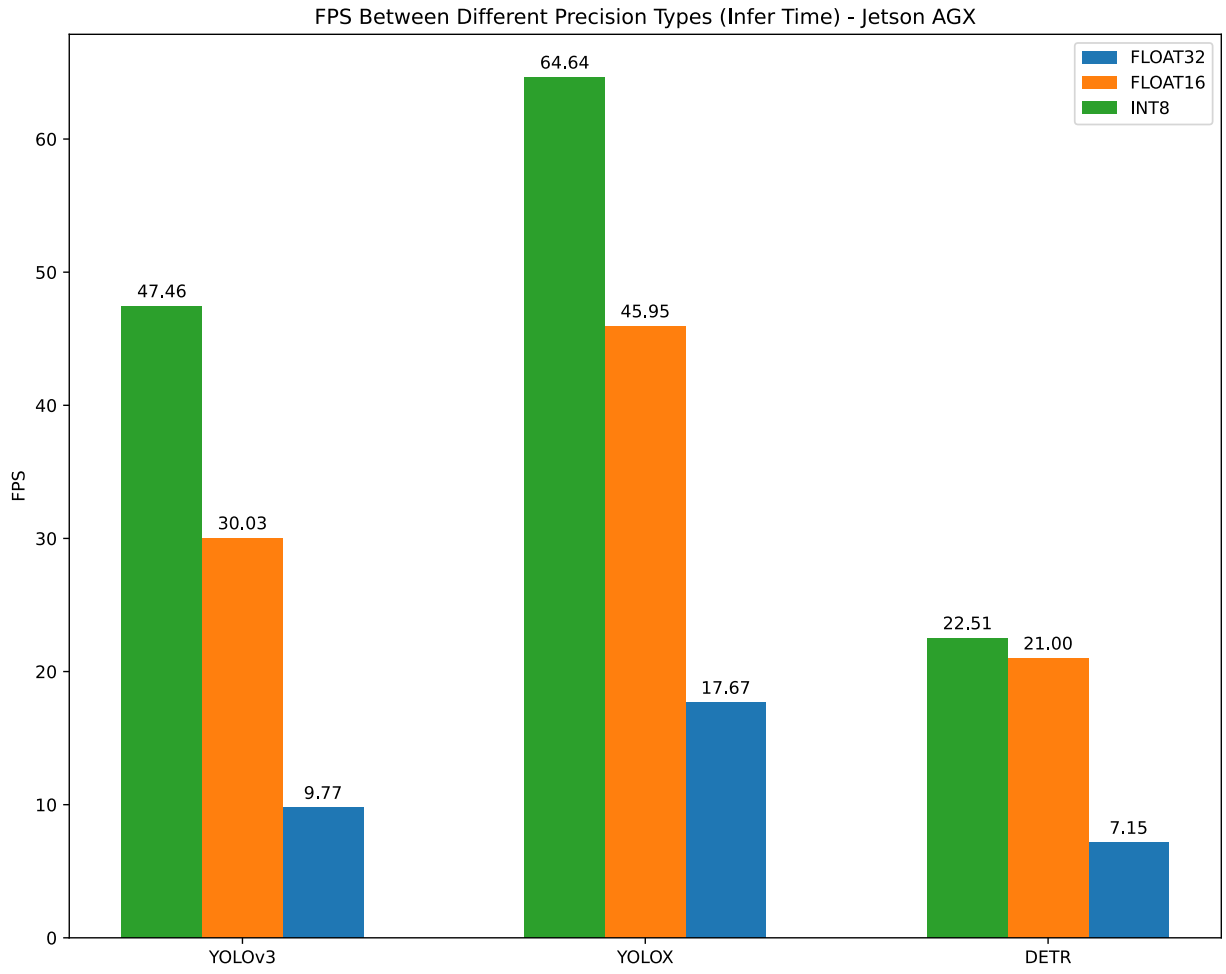


Figure 6.12: Inference speed for YOLOv3, YOLOX, and DETR at different precision types on the Jetson Xavier AGX. Run time improvements on the Xavier AGX follow a similar trend as the Xavier NX.

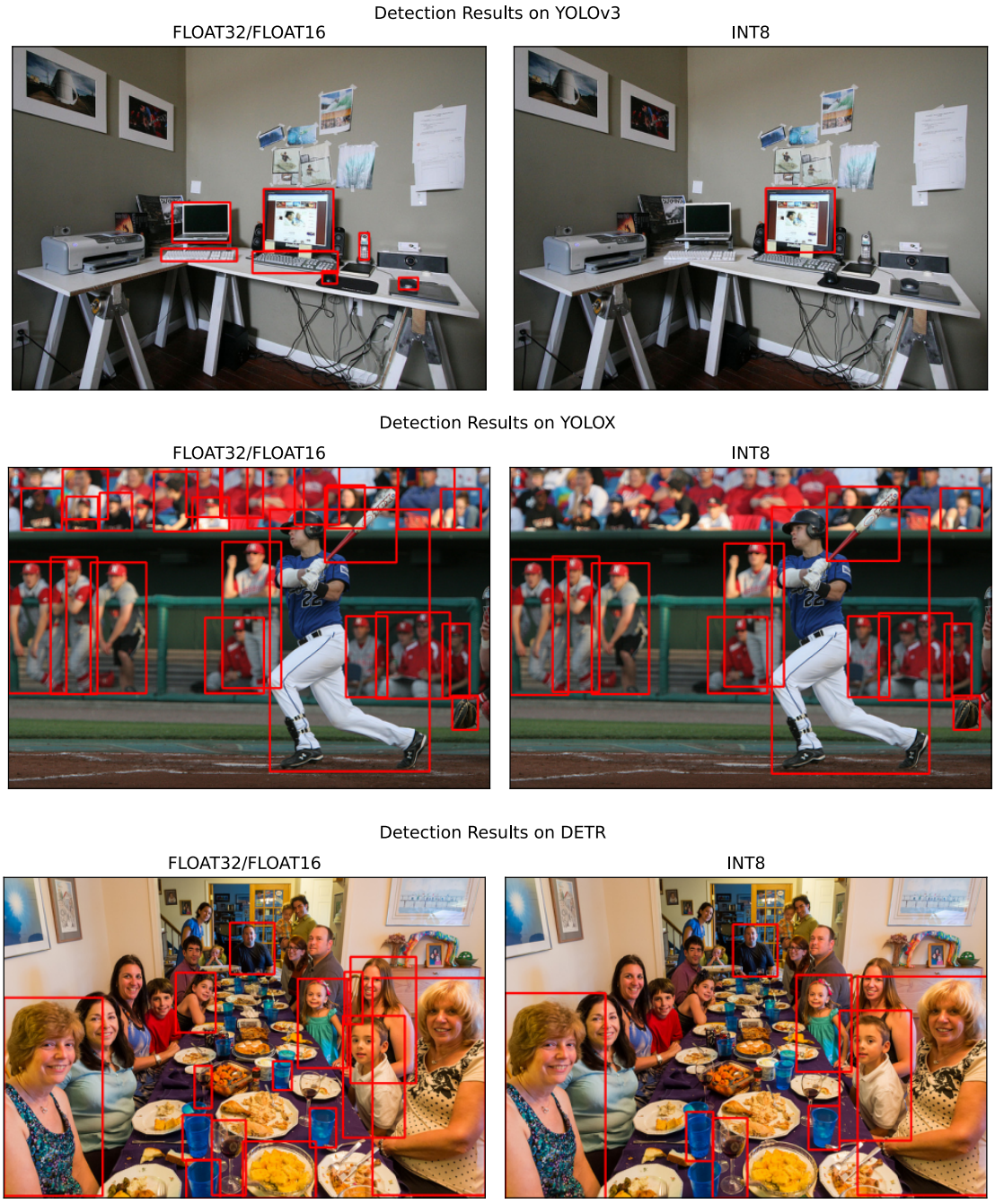


Figure 6.13: Detection differences between FLOAT32/FLOAT16 and INT8 on YOLOv3, YOLOX, DETR. There is notable decrease in detections on the INT8 results, where subjects and items in the background are lost or fall under the confidence threshold (0.25).

6.4 CNN Pruning Performance (SparseML)

6.4.1 General Pruning Process

As mentioned in Chapter 2, pruning is a very useful technique for further reduction of network size. This reduction can manifest in a reduction of weight file size, which in practical applications can help increase update times. Additionally, the reduction can manifest as speed improvements, but this depends on pruning type. For the pruning experiments done in this section, we utilize a pruning toolkit called "SparseML". The general process for using this toolkit is creating a "recipe", which consists of hyperparameters for training and sparisfying. This toolkit provides an API to integrate with Deep Learning frameworks, such as PyTorch and Tensorflow, and it is intended to work within the original network's training loop. To start pruning, an already trained network is retrained and at a certain number of epochs, the toolkit prunes away a certain percentage of neurons. The decision to prune a neuron or group of neurons is dependent on the pruning algorithm employed. For this analysis, Gradual Magnitude Pruning (GMP) is utilized for its simplicity and pruning performance.

Pruned YOLOv2 networks from Xilinx were evaluated to provide a baseline for pruned YOLO-based networks (Figure 6.16). The mAP loss across all classes in VOC2007 was minimal, with a maximum loss of 7% in the dog class. The extent of pruning is represented as "sparsity", where the higher the sparsity value, the greater number of pruned parameters.

6.4.2 Pruning Analysis

In addition to the YOLOv2 pruning evaluation, ResNet50 was pruned with pre-built recipes from SparseML. One recipe was for unstructured pruning and targeted 70% sparsity. Two other recipes were structured pruning with a targeted sparsity of 40% and 60%. The ResNet50 model was originally trained on "Imagenette", which is a 10-class subset of Imagenet.

6.4.3 Pruning Conclusions

From the analysis done on object detectors and classifiers, pruning can prove useful in reducing the number of neurons in a given neural network. This has cascading effects on other aspects of a neural network. One such effect is reducing the weight file size, which can be up to 400 megabytes for small to medium sized networks. Sending over-the-air (OTA) updates to embedded devices can require the file sizes to be small as possible, so actions like pruning can greatly aid in this type of support work. But, between the two pruning types, what situations would each be best suited for? Going back to the OTA update scenarios, the performance and sparsity levels shown in both Figure 6.16 and Figure 6.14, unstructured pruning reduces size of a network's weight file significantly while preserving the majority of network performance.
For acceleration purposes, unstructured pruning for traditional GPU devices would not help since the actual structure of the network's layers do not change. Structured pruning does

offer a level of GPU-based acceleration, and Figure 6.1 does show marginal improvements in inference speed when compared to the original and unstructured ResNet50. The structurally pruned network also sees similar marginal improvements on a desktop CPU when compared to the original ResNet50. Although, it is important to keep in mind that there is relatively significant loss in accuracy when sparsity level is increased.

6.5 Real-time Implementation

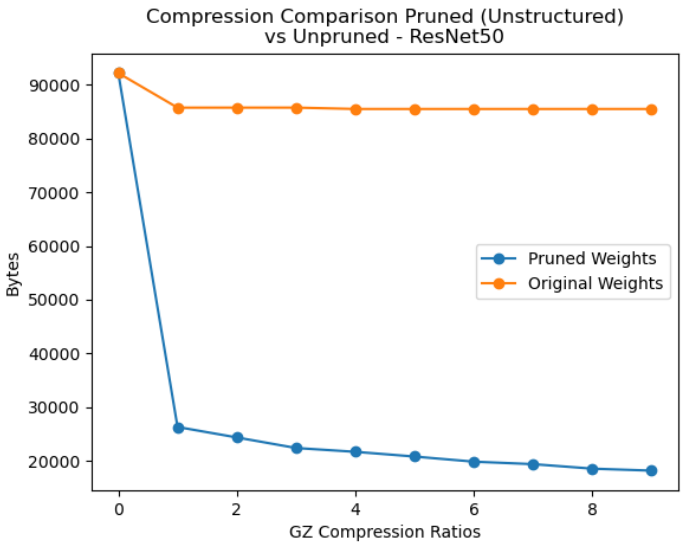
6.5.1 CNN Optimizations

Quantization

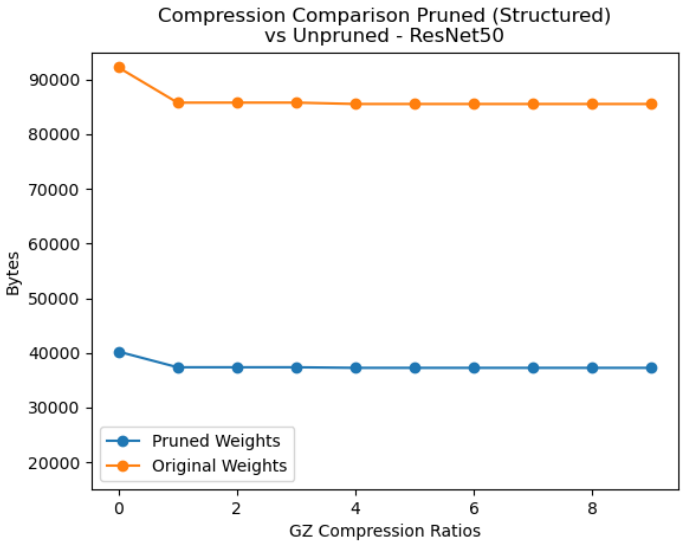
In regards to the system integration process, the same INT8 quantization detailed in the previous section is utilized on the RADAR-based detector. The calibration portion of this process uses the whole real-time MVDR-based dataset defined in Chapter 4. The mAP for the YOLOv5n model was 71.57%, which comes to a mAP degradation of 10.82%. This is noticeably larger than the networks seen in the Quantization Analysis section. Some reasons for this might include calibration size or an issue with the YOLOv5 architecture. Another possible reason is the robustness of the RADAR training, where the computed weights did not lend themselves to a properly spaced factor for INT8. Speed-wise, FLOAT16 inference speeds on the RADAR-trained YOLOv5n were on average 10ms (100 FPS). The INT8 model accelerated the inference speed to 3.36 ms (297.61 FPS), which is a 2.97x increase. Although, considering the mAP performance drop and the already high FPS on the FLOAT16 model, the decision was made to stick with the FLOAT16 model for the real-time system.

