Efficient Deep Learning and Its Applications

A Dissertation Presented for the

Doctor of Philosophy

Degree

The University of Tennessee, Knoxville

Zi Wang

May 2022

Abstract

Deep neural networks (DNNs) have achieved huge successes in various tasks such as object classification and detection, image synthesis, game-playing, and biological developmental system simulation. State-or-the-art performance on these tasks is usually achieved by designing deeper and wider DNNs with the cost of huge storage size and high computational complexity. However, the over-parameterization problem of DNNs constrains their deployment in resource-limited devices, such as drones and mobile phones.

With these concerns, many network compression approaches are developed, such as quantization, neural architecture search, network pruning, and knowledge distillation. These approaches reduce the sizes and computational costs of DNNs while maintaining their performance.

In this dissertation, we first focus on two of the most popular network compression schemes, i.e., network pruning and knowledge distillation. We aim to (1) develop more efficient network pruning approaches that can remove a large percentage of parameters/FLOPs from the DNNs while minimizing the performance degradation, and (2) train compact neural networks with the help of large, pre-trained networks under challenging scenarios in which limited information of the pre-trained networks are accessible. In the second part, we will develop efficient deep learning algorithms for a real-world application, i.e., modeling the biological cell migration process with deep reinforcement learning. The main contribution of this dissertation is summarized as follows.

- We propose a novel network pruning approach, which removes filters based on the redundancy measurement in each layer. Different from existing works that prune the

least important filters across all layers, we find that pruning filters from the layer with the most redundancy performs better.

- We study knowledge distillation, which trains a compact network by mimicking the output of a pre-trained, over-parameterized network, under more challenging scenarios. In specific, we explore the possibility to learn from the pre-trained model when (1) the training set is not accessible and (2) the pre-trained model only returns top-1 index rather than probabilities.

- We leverage efficient deep learning tools in the cell migration modeling with reinforcement learning, which helps reduce the training time. Therefore, novel biological mechanisms can be discovered within an acceptable period of time.

Table of Contents

1 Introduction **1**

1.1 Motivation and challenges 1

1.2 Dissertation overview 4

2 Background and Related Works **5**

2.1 Neural network compression 5

2.1.1 Overview 5

2.1.2 Network pruning 5

2.1.3 Knowledge distillation 7

2.2 Reinforcement learning 9

2.3 Cell migration modeling 9

I Neural Network Compression with Channel Pruning and Knowledge Distillation **11**

3 Network Pruning via Structural Redundancy Reduction **12**

3.1 Overview 12

3.2 A theoretical analysis on network pruning: a statistical perspective 14

3.2.1 Rethinking network pruning from the perspective of structural redundancy reduction 14

3.2.2 Empirical study of the theoretic analysis 18

3.3 Methodology 25

3.3.1 Notations and preliminaries . . . 25
3.3.2 Overall framework . . . 26
3.3.3 Graph establishment . . . 28
3.3.4 ℓ-covering number . . . 28
3.3.5 Decomposition of a graph . . . 28
3.3.6 Graph redundancy, intuition and quantification . . . 29
3.3.7 Estimate of the 1-covering number. . . . 29
3.3.8 Filter selection strategy . . . 31
3.4 Experiments . . . 32
3.4.1 Setup . . . 32
3.4.2 Performance evaluation of single-shot pruning . . . 34
3.4.3 Performance evaluation of progressive pruning . . . 34
3.5 Analysis and ablation study . . . 38
3.5.1 ℓ-covering number estimate and computation time . . . 38
3.5.2 Filter selection criteria . . . 42
3.5.3 The value of gamma . . . 42
3.5.4 Other criteria for structural redundancy identification . . . 45
3.6 Conclusion . . . 46

4 Zero-Shot Knowledge Distillation from a Decision-Based Black-Box Model 48
4.1 Overview . . . 48
4.2 Knowledge distillation . . . 51
4.3 Decision-based black-box knowledge distillation . . . 51
4.3.1 Sample robustness . . . 53
4.3.2 Soft label construction . . . 55
4.3.3 Training of student model . . . 57
4.4 Zero-shot decision-based black-box knowledge distillation . . . 57
4.5 Experiments . . . 58
4.5.1 Experiment setup of DB3KD . . . 59