Pruning

Similar to quantization, we attempt to apply pruning onto YOLOv5n in both unstructured and structured fashion. For unstructured pruning, the sparsity is set a relatively high 90% sparsity for all convolutional and fully-connected layers. The resulting model evaluates to an average mAP score of 76.5% on the MVDR-based dataset, but under a GZ compression ratio of 9, reduces the weight file size from 371.8 MB to 77 MB. This is a 4.78x decrease in weight filesize. Structured pruning is also applied on a select number of layers. These layers are chosen based on their association with higher-level structures like residual blocks. These dependencies can be automatically generated via SparseML's ONNX dependency functions. The sparsity was set at levels below 50%, but all sparsity levels above 0.1% resulted in mAP performance of less than 1%.



(a) Weight filesizes after Unstructured GMP on ResNet50



(b) Weight filesizes after Structured GMP on ResNet50

Figure 6.14: Comparison of ResNet50 file sizes between structured and unstructured pruning. *For unstructured pruning, the maximum filesize reduction was over 70% (from 85,000 bytes to 25,000 bytes) at an initial compression ratio of 1. Proceeding compression ratios reduced filesizes to a much lesser extent but consumed more CPU utilization as the ratio increased. The initial filesizes without compression are the same between pruned and unpruned because all the pruned weights are "zeroed" out. Additionally, the compression applied to the unpruned models have very little effect after the first compression ratio. For structure pruning, entire convolutional filters were removed rather than zeroed out, which results in the model automatically being smaller without any compression. Applying compression yields very little filesize savings and like its unpruned counterpart, does not change after compression ratio of 1. As expected, structured pruning yielded less sparsity with a maximum filesize reduction of ∼56% at no compression.*

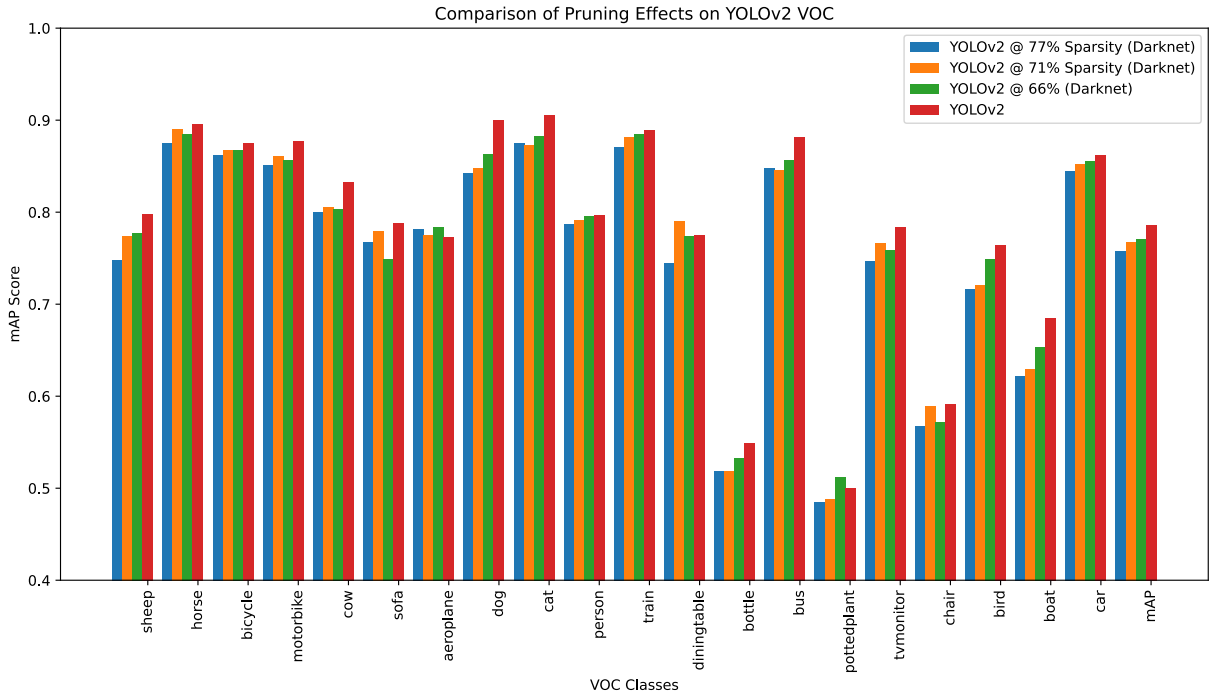
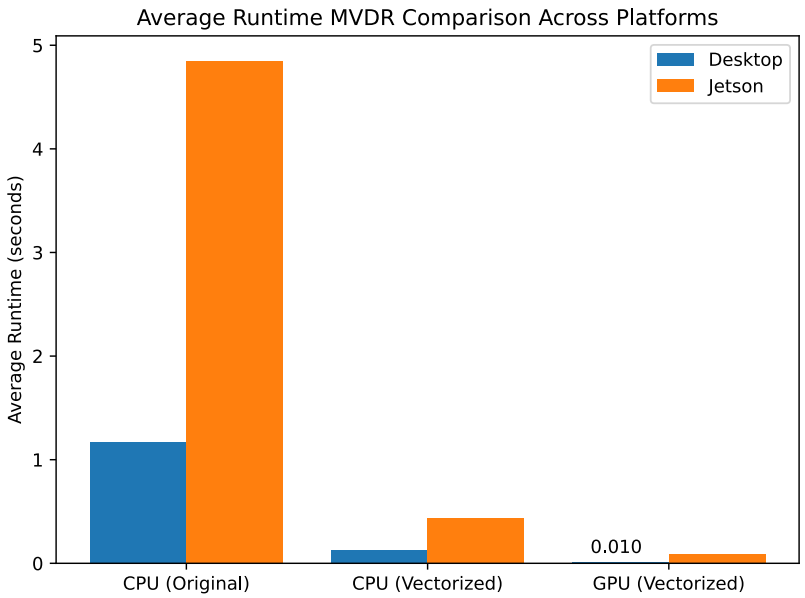


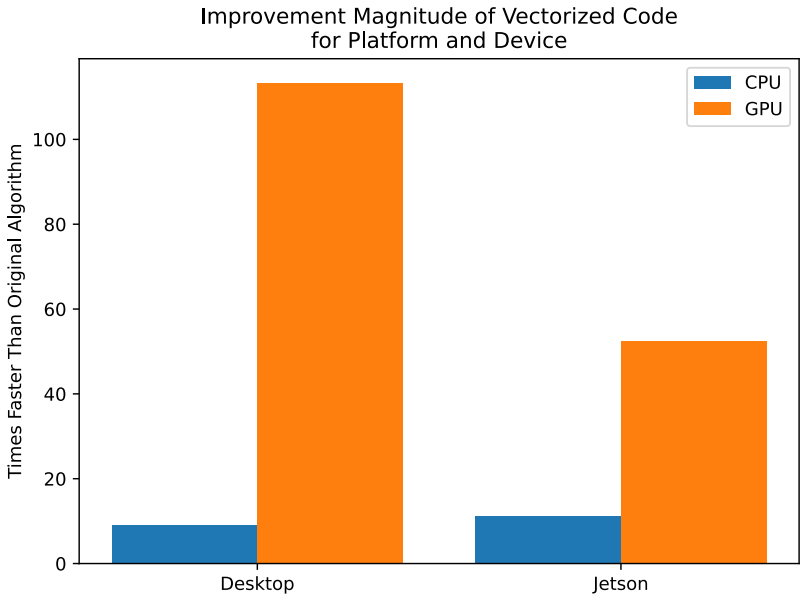
Figure 6.16: mAP comparison between different sparsity levels of YOLOv2. *This bar chart benchmarks the mAP performance of YOLOv2 at no sparsity, 77%, 71%, and 66% sparsity on VOC2007 test set. All these models are sourced from Xilinx's Model Zoo for the Ultrascale FPGA products (although these runs were on a discrete GPU). This shows us that heavily pruned object detectors can still retain the majority of their original mAP score. Class-wise, the mAP discrepancy ranges from 1% to 6%.* ***Credit for assistance: Priya Patel***

Table 6.1: Table of ResNet50 top-1 accuracy and FPS performance on Imagenette under different pruning types. *As shown in pruning literature, unstructured pruning can have high sparsity while preserving the majority of the accuracy performance.*

Model Type	Sparsity	Imagenette Top-1	GPU FPS	CPU FPS
ResNet50 (Original)	0%	0.99	122.3	5.6
ResNet50 (Unstructured Pruning)	90%	0.97	120.6	5.5
ResNet50 (Structured Pruning)	40%	0.92	123.3	7.4
ResNet50 (Structured Pruning)	60%	0.84	122.3	9.5



(a) Run time speeds between original, vectorized on CPU, and vectorized on GPU



(b) Magnitudes of improvement over the original implementation

Figure 6.15: Performance and magnitude improvements between non-vectorized algorithm, vectorized algorithm on CPU, and vectorized algorithm on GPU. *From sub-figure a, we can see that the original non-vectorized algorithm was not meeting the minimum FPS threshold (1 FPS) on the Xavier NX. Vectorizing and implementing on the GPU allowed a speed up of 50x (sub-figure b) on the Xavier NX. Reducing the number frames fed into MVDR can give even more headroom for other processes.*

6.5.2 RADAR Processing Optimizations

In regards to RADAR-specific processing, this thesis only focuses on the implementation aspect. One particular RADAR processing algorithm is Minimum Variance Distortionless Response (MVDR). This is a beamforming algorithm that transforms a segment of RADAR data into a matrix representing the environment, where the y-axis is the degrees from the RADAR device and x-axis is the distance away from the RADAR. MVDR is used for display purposes since it is one of the fastest beamforming algorithms.

A traditional implementation of MVDR is to loop through a matrix that holds a stack of raw RADAR frames. Implementing this in Python results in utilizing Pythonic for-loops on NumPy matrix elements. This is known to be a non-optimal method since there are no strict datatype declaration in Python and the interpreter needs time to figure out what the datatype of the iterator is. Vectorization of both for-loops can result in acceleration of the algorithm. The vectorization process is essentially boiling down the different repetitive element-by-element or column-by-column operations into a NumPy-based or equivalent matrix-based operation that can be accelerated with a C++ implementation or other matrix acceleration libraries (ex: BLAS).

With the vectorized code, we see a 10x improvement in CPU run time for both a mid-range desktop CPU and the Carmel ARM CPU in the Xavier NX. Since we decomposed MVDR into a set of matrix operations, we can take advantage of GPU acceleration. The vectorized MVDR code is re-implemented in CUDA with CuPy, and we see a 120x increase in run time speeds on the NVIDIA 2070 Super. Additionally, we see a 50x increase in run time speeds on the Xavier NX's Tegra-based GPU. All run times on all the aforementioned hardware platforms are detailed in Figure 6.15. By essentially avoiding many of the slow aspects of Python and utilizing the GPU in cases of matrix calculations, we can maximize device throughput and decrease latency.

6.6 RADAR Performance Benchmarks

In Figures 6.18, 6.19, 6.20, we analyze the overall pipeline FPS, power usage, and thermal behavior of the Xavier NX running the full real-time system for a preset number of fully-processed frames. The system is enclosed in the final iterative prototype seen in Figure 6.3 and run for 2,000 timesteps. Additionally, we conduct a small experiment on stationary subjects to determine the error rate of the vital-sign estimation methods in the system.

6.6.1 Heartrate and Respiratory-Rate Performance

The real-time system utilizes established works for methods from Abdel-Kareem Moadi and Chandler Bauder to produce heartrate and respiratory-rate estimations. The estimation methods are not central to the thesis's focus, but the following will be a simple benchmark of the estimated heartrates against a pulse sensor and respiration sensor over time. The pulse sensors are Contec CMS50D+ pulse oximeters while the respiration sensor is the NeuLog

Breathing Belt.

We record multiple clips of subjects at 1 meter, 2 meter, and 3 meter ranges. In addition to different ranges, we also record data at different angles. All subjects are sitting stationary with the device at the same level as their chest. Subjects were not limited in terms of breathing rate or pattern during recording. After comparing the ground truth against the estimated values, the average error for single subjects across all ranges and angles was 0.346 and 2.52 for respiration rate and heartrate respectively. For multi-subject scenarios, the average error was 0.68 and 2.7 for respiration rate and heartrate respectively. For average error per range, please refer to Figure 4 in the Appendix.

6.6.2 FPS Performance

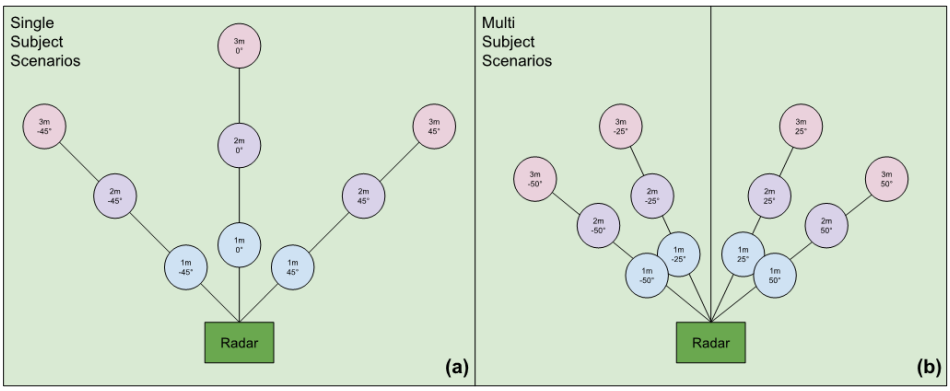
The FPS performance of the system was measured simultaneously with the power analysis, and we see a maximum average FPS of 14 on the highest power profile. All power profiles exhibit very slow FPS for the first ten to twelve frames due to CUDA kernel initialization for both the neural network and MVDR. The variances between each individual frame time can range from 5 to 25 FPS based on power profile and MVDR "wait-time". This wait-time is the time it takes for the Xavier NX to collect the set number of raw RADAR frames from the VMD3. Compounding this wait-time are the main processes, such as running MVDR and the object detector. This analysis was done with one person walking around, but additional subjects in the scene would add additional processing time due to the iterative steps we take with AMTC.

In terms of power profiles, higher wattage power profiles are able to run the pipeline at higher speeds. In support of the observations seen in Figure 6.19, the pipeline under 10W 4 Core is marginally faster than 15W 2 Core. This makes sense since the pipeline relies on multiple processes that could be scheduled on different cores to increase parallel performance. A potential power plan that allows for a 10W 6 Core could work as an optimal power profile for a battery-powered version of this setup.

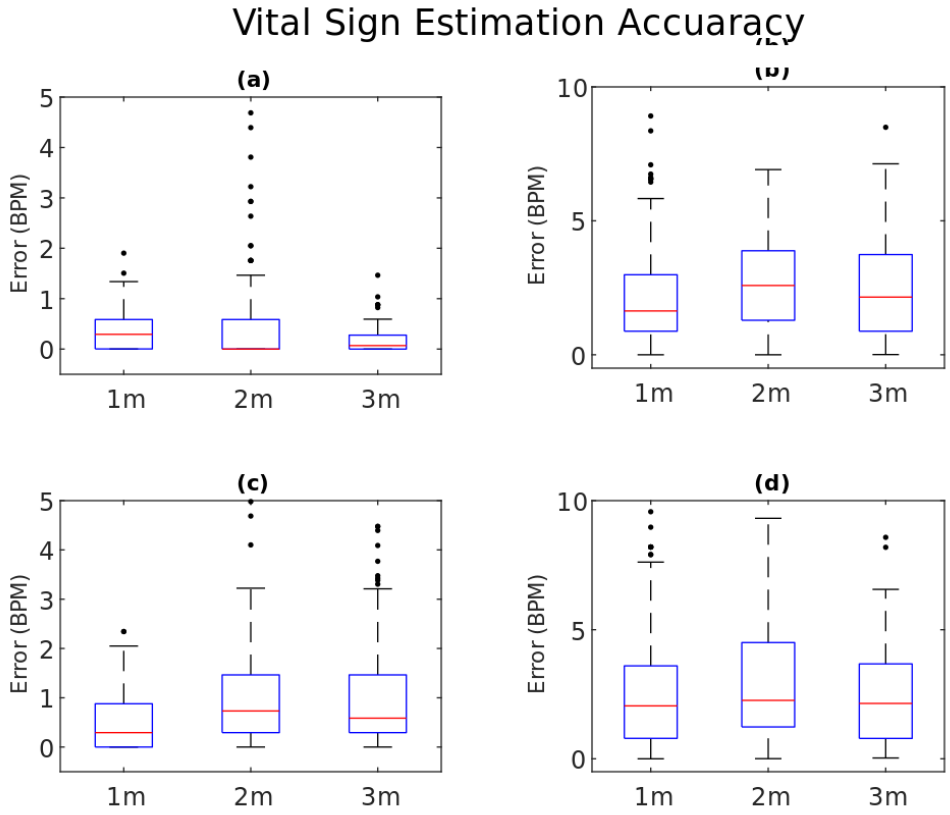
6.6.3 Power Analysis

Because this device is specified as "portable" and "handheld", power consumption is a prudent factor to consider. If the device is powered by a typical lithium-ion battery, considerations regarding the size and charge capacity of a battery rely on the power consumption of the three hardware components described previously. There is a delicate balancing act between reducing power consumption (increasing the up-time of the device) and the speed of the whole system.

For the Xavier NX, there are preset "power profiles" which are typically denoted by their targeted maximum Wattage and the number of CPU cores enabled. There are other attributes that power profiles include but are not apparent from the name, such as CPU and GPU frequencies. To create a power profile, one must create an appropriate configuration file that details each CPU and GPU core's clock frequency [42]. Unsurprisingly, the higher



(a) Diagram showing experiment setup for single and multi-subject capture



(b) Box and whisker plot of error rates at different ranges

Figure 6.17: Respiration-rate (a,c) and Heart-rate (b,d) error rates. *The average heartrate and respiration rate error across all ranges for single subject is 2.52 and 0.346 BPM, respectively. For multi-subject, average heartrate and respiration rate error across all ranges is 2.7 and 0.68 BPM, respectively.* ***Credit for assistance: Abdel-Kareem Moadi***

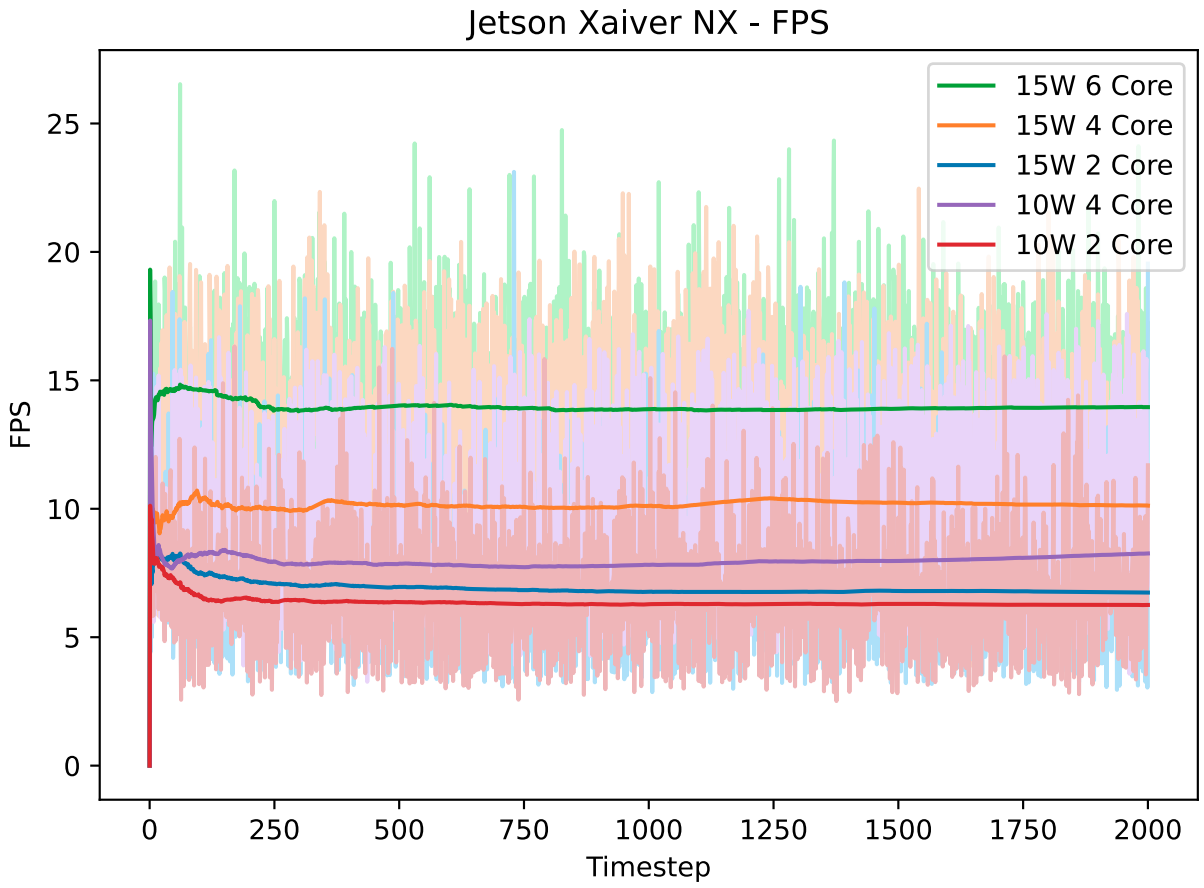


Figure 6.18: Chart showing the average FPS of the real-time pipeline over 2000 timesteps. *The speed is the accumulated time from the "wait" for RADAR data accumulation, processing through a given DoA method (ex: MVDR), localization with the NN method, tracking with BYTE, and other miscellaneous operations that occur on the main thread. It can be seen that the higher wattage power profiles are able to run the pipeline at higher speeds. It can also be observed that the pipeline under 10W 4 Core is marginally faster than 15W 2 Core. This signals that the number of cores matters more than the frequency each core is running on. Figure 6.5 supports this conclusion since the pipeline relies heavily on multi-processing.*

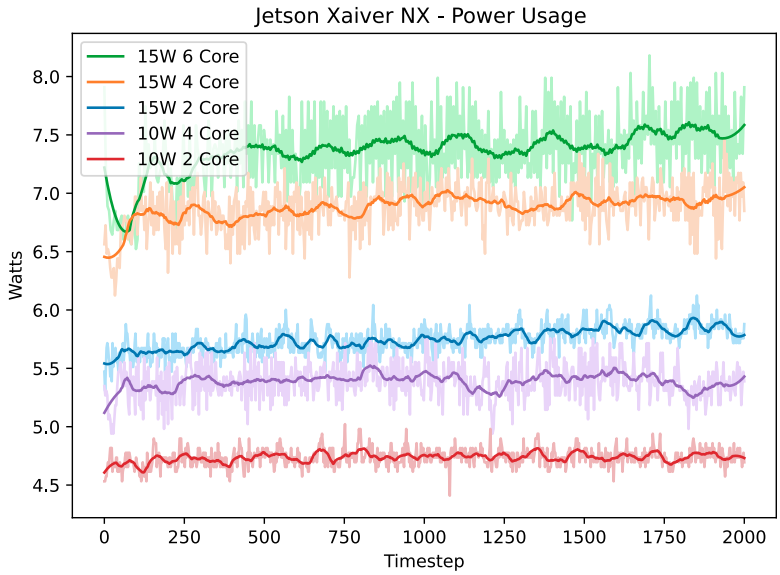


Figure 6.19: Chart showing power usage of the Xavier NX over 300 timesteps. *It is quite obvious that the higher wattage power profiles with associated higher CPU core counts consume more wattage. What is interesting though is the relative similar measurements between 15W 2 Core and 10W 4 Core.*

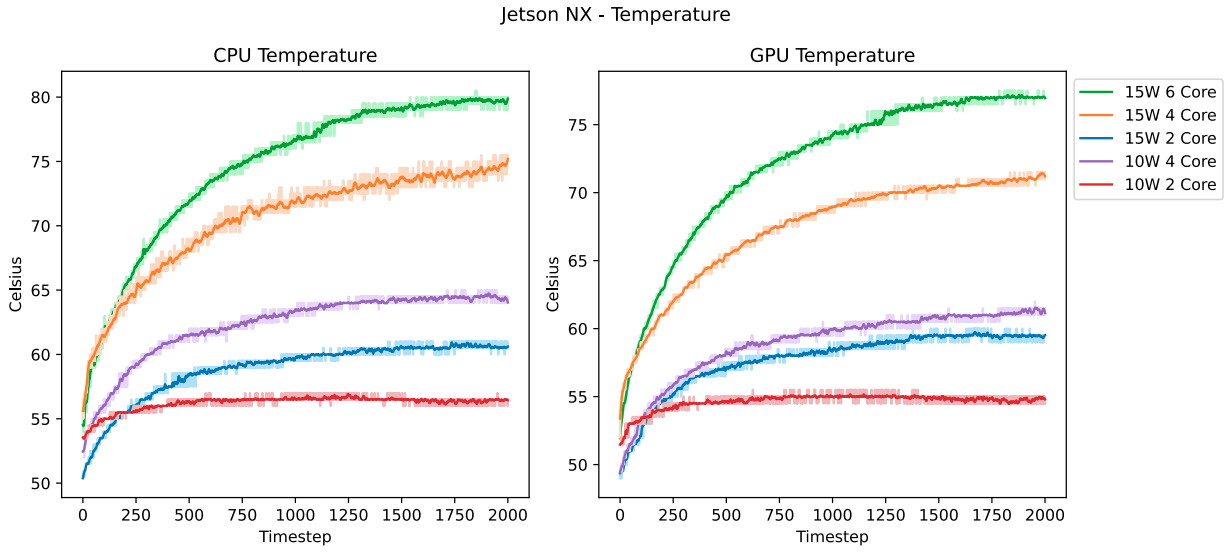


Figure 6.20: Chart showing the average temperature observed on both the CPU and GPU temperature sensor over 2000 timesteps. *Each run is buffered with a 10 to 20 minute wait to reset to an idle temperature of around 45 to 50C. Regardless of power profile, the temperature increase is almost identical in both value and in their linear trend. Also, as maximum wattage increases, the slope of the temperature increase increases.*

wattage power profiles allow for higher real-world power utilization. Although, there is an interesting observation with the actual power usage for power profiles "15W 2 Core" and "10W 4 Core". Both profiles utilizes roughly the same amount of wattage, but because the "10 W 4 Core" profile offers more CPU cores.