4.5.2 Performance evaluation of DB3KD . . . 60
4.5.3 Ablation studies and analyses of DB3KD . . . 63
4.5.4 Experiment setup of ZSDB3KD . . . 66
4.5.5 Performance comparison of ZSDB3KD . . . 67
4.5.6 Ablation studies and analyses of ZSDB3KD . . . 67
4.6 Conclusion . . . 71

II Modeling Cell Migration with Efficient Deep Reinforcement Learning 72

5 Neighbor Relationship Determination in Metazoan Embryos 73
5.1 Overview . . . 73
5.2 Methodology . . . 75
5.2.1 Voronoi diagram . . . 75
5.2.2 Framework . . . 77
5.2.3 Feature extraction . . . 77
5.2.4 Classification . . . 79
5.3 Experiments . . . 80
5.3.1 Simulation setup . . . 80
5.3.2 Runtime performance and accuracy . . . 81
5.3.3 Verification and visualization . . . 84
5.4 Discussions . . . 87
5.4.1 Analysis of the accuracy rate . . . 87
5.4.2 Customization on different purposes . . . 96
5.5 Conclusion . . . 97

6 Cell Migration Modeling with Deep Reinforcement Learning 98
6.1 Overview . . . 98
6.2 Modeling approach . . . 100

6.2.1 Individual cell movements . . . 101
6.2.2 Collective cell migration . . . 102
6.3 Methods . . . 102
6.3.1 ABM framework . . . 102
6.3.2 Cell movement via deep Q-network . . . 102
6.3.3 Behaviors of the dumb cells . . . 109
6.3.4 Behaviors of the intelligent cell . . . 109
6.4 Experiments . . . 112
6.4.1 Computational environment and platform . . . 112
6.4.2 Model setup . . . 112
6.4.3 An agent-based deep reinforcement learning framework for *C. elegans* embryogenesis . . . 113
6.4.4 Regulatory mechanisms of individual cell movements . . . 114
6.4.5 Regulatory mechanisms of group cell migration . . . 121
6.5 Discussion . . . 121
6.6 Conclusion . . . 125

7 Efficient Cell Migration Modeling with Hierarchical Deep Reinforcement Learning 126

7.1 Overview . . . 126
7.2 Design and scheme . . . 128
7.2.1 Image data and setup for reinforcement learning . . . 128
7.2.2 Reward construction . . . 129
7.2.3 HDRL for model formation . . . 131
7.2.4 Deployment of learned model through transfer learning . . . 132
7.3 HDRL model formation in C. elegans embryogenesis . . . 132
7.4 HDRL reveals modular organization of Cpaaa migration . . . 134
7.5 Collective behavior explains organization of cell movement . . . 136
7.6 Local cell-cell interactions underlie sequential rosettes . . . 137

7.7 Transfer learning from HDRL distinguishes movement patterns 140
7.8 Discussion 145
7.9 Methods 147
7.9.1 Observational dataset and annotation 147
7.9.2 Reinforcement learning system setup 147
7.9.3 Neighbor relationship model 148
7.9.4 Motion model 148
7.9.5 Neighbor distance model 149
7.9.6 HDRL model architecture and parameters 150
7.9.7 Transferred motion model 152
7.9.8 Analysis of feature maps 152
7.9.9 Microscopy, cell tracking and visualization 153
7.9.10 Characterization of Cpaaa migration paths 154
7.10 Conclusion 154

8 Discussion, Conclusion, and Future Work **155**

A Supplement for Chapter 7 **186**
A.1 Architecture of the CNN and the policy network in HDRL 186
A.2 MI curves of the successful migration scenarios using the Motion Model . . 186
A.3 Feature maps of Cpaaa migration at one of the early migration timesteps . . 186
A.4 Examples of individual difference maps at a single timestep 191
A.5 The number of pixels in the effective area of the summary difference map at each timestep in three embryos 191
A.6 Distribution for normalized minimal distance between neighbor cells 191

Vita **194**

List of Tables

3.1 The manually augmented network architectures used for the experiments. . 19

3.2 Results of ResNet on CIFAR-10. MW results is with our own implementation. GR is graph redundancy. 35

3.3 ResNet50 results on ImageNet. Results of MW, APoZ, and Taylor are based on our own implementation. 36

3.4 Average computation time of the oracle and our proposed method for the 1-covering number calculation/estimate (in secs). 41

3.5 Performance with different filter selection criteria after pruning 30% FLOPs of AlexNet. 43

3.6 Performance of the pruned ResNet50 networks with different redundancy reduction measurements. 47

4.1 Performance evaluation of the proposed DB3KD approach on small datasets (MNIST, Fashion-MNIST, and CIFAR-10). 61

4.2 Performance evaluation of the proposed DB3KD approach on high-resolution, fine-grained dataset (Flowers-102). 62

4.3 Result of ZSDB3KD with MNIST and Fashion-MNIST. S: score-based teacher. D: decision-based teacher. 68

4.4 Result of ZSDB3KD on AlexNet with CIFAR-10. 69

5.1 Volume ratios after asymmetric cell division during *C. elegans* early embryogenesis. 82

5.2 Cells involved in the four Notch signaling pathways in *C. elegans embryogenesis*. 86
5.3 Cell-cell squeeze direction of ABala and ABalp with their neighbors at AB8 stage. 89
5.4 Cell neighbor pairs with a high frequency of errors (90% to 100%) in FP and FN results. 93

A.1 Architecture of the CNN and the policy network in HDRL. 187

List of Figures

3.1 AlexNet on CIFAR-10. The shaded regions indicate one standard deviation. 21

3.2 VGG16 on Birds200. The shaded regions indicate one standard deviation. . 23

3.3 Overall workflow of the proposed approach. The number below each graph refers to the measurement of redundancy, which are just used for the illustration purpose and do not reflect the real measurements of the graphs. . 27

3.4 An example of the calculation of ℓ-covering number and quotient space size. Darker color nodes indicate the selected ball centers and lighter color nodes represent the elements within the corresponding balls. 30

3.5 Progressive pruning results of AlexNet on CIFAR-10. 37

3.6 Progressive pruning results of of VGG-16 on Birds-200. 39

3.7 The value of n_1 and n_2 by changing γ from 0.001 to 0.3. Black solid line refers to $n_1 = n_2$. 40

3.8 The network structure comparison when 40% filters are pruned from AlexNet using different γs. 44

4.1 The overall workflow of the proposed approach. Left: classic KD. Bottom: decision-based black-box KD (DB3KD). Samples are iteratively fed to the DB3 teacher to compute the sample robustness, which is transformed as soft labels for training the student via KD. Right: Zero-shot DB3KD (ZSDB3KD). Pseudo samples are generated by moving random noises away from the decision boundary and approaching the distribution of the original training samples, which are used as the transfer set for training the student via DB3KD. 52
4.2 Strategies for computing sample robustness. 54
4.3 The iterative procedure for the optimization of MBD. 56
4.4 Performance comparison with different numbers of queries for computing sample robustness. 64
4.5 (a) The average minimal boundary distances over number of queries. Error bar indicates one standard deviation. (b-d) Normalized average minimal boundary distances of the samples of different classes. Darker colors indicate smaller distances between two classes. 65
4.6 Analysis and ablation study of ZSDB3KD with MNIST. Left: evolution of pseudo images over iterations. Middle: averaged images compared to other white-box zero-shot KD approaches. Upper right: the accuracies with different iterations of sample generation. Bottom right: the accuracies with different numbers of samples used for training the student. 70

5.1 Voronoi diagrams. 76
5.2 The framework of the proposed cell neighbor determination model. 78
5.3 Running time comparison from 4 to 350-cell stage. 83
5.4 Accuracy comparison from 4 to 350-cell stage. 85
5.5 The neighbor relationship from the Voronoi, kNN and SVM approaches during the 3rd and 4th Notch signaling pathways of *C. elegans* embryogenesis. 88

5.6 Distribution of the FP and FN results during embryogenesis simulation. . . 91
5.7 The number and frequency of errors of all cell pairs in FP and FN results. . 92
5.8 Top 30 cells with the most errors in FP and FN results, and analysis from AceTree. 95