6.6.4 Thermal Analysis

The temperature was taken from the CPU and GPU thermal sensors and are plotted in Figure 6.20. Temperature readings were taken with a small fan that attached to the main thermal pad. Each run on a power profile was done with all the hardware components within the 3D printed enclosure. This setup provides a sustainable level of cooling for both CPU and GPU for the number of timesteps taken. The GPU experiences slightly faster increase in temperature since the majority of heavy computation is placed on the GPU. Power profiles again play a large role in the speed and extent of the GPU and CPU temperature increases.

Chapter 7

Conclusion

7.1 Summary

The work performed in this thesis revolved around implementing and benchmarking methods for real-time detection and tracking on a select number of hardware platforms. These devices are embedded systems that offer SWaP characteristics for efficient edge computing. Following the pipeline in Figure 1.2, the implementation and benchmarking aspects are composed of multiple different types of analyses. The first component is a mAP benchmark done on various YOLO-based CNN object detectors. The YOLO-based networks were the main focus due to their architectures and methods composed around efficient and real-time computation. All networks were benchmarked on COCO val2017 for their mAP scores at IoU@0.5 as well as IoU@0.5:0.95:0.05. Additionally, we benchmark the inference speeds of each network on three different GPU-based devices - a discrete GPU, Jetson Xavier AGX, and Jetson Xavier NX.

Once a CNN detection method is selected for implementation on a Jetson device, the same software structure and methods used for network development and training do not maximize throughput on these embedded systems. Rather, we must employ optimizations, such as quantization and pruning. This thesis does a brief analysis on both these optimizations and how they affect network metric performance. For example, INT8 quantization across different object detection architectures cause an average mAP decrease of 4%. Additionally, pruning can be branched into structured and unstructured pruning. Each type of pruning have their own advantages and disadvantages. For example, unstructured pruning can output networks with higher sparsity while structured pruning can increase inference times on GPUs with no sparse matrix operation support. Pruned YOLOv2 models from Xilinx Model Zoo are benchmarked for their mAP and class AP scores. These models are at 66%, 71%, and 77% sparsity levels, and their performance loss is linear but minor, with the maximum mAP loss being 6%.

Further work with the YOLO-based networks revolve around detection of signatures within 2D RADAR outputs. This thesis trains all YOLO networks from the previous COCO benchmarks on synthetic and real RADAR data and select YOLOv5n as the overall optimal

choice for detection performance and run time performance. In conjunction with detection methods, the thesis also evaluated multi-object tracking methods on the MOT dataset as well as the previously utilized RADAR datasets. We see methods utilizing state-based estimations like Kalman filters have the best balance HOTA performance and speed across different types of CPU devices, such as a traditional desktop CPU and ARM-based CPUs.

As part of the "Analysis" phase of the thesis pipeline, an exploration of sensitivity analysis was done on object detection networks. This sensitivity analysis follows work done by previous IRIS lab students as well as other researchers, but focuses on compression methods and their effect on three object detectors with distinctive neural architectures. These networks were YOLOv3 Ultra, YOLOX, and DETR. JPEG, WebP, Bilinear Resizing, and a neural network-based image compression method were utilized for evaluation on COCO val2017. Additionally, H264 and H265 were video compression methods that were also applied on these networks for evaluation on COCO val2017. We see a negative log trend for all compression methods except the neural network-based compression method. Additionally, space saving analysis was done on COCO val2017 for image compression methods and the training videos in MOT17 for the video compression methods. From this, the following conclusions were made for utilizing compression methods in real-life IoT device scenarios:

- WebP at a quantization level of 50 will save you 95.7% of the original image matrix while mAP performance only degrades by a maximum of 5%.
- If WebP encoder is too slow on a given system, using JPEG at a quality level of 50 yields similar mAP performance and size reduction but worse image quality
- There is very little difference in mAP performance between H264 and H265, so the improved image quality of H265 makes it an optimal choice at a CRF of 30
- If a given device does not have an accelerator for H265 encoding, utilizing H264 yields similar mAP performance

Additionally, this thesis also produces a prototype device that can detect people and estimate their vital signs. This prototype is composed of hardware platforms, algorithms, and optimizations discussed in the "analysis phase" of the thesis pipeline. This prototype development is what is considered the "application phase" in the same pipeline, and it consists primarily of the prototype enclosure design, the software pipeline, the hardware pipeline, software optimizations, and benchmarking overall-system performance.

The prototype's physical design is a simple rectangular case that was designed in CAD and 3D printed with standard PLA plastic. The design was split into two parts, one for the sensors and compute device and one for the display. These two parts are then simply snapped together to finish assembly. For the hardware pipeline, the thesis utilizes different communication methods between different sensors as well as displays with the compute device. The connection between the camera sensor and Jetson devices is the CSI port, and the connection with the RADAR device is an Ethernet connection.

For the software pipeline, we employ a multiprocessing approach to asynchronously communicate with sensors, process data, and display as well as transmit the output data as fast as possible. For communication between these different processes, the program utilizes shared array and value pools as well as conventional multiprocessing Queues and Pipes. Optimizations for this real-time system include previously mentioned quantization and pruning of neural networks, but the thesis also defines smaller optimization methods for other components. One such component is the RADAR processing method, MVDR, which was converted from traditional for-loops to matrix operations. This also meant that this method could be run on the GPU and resulted in a 50x speed up on the Jetson Xavier NX when compared to running on the CPU.

Performance benchmarks regarding speed, temperature, and power usage over different power profiles on the Xavier NX showed multiple things. Firstly, the number of CPU and GPU cores activated is more important than the maximum frequency of each core. This is showcased by the 10W 4 Core power profile running marginally faster than 15W 2 Core while consuming less overall wattage and producing less heat. Thermal benchmarks show that an Xavier NX module on a 3rd-party small form factor board such as the NCB00, when equipped with a small fan, can sustain comfortable temperatures around 60 to 70 C. Additionally, the heartrate and respiration rate accuracy of the system was taken, with major assistance from Abdel-Kareem Moadi. Single and multiple subjects were recorded sitting stationary at three different ranges as well as two different angles for each range. For single subject, average heartrate error across all ranges was 2.52 BPM while the average respiration error across all ranges was 0.346 BPM. For multi-subject, the average heartrate error was 2.7 BPM and average respiration error was 0.68 BPM. With the findings and deliverables from this thesis, we hope that the adoption of GPU-based embedded devices continue for vital-sign estimation and other applications.

7.2 Contributions

As laid out in Chapter 1, this thesis has 6 main contributions:

- **Contribution 1 (major):** The application of neural networks for real time detection and localization on various embedded systems. Within this contribution, there are multiple analyses done. A baseline performance of YOLO-based networks on COCO val2017 is conducted. This also includes inference speeds across different GPU platforms. Newer networks such as YOLOv4, YOLOv4-CSP and YOLOv5L achieves very comparable performances. YOLOv4 achieves the highest mAP@0.5 and mAP@[0.5:0.95:0.05] with 68.21% and 49.41% respectively. Although, YOLOv4 CSP and YOLOv5L contain less parameters and run faster on all GPU-based systems.

  In terms of inference speed, there is a trend between the speed-up magnitude of the discrete GPU and the Jetson Xavier devices. For sub-10 million parameter networks, the discrete GPU is 6.79x and 7.86x faster when compared to the Xavier AGX and NX respectively. For networks over 10 million parameters, we only see a 3.94x and 6.94x speed up on the discrete GPU when compared to the Xavier AGX and NX respectively.

Sub-10 million networks were on average 1.20x times faster on the AGX, and for bigger networks, were on average 1.76x times faster on the AGX versus the NX. All sub-10 million parameter networks had real-time inference speeds on both Jetsons, making them realistic targets for inclusion into the real-time system.

An additional analysis is taking previous COCO networks and retraining them on real and synthetic RADAR datasets. For synthetic RADAR training, all networks were set with roughly the same hyperparameters and tested against an MVDR and Range-Velocity datasets. From this training and evaluation, YOLOv5L performs the best with an average mAP score of 78.1% and 87.11% on the MVDR and Range-Velocity datasets respectively. When comparing performance amongst sub-10 million parameter networks, the leading network depends on the dataset. For MVDR-based data, YOLOv4-Tiny performed the best with an mAP of 67.73%. For Range-Velocity data, YOLOv3 Tiny yielded an mAP of 84.15%. With real RADAR data, only the sub 10-million parameter networks were trained again. Training occurred with all the same hyperparameters as the synthetic training excluding the epoch value. Comparatively to the synthetic RADAR, the real-time dataset increased network mAPs by an average of 19.25%, but YOLOv4-CSP continues to outperform all other sub-10 million networks. When comparing across different input sizes, YOLOv4-CSP continues to lead in input sizes of 416 and 208. At the smallest input size, YOLOv5n performs the best with an mAP of 82.39%.

- **Contribution 2 (major):** Analyzing the effects of data augmentations on a set of neural networks, with a focus on compression methods. More specifically, the analysis measures the negative mAP impact of different image and video compression methods on SoTA detectors trained on COCO 2017. For the reasoning behind this analysis, the contribution is an extension of existing sensitivity analysis work done by IRIS students and other researchers. Additionally, possible future iterations of the vital-sign based system could pivot towards a system architecture that requires streaming data. This streaming will have to be compressed and can affect accuracy.

  The compression analysis saw that all image and video compression methods (excluding the Gaussian Entropy Network) caused a negative logarithmic trend in mAP. This can be described as the mAP performance holding steady for most of the lowest compression parameters until at a certain mid to high-range compression parameter, the mAP drops precipitously. The analysis also ran average file size reduction for each image compression method on the COCO val2017 dataset as well as the MOT17 training videos for the video compression methods. The Gaussian Entropy Network reduced the images the most, but the quantization of features map was not real-time. Both video compression methods showed very similar file size reduction, but H265 produced higher image quality.

- **Contribution 3 (minor):** Creation of synthetic RADAR data and its various iterations for various network localization. The contribution contains analysis on the datasets themselves in terms of size and position characteristics. Additionally, we discuss the need for reflection-based perturbations that mimic the appearance of the existing MVDR or Range-Velocity RADAR data. The synthetic RADAR data goes

through two iterations. The first one was based on various autonomous car tasks, where the sensor is in the bottom-center and objects near the top would be the farthest away.

This first iteration was trained on YOLOv3 and scored a mAP of 71.73% and 31.02% on Range-Velocity and MVDR testsets respectively. This disparity in mAP performance created a second iteration focused on emulating a mix of appearances shared between the two datasets. This second iteration contained parameters to create clips of RADAR signatures moving on a canvas. The new frames would also contain various perturbations within the scene, and the signature would have inconsistent reflections. Re-evaluating YOLOv3 with the same training parameters resulted in a mAP of 84.15% and 72.22% on Range-Velocity and MVDR datasets respectively.

- **Contribution 4 (minor):** Optimization of various neural networks for run time acceleration on embedded devices. These optimizations are quantization of network weights into FLOAT16 and INT8 as well as unstructured and structured pruning for image classification and object detection. For the quantization analysis, the same networks analyzed in Contribution 2 (YOLOv3 Ultralytics, YOLOX, DETR) are utilized here. Quantizing models to FP16 increased average inference speed by 3.27x on the Xavier AGX and 2.80x on the Xavier NX. Because the precision of the weights is still within range for both FLOAT32 and FLOAT16, there is no drop in mAP performance. For INT8 quantization, mAP performance drops on average by 4%, with YOLOv3 suffering the most at 7% mAP decrease. But, INT8 quantization results in an average speed up of 3.8x and 4.88x on the Xavier AGX and NX respectively when compared to FLOAT32. Between INT8 and FLOAT16, the average speed up was 1.35x and 1.48x on the AGX and NX respectively.

  The pruning analysis was done on an image classifier called ResNet50 and an object detector called YOLOv2. For object detection, mAP evaluation was done on YOLOv2 as well as versions of it pruned to different sparsity levels of 66%, 71%, and 77%. These models were provided by Xilinx and were trained on VOC2007 and VOC2012. Across all sparsity levels, the mAP loss was minimal with the maximum mAP being 6%. Pruning on ResNet50 was done with the help of SparseML, a toolkit for pruning and quantizing models. ResNet50 was pruned in an unstructured and structured method with sparsity levels of 70% and 40% respectively. Unstructured pruning showed lower loss of accuracy (-3.33% from baseline) than structured pruning (-6.6%) at a higher level of sparsity.

  Additionally, we explore effects of weight file compression on both types of pruning methods. Due to higher sparsity target, the unstructured ResNet50 compresses to 25,000 bytes (70% savings) compared to structured ResNet50 compressing to 40,000 bytes (56% savings). Increasing GZ compression ratios can bring unstructured ResNet50 down further to 20,000 bytes. Although, it should be noted that unstructured pruning has little to no effect on run time when it comes to most GPUs, due to inefficient sparse matrix operations.

- **Contribution 5 (minor):** Data collection tasks performed for RADAR detection and cross-modal association. RADAR datasets provided by Yao Yu from MaXentric

and Marvin Joshi from the University of Tennessee allowed us to conduct analyses, such as the Synthetic Data benchmarking and MOT tracking. This contribution encompasses the involvement of the author in the annotation process of both datasets. Additionally, when the real-time system was at a mature phase, real-time collection and new annotation methods allowed for the collection more data. This contribution also encapsulates the real-time collection process as well as the auto-labeling that was used to speed up the RADAR and optical annotations. Additionally, new annotations such as association across modalities future-proofs this dataset for additional research.

- **Contribution 6 (minor):** Benchmarking of multi-object tracking in the optical domain and applied to RADAR tracking in real-time system. Furthermore, the cross-modal association task is also tackled with an introductory method. The MOT benchmarking included taking different tracking methods based on their unique attribute. These methods are IoU Tracker, SORT, DeepSORT, and BYTE. Their unique attributes are, in respective to the order, computational simplicity, state-based estimation, state-based estimation with neural network-derived features, and stated-based estimation with low-confidence thresholds.

  All trackers were benchmarked on MOT17 and MOT20 and utilize the same bounding boxes from YOLOX-X, which is a near-SOTA detector retrained on various pedestrian dataset. This detector was taken from the original BYTE implementation. In additional to HOTA benchmarking, we also analyze the run time speeds of each tracker to establish feasibility in the real-time system. From this optical benchmarking analysis, we found that BYTE worked the best because of its usage of low-confidence detections to reduce dropped tracks. DeepSORT's appearance vectors allowed it to performance notably better than SORT on MOT20. This can be due to MOT20's higher pedestrian density making simple position information not as reliable as a combination of position and appearances.

  Run time analysis on a traditional desktop CPU, Jetson Xavier AGX, and Jetson Xavier NX shows that traditional DeepSORT is not viable for embedded system working in high density environments (such as ones in MOT20). BYTE offers the most balance in terms of high HOTA performance and real-time speeds while IoUTracker could be feasible for low-density environments on a low-power device. For the cross-modal association task, we utilize a simple support vector machine (SVM) to do binary classification against a complete bipartite graph. The two sets within the bipartite graph are detections from RADAR and RGB detections respectively. Cross validation is done on the SVM with a fold value of 10 and the validation accuracy was 96.26% and testing accuracy was 94.25%.

7.3 Future Work

This section will detail some future conceptual ideas related to the goals that the thesis methods attempted to solve. Regarding localization methods, the CNN-based detectors showed promising accuracy with MVDR-based RADAR data, but there are additional enhancements that could further robustify the detectors. One such example is incorporating

temporal data into the network via a LSTM or other method, similar to Meng et. al [36]. This could help in cases where the signature disappears for more than one frame. Incorporating this temporal data into the MOT trackers' methodology could also help increase HOTA scores by decreasing the number of new, mistaken track IDs.

Regarding the current RADAR setup, the vital sign estimation requires users to stay stationary for a clear heartbeat or respiration signal. This limits practical utilization, so extending vital sign estimation to moving subjects is an active area that the overarching project is researching. One practical solution is to deploy multiple VMD3s within a constrained environment to help reduce signal noise of a subject (Figure 7.1). There are also existing methods, such as MoRe-Fi by Zheng et. al, that utilizes a Variational Encoder and Decoder (VED) network that can take in complex data and recover respiration waveforms from moving subjects [61]. Extending this neural network-based work to fit a broader array of movement could help with increasing vital sign accuracy.

In addition to the multi-sensor system, the RADAR data in this thesis was a simple 2D matrix that was interpreted as an "image", but using a RADAR device that can provide real-time, three dimensional matrices could help improve all aspects of the system. One example is vital sign targeting. At the moment, the system works best if an individual's chest is near level with the VMD3, but having a 3D reconstruction of the environment can help us target an individual's chest without restraining pose or movement. Additionally, a 3D environment and an employment of pose estimation from previous works can help increase localization and classification accuracy. Although, it is important to keep in mind that all of this additional data and more sophisticated methods could require higher computer power requirements.

Regarding cross-modality, there are works related to optical heart-rate and respiration rate estimation that are actively being developed for this sy stem. The camera interface provides the needed information for this future estimation method. Cross-modality association can help fuse both RADAR and optical-based estimation for an overall more accurate estimation. In terms of the cross-modality method, there are some drawbacks. One such drawback is the constraint of camera position relative to the RADAR device. If the camera, for whatever reason, was at a much larger height or at an angle compared to the RADAR device, there could be a miscalibration between the two devices' location output. Methods that can derive a perspective characteristic from an RGB image could possibly compensate for differing RADAR and camera configurations.

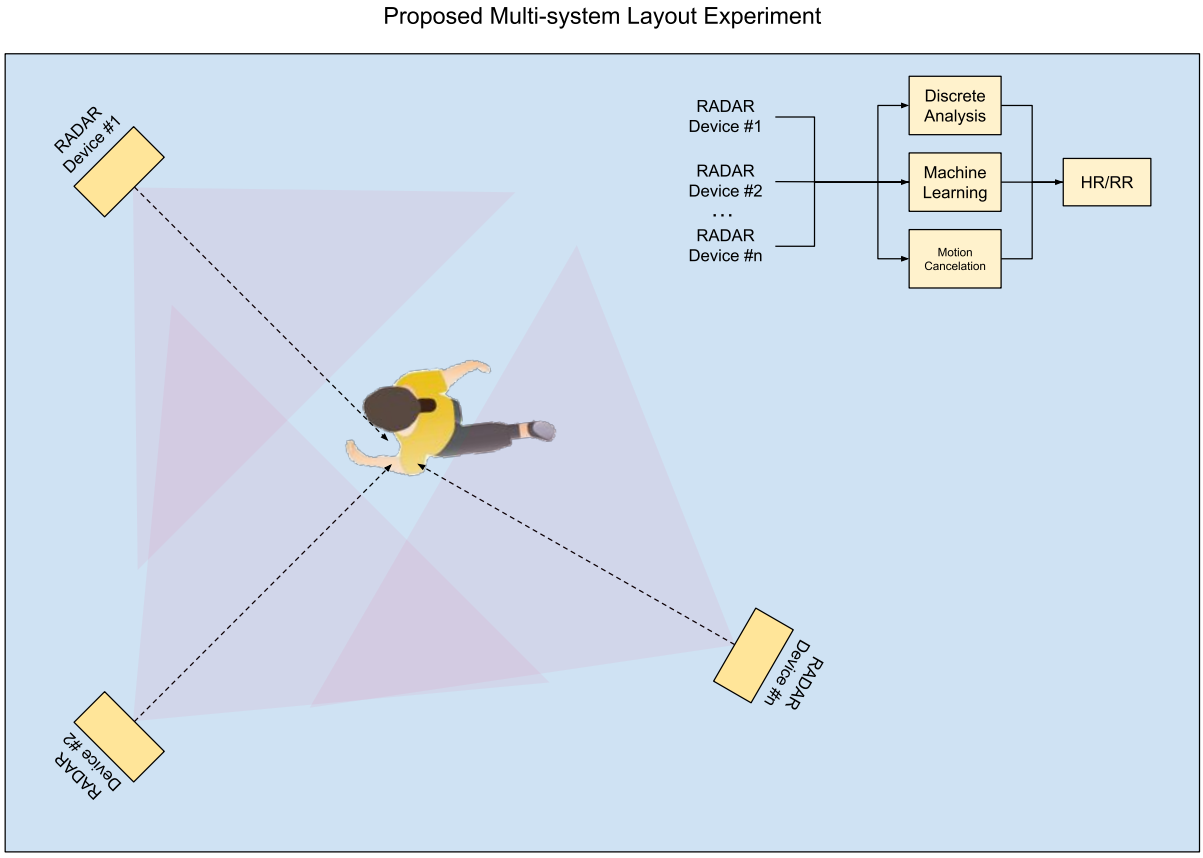


Figure 7.1: Proposed multi-RADAR system for monitoring moving subject respiration and heartrate. *Previous works by [61] have shown the possibility of using encoder-decoder neural networks to remove movement noise from a RADAR device. Exploring different network structures for this noise removal as well as incorporating more than two RADAR devices can improve respiration estimation method robustness. This also implies that the current real-time system described in Chapter 6 would need to be adjusted to handle additional sensor processing. A reconsideration of compute devices would be necessary to handle this amount of RADAR data.*

Biblography

[1] Adib, F., Hsu, C.-Y., Mao, H., Katabi, D., and Durand, F. (2015). Capturing the human figure through a wall. *ACM Trans. Graph.*, 34(6). x, 11, 13

[2] Arani, E., Gowda, S., Mukherjee, R., Magdy, O., Kathiresan, S., and Zonooz, B. (2022). A comprehensive study of real-time object detection networks across multiple domains: A survey. 18

[3] Banitalebi-Dehkordi, A., Azimi, M., Pourazad, M. T., and Nasiopoulos, P. (2014). Compression of high dynamic range video using the hevc and h.264/avc standards. In *10th International Conference on Heterogeneous Networking for Quality, Reliability, Security and Robustness*, pages 8–12. 72, 93

[4] Benjak, J., Hofman, D., Knezović, J., and Žagar, M. (2022). Performance comparison of h.264 and h.265 encoders in a 4k fpv drone piloting system. *Applied Sciences*, 12(13). 93

[5] Bewley, A., Ge, Z., Ott, L., Ramos, F., and Upcroft, B. (2016). Simple online and realtime tracking. In *2016 IEEE International Conference on Image Processing (ICIP)*, pages 3464–3468. 20

[6] Bochkovskiy, A., Wang, C.-Y., and Liao, H.-Y. M. (2020). Yolov4: Optimal speed and accuracy of object detection. 18

[7] Bégaint, J., Racapé, F., Feltman, S., and Pushparaja, A. (2020). Compressai: a pytorch library and evaluation platform for end-to-end compression research. xix, 73, 86

[8] Carion, N., Massa, F., Synnaeve, G., Usunier, N., Kirillov, A., and Zagoruyko, S. (2020). End-to-end object detection with transformers. 18

[9] Cheng, Z., Sun, H., Takeuchi, M., and Katto, J. (2020). Learned image compression with discretized gaussian mixture likelihoods and attention modules. xvii, xix, 73, 77, 86

[10] Ciaparrone, G., Luque Sánchez, F., Tabik, S., Troiano, L., Tagliaferri, R., and Herrera, F. (2020). Deep learning in video multi-object tracking: A survey. *Neurocomputing*, 381:61–88. 20

[11] Cornett, D. C. (2021). *Evaluation of robust deep learning pipelines targeting low SWaP Edge deployment*. PhD thesis, University of Tennessee, Knoxville. viii, xii, 27, 29, 30, 34, 38

[12] Darling, G. (2021). Iot vs. edge computing: What's the difference? 34

[13] Dendorfer, P., Rezatofighi, H., Milan, A., Shi, J., Cremers, D., Reid, I., Roth, S., Schindler, K., and Leal-Taixé, L. (2020). Mot20: A benchmark for multi object tracking in crowded scenes. 43

[14] Electronics, W. (2022). 5.5inch capacitive touch amoled display, 1080×1920, hdmi, toughened glass cover. https://www.waveshare.com/5.5inch-hdmi-amoled.htm. 35

[15] Everingham, M., Van Gool, L., Williams, C. K. I., Winn, J., and Zisserman, A. (2010). The pascal visual object classes (voc) challenge. *International Journal of Computer Vision*, 88(2):303–338. 42

[16] Fabbri, M., Brasó, G., Maugeri, G., Cetintas, O., Gasparini, R., Osep, A., Calderara, S., Leal-Taixé, L., and Cucchiara, R. (2021). Motsynth: How can synthetic data help pedestrian detection and tracking? *CoRR*, abs/2108.09518. xi, 23, 24, 25

[17] Federico Busato, J. P. (2020). Exploiting nvidia ampere structured sparsity with cusparselt. 24

[18] Feher, B., Sidi, L., Shabtai, A., and Puzis, R. (2016). The security of webrtc. 102

[19] Gale, T., Elsen, E., and Hooker, S. (2019). The state of sparsity in deep neural networks. *CoRR*, abs/1902.09574. 24

[20] Ge, Z., Liu, S., Wang, F., Li, Z., and Sun, J. (2021). YOLOX: exceeding YOLO series in 2021. *CoRR*, abs/2107.08430. 18, 65

[21] Google (2022). Webp compression study. https://developers.google.com/speed/webp/docs/webp_study. 72

[22] Han, D. (2013/03). Comparison of commonly used image interpolation methods. In *Proceedings of the 2nd International Conference on Computer Science and Electronics Engineering (ICCSEE 2013)*, pages 1556–1559. Atlantis Press. 73

[23] Han, S., Pool, J., Tran, J., and Dally, W. J. (2015). Learning both weights and connections for efficient neural networks. *CoRR*, abs/1506.02626. 24

[24] Jiao, L., Zhang, F., Liu, F., Yang, S., Li, L., Feng, Z., and Qu, R. (2019). A survey of deep learning-based object detection. *IEEE Access*, 7:128837–128868. xi, 16, 17

[25] Jocher, G., Kwon, Y., guigarfr, perry0418, Veitch-Michaelis, J., Ttayu, Suess, D., Baltacı, F., Bianconi, G., IlyaOvodov, Marc, e96031413, Lee, C., Kendall, D., Falak, Reveriano, F., FuLin, GoogleWiki, Nataprawira, J., Hu, J., LinCoce, LukeAI, NanoCode012, NirZarrabi, Reda, O., Cohen, P., Skalski, P., SergioSanchezMontesUAM, Song, S., and Shead, T. M. (2021). ultralytics/yolov3: v9.6.0 - YOLOv5 v6.0 release compatibility update for YOLOv3. 18, 73

[26] Jonathon Luiten, A. H. (2020). Trackeval. https://github.com/JonathonLuiten/TrackEval. 65

[27] Kharat, A., Duddalwar, V., Saoji, K., Gaikwad, A., Kulkarni, V., Naik, G., Lokwani, R., Kasliwal, S., Kondal, S., Gupte, T., and Pant, A. (2020). Role of edge device and cloud machine learning in point-of-care solutions using imaging diagnostics for population screening. 2

[28] Leach, S. (2021). Analysis of Hardware Accelerated Deep Learning and the Effects of Degradation on Performance. *Masters Theses*. 27

[29] Li, Y., Huang, C., and Nevatia, R. (2009). Learning to associate: Hybridboosted multi-target tracker for crowded scene. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 2953–2960. 21

[30] Liang, T., Glossner, J., Wang, L., and Shi, S. (2021). Pruning and quantization for deep neural network acceleration: A survey. *CoRR*, abs/2101.09671. 24

[31] Lin, T.-Y., Maire, M., Belongie, S., Bourdev, L., Girshick, R., Hays, J., Perona, P., Ramanan, D., Zitnick, C. L., and Dollár, P. (2014a). Microsoft coco: Common objects in context. 43

[32] Lin, T.-Y., Maire, M., Belongie, S., Hays, J., Perona, P., Ramanan, D., Dollár, P., and Zitnick, C. L. (2014b). Microsoft coco: Common objects in context. In *European conference on computer vision*, pages 740–755. Springer. xi, 17

[33] Liu, H., Alali, A., Ibrahim, M., Cao, B. B., Meegan, N., Li, H., Gruteser, M., Jain, S., Dana, K., Ashok, A., Cheng, B., and Lu, H. (2022). Vi-fi: Associating moving subjects across vision and wireless sensors. In *2022 21st ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN)*, pages 208–219. xi, 22, 23

[34] Liu, Y., Wang, Z., Zhou, X., and Zheng, L. (2021). Synthetic data are as good as the real for association knowledge learning in multi-object tracking. *CoRR*, abs/2106.16100. 25

[35] Martin, S. (2020). Skycatch high-precision package for mining 2020. 30

[36] Meng, K. and Meng, Y. (2019). Through-wall pose imaging in real-time with a many-to-many encoder/decoder paradigm. xi, 14, 15, 128

[37] Milan, A., Leal-Taixe, L., Reid, I., Roth, S., and Schindler, K. (2016). Mot16: A benchmark for multi-object tracking. 26, 43

[38] Nagel, M., Fournarakis, M., Amjad, R. A., Bondarenko, Y., van Baalen, M., and Blankevoort, T. (2021). A white paper on neural network quantization. *CoRR*, abs/2106.08295. 22

[39] Neuralmagic (2021). Part 1: What is pruning in machine learning? - neural magic. 24

[40] NVIDIA (2018). Nvidia jetson agx xavier module - silicon highway. https://www.siliconhighway.com/wp-content/gallery/jetson-agx-xavier-module-datasheet-us-867901-r5-hr(1).pdf. 30

[41] NVIDIA (2019). Nvidia jetson xavier nx datasheet. https://siliconhighway.com/wp-content/gallery/jetson-xavier-nx-datasheet-us-1154975-r4-web-original.pdf. 30

[42] NVIDIA (2022a). Power management for jetson xavier nx and jetson agx xavier series devices. 117

[43] NVIDIA (2022b). Quark carrier for nvidia® jetson xavier™ nx. https://connecttech.com/product/quark-carrier-nvidia-jetson-xavier-nx/. 39

[44] Park, N. and Kim, S. (2022). How do vision transformers work? 73

[45] Poyser, M., Atapour-Abarghouei, A., and Breckon, T. P. (2020). On the impact of lossy image and video compression on the performance of deep convolutional neural network architectures. xviii, 27, 72, 81

[46] Redmon, J., Divvala, S. K., Girshick, R. B., and Farhadi, A. (2015). You only look once: Unified, real-time object detection. *CoRR*, abs/1506.02640. 18

[47] Redmon, J. and Farhadi, A. (2016). YOLO9000: better, faster, stronger. *CoRR*, abs/1612.08242. 18

[48] Redmon, J. and Farhadi, A. (2018). Yolov3: An incremental improvement. 18

[49] Rippel, O. and Bourdev, L. (2017). Real-time adaptive image compression. 93

[50] Seals, M. (2019). *On the Robustness of Object Detection Based Deep Learning Models*. PhD thesis, University of Tennessee. 27

[51] Stiefelhagen, R., Bernardin, K., Bowers, R., Garofolo, J., Mostefa, D., and Soundararajan, P. (2007). The clear 2006 evaluation. In Stiefelhagen, R. and Garofolo, J., editors, *Multimodal Technologies for Perception of Humans*, pages 1–44, Berlin, Heidelberg. Springer Berlin Heidelberg. xii, 26, 28

[52] Thakur, A., Papakipos, Z., Clauss, C., Hollinger, C., Boivin, V., Lowe, B., Schoentgen, M., and Bouckenooghe, R. (2022). abhitronix/vidgear: Vidgear v0.2.5. 102

[53] Wang, C.-Y., Bochkovskiy, A., and Liao, H.-Y. M. (2020). Scaled-yolov4: Scaling cross stage partial network. 18

[54] Wang, Y., Jiang, Z., Gao, X., Hwang, J.-N., Xing, G., and Liu, H. (2021a). Rodnet: Radar object detection using cross-modal supervision. In *2021 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pages 504–513. xi, 15, 16

[55] Wang, Z., Zhao, H., Li, Y.-L., Wang, S., Torr, P. H. S., and Bertinetto, L. (2021b). Do different tracking tasks require different appearance models? xi, 19

[56] Wojke, N., Bewley, A., and Paulus, D. (2017). Simple online and realtime tracking with a deep association metric. 21

[57] Xinyu, W. (2022). Tensorrtx. https://github.com/wang-xinyu/tensorrtx. 106

[58] Zhang, Y., Sun, P., Jiang, Y., Yu, D., Yuan, Z., Luo, P., Liu, W., and Wang, X. (2021). Bytetrack: Multi-object tracking by associating every detection box. 21, 65

[59] Zhao, M., Li, T., Alsheikh, M. A., Tian, Y., Zhao, H., Torralba, A., and Katabi, D. (2018). Through-wall human pose estimation using radio signals. *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 7356–7365. 12, 14

[60] Zheng, S. and Ding, C. (2018). Minimal support vector machine. 61

[61] Zheng, T., Chen, Z., Zhang, S., Cai, C., and Luo, J. (2021). MoRe-fi. In *Proceedings of the 19th ACM Conference on Embedded Networked Sensor Systems*. ACM. xxiii, 128, 129

[62] Zhu, Q., Chen, M., Wong, C.-W., and Wu, M. (2021). Adaptive multi-trace carving for robust frequency tracking in forensic applications. *IEEE Transactions on Information Forensics and Security*, 16:1174–1189. 99

[63] Zhu, Y., Cai, H., Zhang, S., Wang, C., and Xiong, Y. (2020). Tinaface: Strong but simple baseline for face detection. 57

[64] Zhuang, B., Liu, L., Tan, M., Shen, C., and Reid, I. (2020). Training quantized neural networks with a full-precision auxiliary module. 22

Appendix

Summary of Training Details and Results

This appendix is an extension of the training details from Chapter 4. We include graphs of the localization loss and mAP @ IoU=0.5 across the training epochs. Additionally, training hyperparameters, such as epoch length and learning rates, are provided for the sake of completeness.

Table 1: Hyperparameters for synthetic RADAR network training. *Other hyperparameters associated with a given network are the default values of their COCO training.*

Model Type	Epochs	Learning Rate	Batch Size
YOLOv5L	5	0.0005	16
YOLOv5n	15	0.0005	16
YOLOv4	5	0.0005	16
YOLOv4 Tiny	10	0.0005	16
YOLOv4 CSP	5	0.0005	8
YOLOv4 CSP Small	6	0.0005	16
YOLOv3	5	0.0005	16
YOLOv3 Tiny	4	0.0005	16

Table 2: Hyperparameters for real data RADAR network training. *Other hyperparameters associated with a given network are the default values of their COCO training.*

Model Type	Epochs	Learning Rate	Batch Size
YOLOv5n	20	0.0005	16
YOLOv4 Tiny	10	0.0005	16
YOLOv4 CSP Small	10	0.0005	16
YOLOv3 Tiny	15	0.0005	16

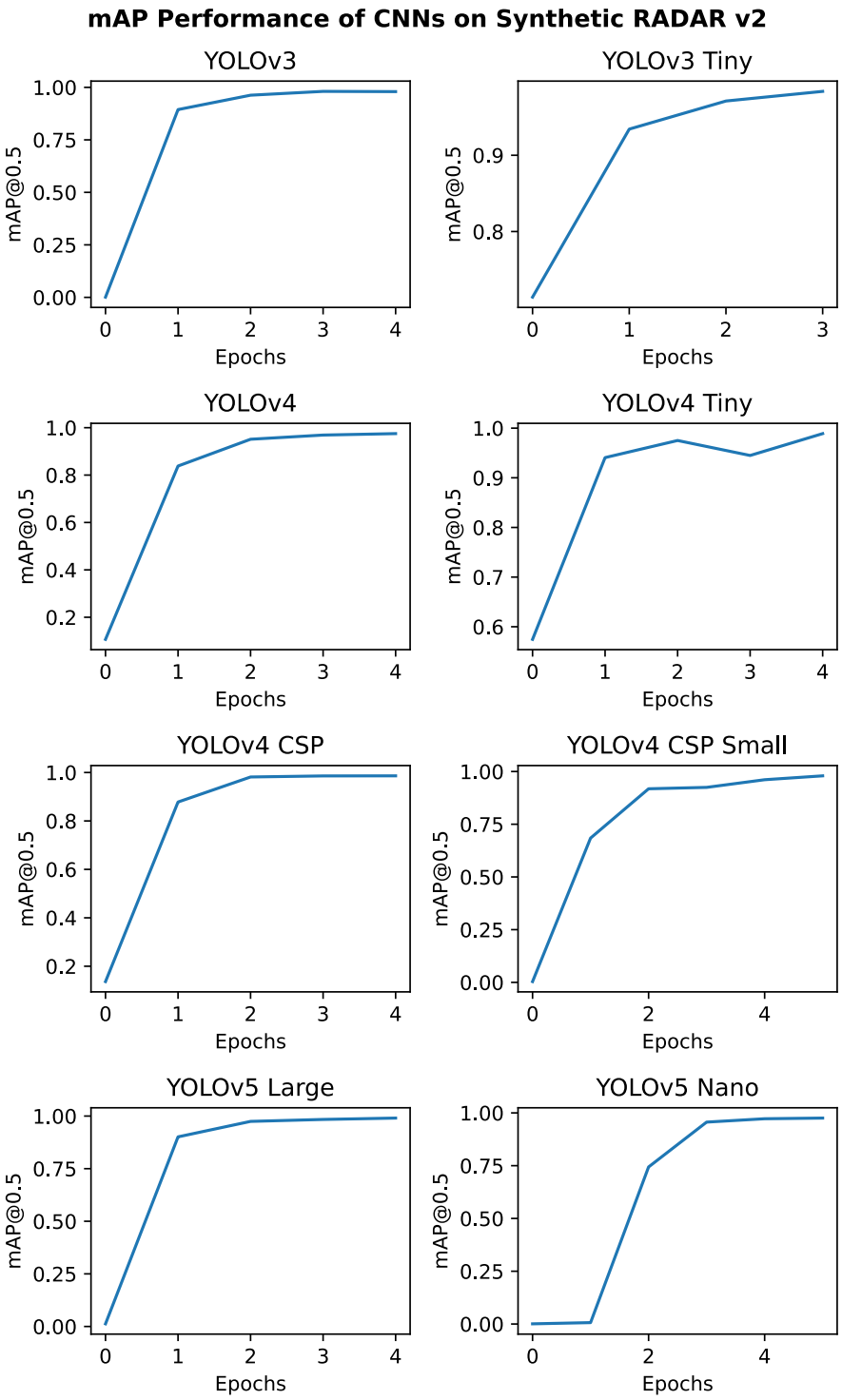


Figure 2: Training mAP graph for YOLO-based networks on Synthetic RADAR v2.

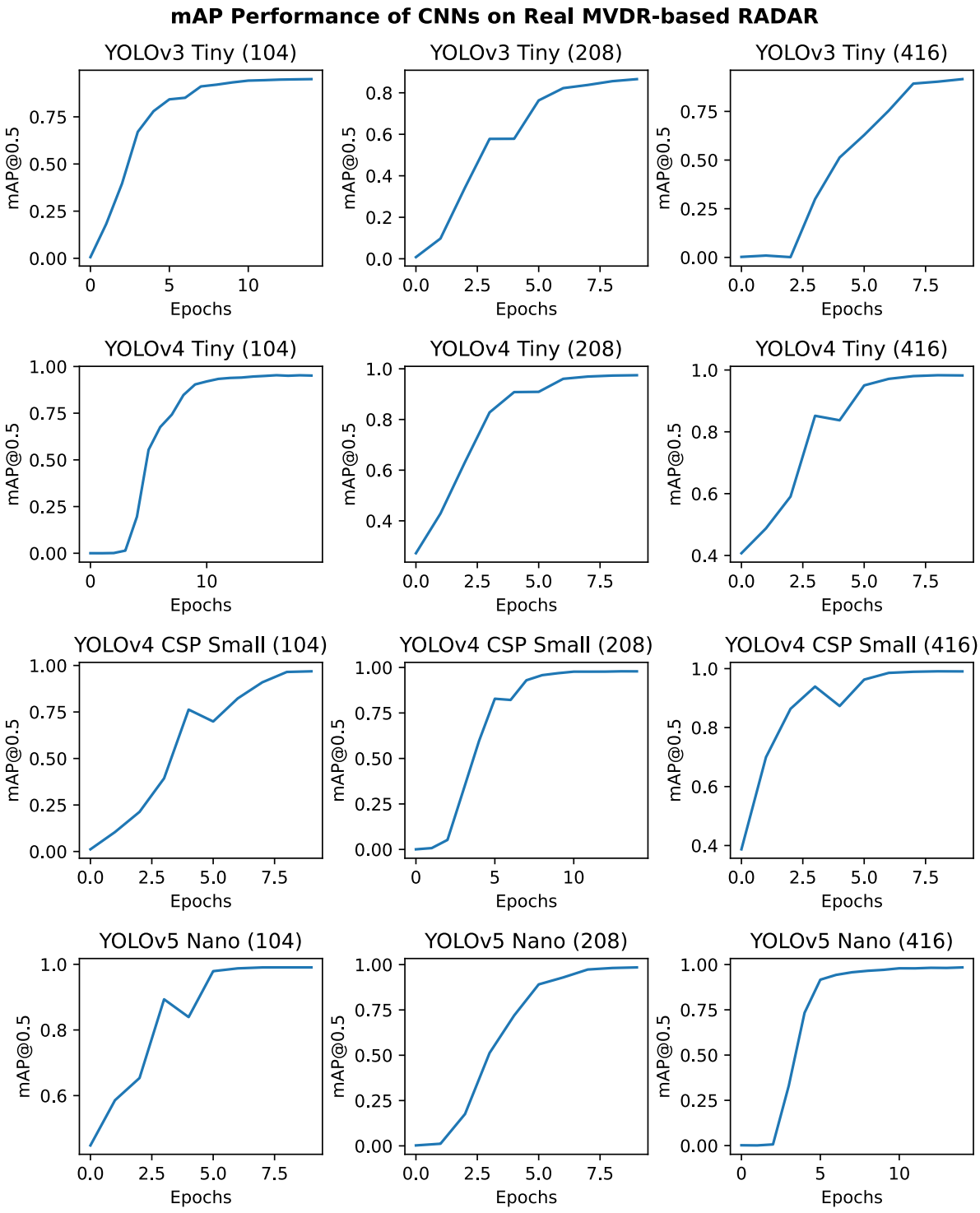


Figure 3: Training mAP graph for YOLO-based networks on real-time MVDR-based RADAR data.

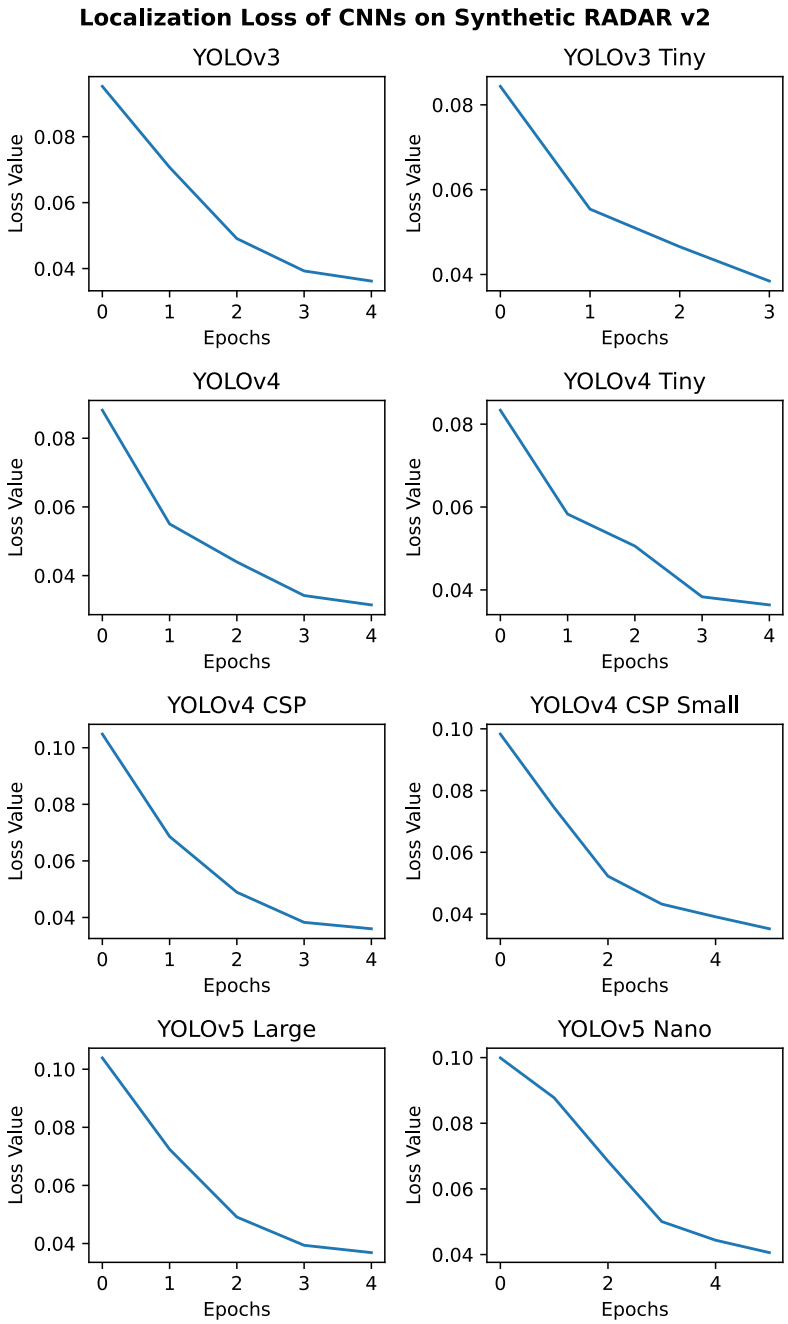


Figure 4: Training localization loss graph for YOLO-based networks on Synthetic RADAR v2.

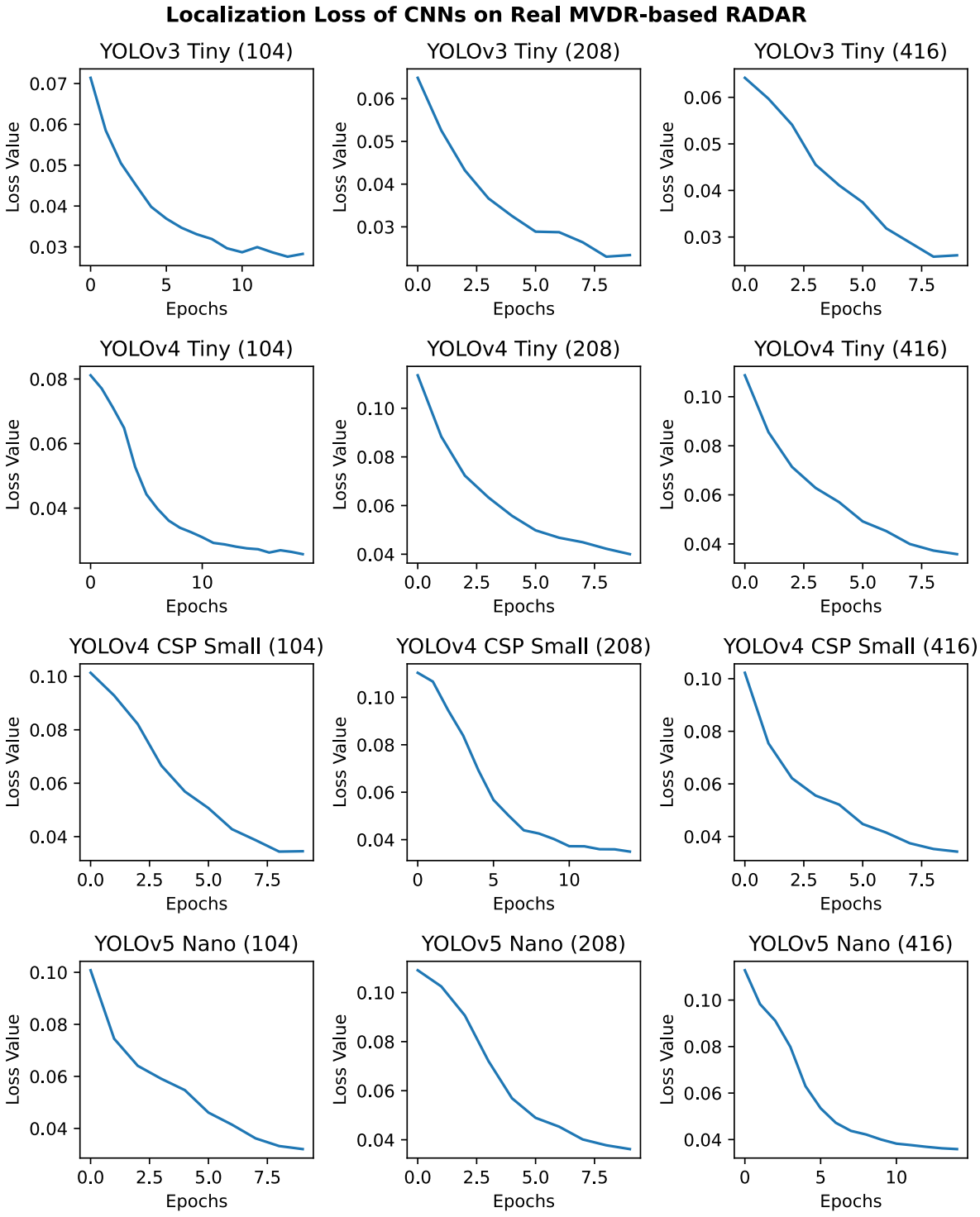


Figure 5: Training localization loss graph for YOLO-based networks on real-time MVDR-based RADAR data.

Multi-Object Tracking Detailed Results

In Chapter 4, MOT tracker speeds across three hardware platforms is discussed, but the averages for each MOT dataset are only shown. To get a better idea of the range of speeds for each MOT clip, Figure 6, 7, and 8 are shown.

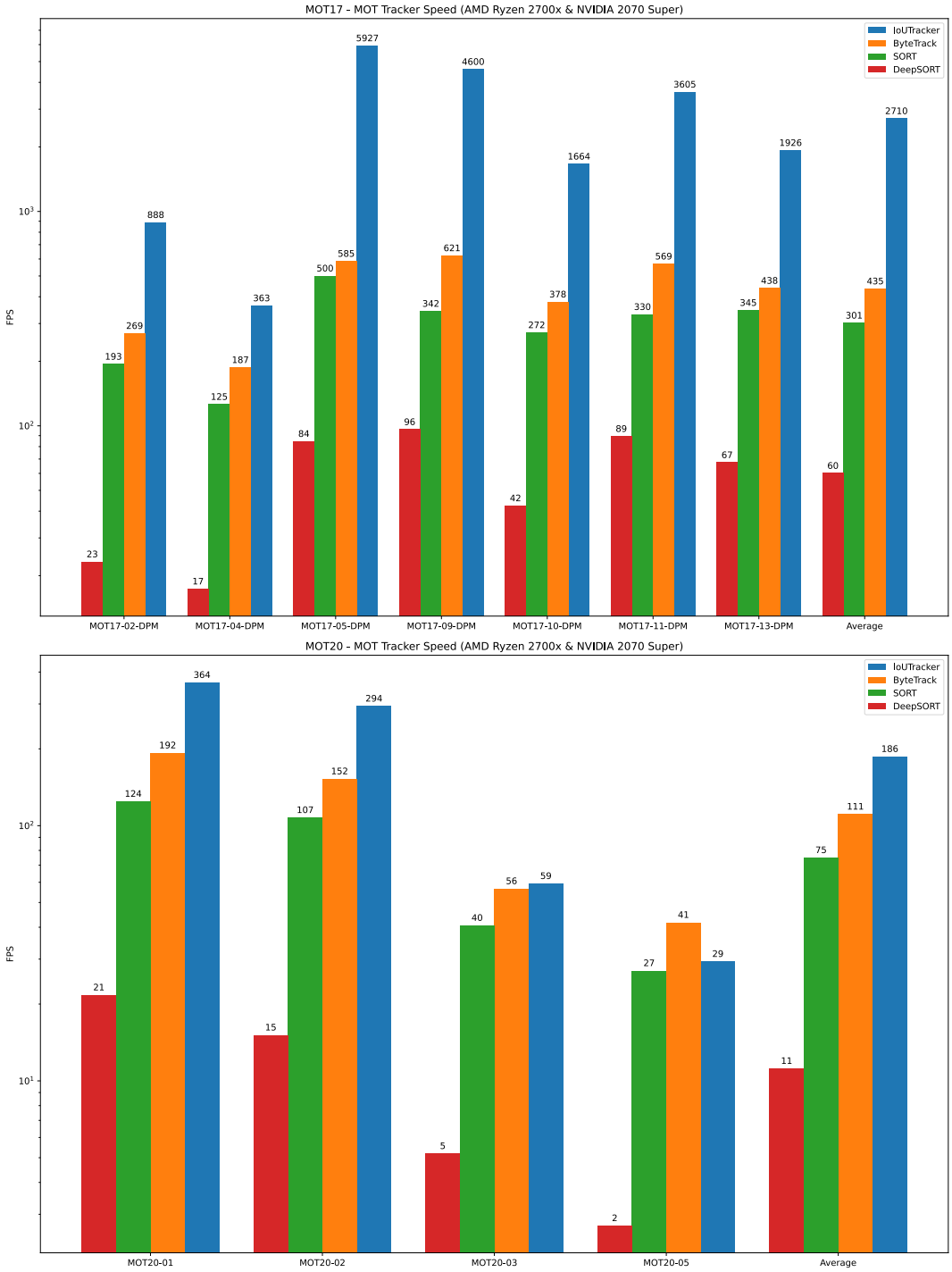


Figure 6: Comparison of tracking method speeds on Desktop CPU and GPU. *All methods ran on the CPU (AMD Ryzen 2700X), but due to its small CNN feature extractor, DeepSORT also utilized the Desktop GPU (NVIDIA 2070 Super). The model was kept at FLOAT32 and had a batchsize that changed to fit the total number of individuals detected. The fastest method was IoU Tracker, and this is due to the lack of advanced state estimation or neural networks, which are present in the other methods.*

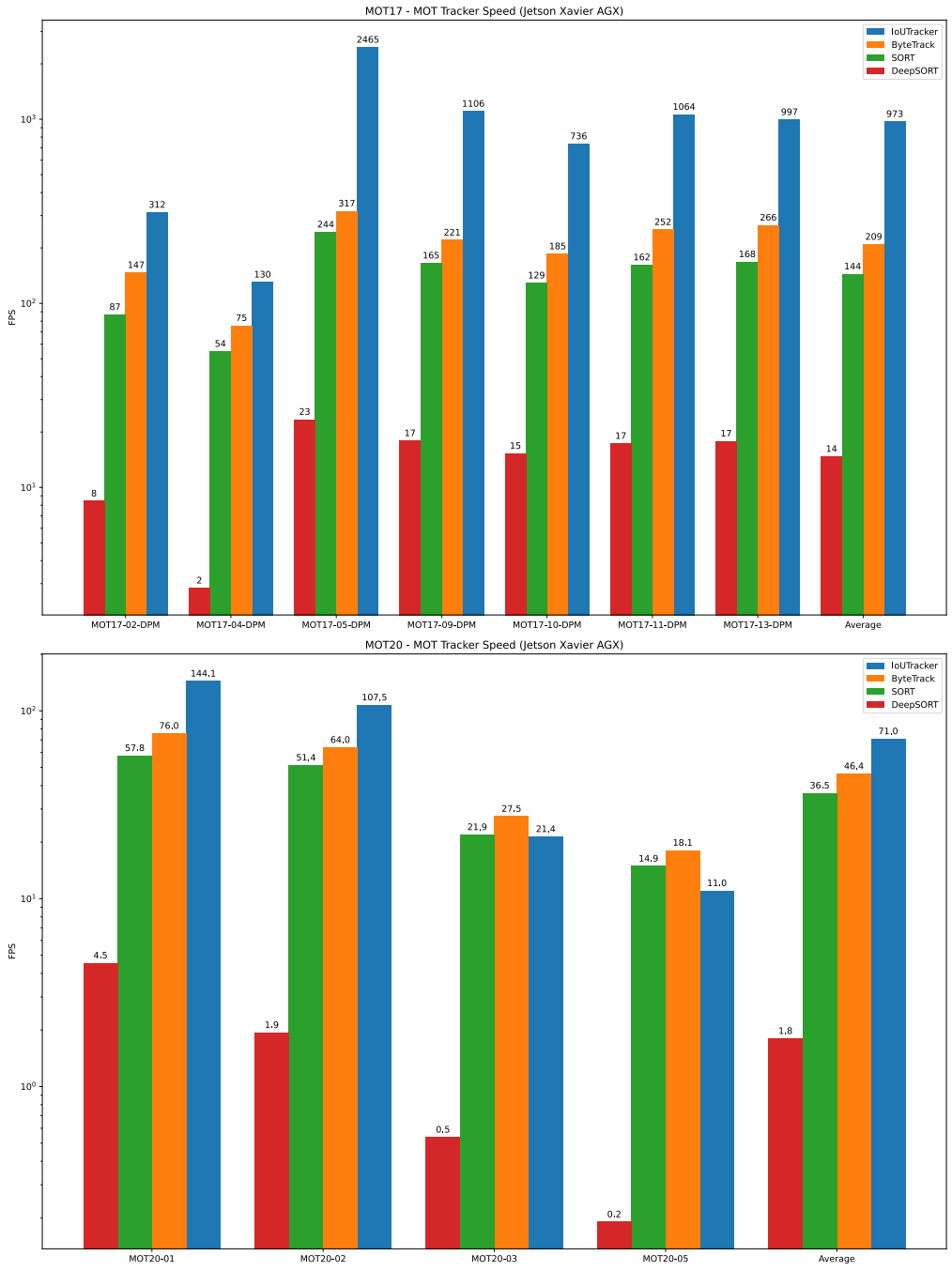


Figure 7: Comparison of tracking method speeds on Jetson Xavier AGX. *All methods ran on the same hardware layout as described in Figure 4.19. The feature extractor in DeepSORT was converted to TensorRT. It was quantized to FP16 to give it an equal footing with other methods while not reducing any potential mAP loss via PTQ. Trends seen in Figure 4.19 mostly maintain in the AGX runtimes, with the only difference being BYTE outperforming IoUTracker in MOT20-03. The AGX runtimes were on average 2.3x slower for BYTE, SORT, and IoUTracker across both datasets. For DeepSORT, MOT17 runtimes were 4.1x slower and MOT20 runtimes were 6.2x slower.*

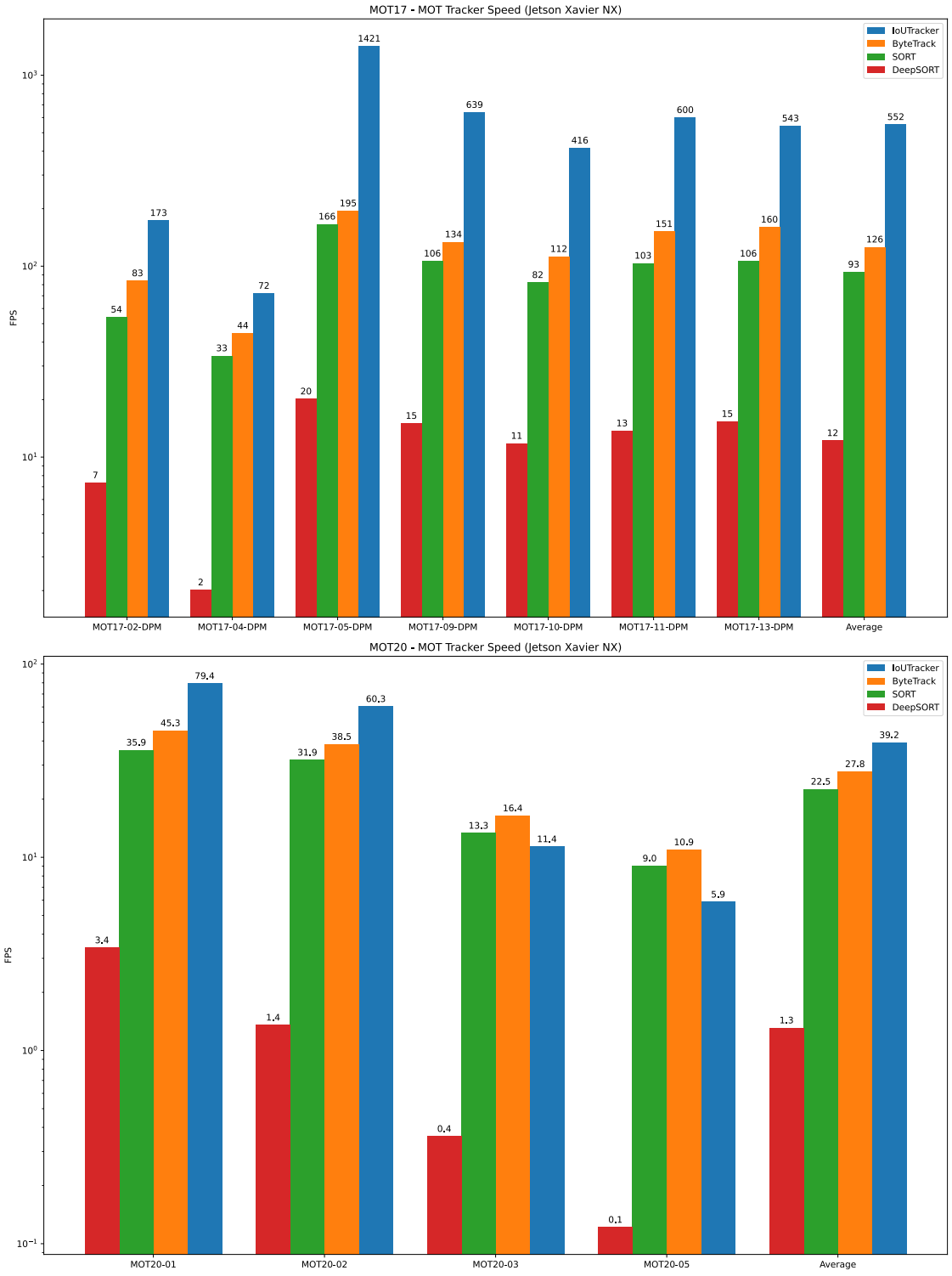


Figure 8: Comparison of tracking method speeds on Jetson Xavier NX. *Trends seen in the AGX runtimes continue in the NX runtimes. The NX runtimes were on average 3.9x slower for BYTE, SORT, and IoUTracker across both datasets. For DeepSORT, MOT17 runtimes were 5x slower and MOT20 runtimes were 8.5x slower.*

YOLO Network Performance

This table is related to the YOLO-baseed object detector discussion in Chapter 2.

Table 3: A table showing the overview of different types of YOLO networks and their mAP performance. *Evaluation is done on two main detection datasets - VOC2007 and COCO (2014 and 2017 versions). For papers that offered multiple network configurations, the most accurate versions are displayed.*

YOLO Type (Input size)	Testset	mAP (threshold=0.5)
YOLO (448px x 448px)	VOC2007	63.4%
YOLOv2 (544px x 544px)	VOC2007	78.6%
YOLOv2 (544px x 544px)	COCO2014	44.0%
YOLOv3 (416px x 416px)	COCO2014	55.3%
YOLOv3 (608px x 408px)	COCO2017	57.9%
YOLOv4 (608px x 608px)	COCO2017	65.7%
YOLOv5x6 + TTA (1536px x 1536px)	COCO2017	72.7%
Scaled-YOLOv4 P7 (1556px x 1556px)	COCO2017	73.4%
YOLOX-X (640px x 640px)	COCO2017	69.6%
YOLOv7-E6E (1280px x 1280px)	COCO2017	74.4%

Heartrate and Respiration Rate Errors

Table 4: Table of respiration and heartrate errors based on different ranges. *The table also breaks error down based on single and multi-subject recordings. 1 meter recordings have the best performance regardless of number of subjects.*

	Single Subject MAE (BPM)		Multi-Subject MAE (BPM)	
	RR	HR	RR	HR
1m	0.37	2.32	0.49	2.64
2m	0.51	2.69	0.92	2.98
3m	0.16	2.56	0.63	2.48

Vita

Vijaysrinivas Rajagopal was born on Feburary 28th, 1999 in Chennai, India. He graduated from the university of Tennessee, Knoxville with a Bachelor of Science in Computer Science in 2021. After completing his undergraduate degree, Vijay began his Master's Degree in Computer Science under Dr. Mongi A. Abidi. He also had the opportunity to work on a funded graduate project from Dr. Aly Fathy.