6.1 The ABM framework. Cells move at each time step based on reading the observed locations (dumb cells) or the output of the neural network (intelligent cell). After a cell's movement, if it is at the right time for division, a new cell is hatched. Such a process repeats until the end of the simulation. 103
6.2 The reinforcement learning framework. A cell interacts with the embryo. At each time step, the cell receives a state S_t, selects an action A_t, gets a reward R_t and enters the next state S_{t+1}. The cell's objective is to maximize the total rewards received. 105
6.3 The deep Q-network framework for cell movement, which cotnains a cell migration loop and a network learning loop. The intelligent cell's movement is selected via the ϵ-greedy mechanism, from either a random sampling of all the possible actions or the output of the neural network. Then it gets a reward, moves to the next location, and repeats this process. The samples generated from the cell migration loop are used to update the parameters of the neural network via backpropagation. Experience replay and target network are implemented to improve the performance. 106
6.4 An example of a specific evaluation step for a single action. A list of cells are pre-selected as the state cells via the cell neighbor determination model. Their locations are concatenated and sent to the neural network, and the output action with the maximal probability is selected as the intelligent cell's next movement. 111

6.5 Comparison between (a) the 3D time-lapse images and (b) the visualizations of the ABM simulation results. Simulation results highly reproduce the observed patterns. 115
6.6 Performance evaluation of the deep reinforcement learning algorithm for cell movement modeling. (a) The accumulated rewards generally goes upward, but tends to be noisy. (b) The loss tends to oscillate because of the implementation of the experience replay and the target network. (c) The average action value grew smoothly over time. 116
6.7 Results of the *Cpaaa intercalation* case. (a) Observation results visualized by Acetree from 3D time-lapse images. (b) Simulation results of the intercalating cell Cpaaa with the *Destination* rule. (c) Simulation results when training Cpaaa only with the *Boundary* and *Collision* rules, without the *Destination* rule, which indicate that Cpaaa fell into a suboptimal location. (d) Simulation results of the cell Caaaa, a neighbor of Cpaaa. Red, yellow, and green circles represent the intelligent cell, input state cells, and non-related cells, respectively. The white circle indicates the destination of the intelligent cell. All four sets of data were collected at the following time steps: 0, 4, 8, 12, 17, and 22 (minutes from the beginning of the simulation). 118
6.8 (a) Migration paths of Cpaaa with directional movement. (b) Simulation results when training Cpaaa only with the *Boundary* and *Collision* rules, without the *Destination* rule. Results indicate that Cpaaa fell into a suboptimal location. Both simulation paths are the averages over 50 runs, and the shaded regions indicate ranges of one standard deviation greater/less than the average values. The horizontal axis represents the developmental time in minutes. The vertical axis represents the projected position of Cpaaa on the AP-axis to the center of the embryo. 120

6.9 The simulation of left-right asymmetry rearrangement. (a) Observation data. The intelligent cell and the leading cell are circled. (b) Simulation results. The cyan circle represents the leading cell, and the others are color coded, as in Fig. 6.7. The white circle here indicates the destination of the intelligent cell only for the purpose of visualization. Both sets of data were collected at the following time steps: 0, 3, 6, and 9 (minutes from the beginning of the simulation). 122

7.1 Concepts and design to model cell movement with HDRL. (a) A schematic showing actors in cell movement modeling. Small circles represent cell nuclei in a tissue. The migrating cell, subgoals, and other cells are colored red, green, and white, respectively. Dashed circle indicates the current neighborhood of the migrating cell. The migrating cell moves along the arrows towards the destination marked with a cyan star. (b) Architecture and major components of the two-level HDRL used. Arrows indicate data flow between components. Green bounding box indicates the input of the CNNs (feature extraction component). Black bounding boxes indicate model components. Model inputs and outputs are shown as text/symbols without bounding boxes (red indicates output). White and blue CNNs indicate separate networks with their own parameters. (c) Architecture of the TMM. The blue CNN indicates that it is transferred from the CNN of the lower-level module in the HDRL framework in (b). 130

7.2 Modeling Cpaaa migration in *C. elegans* embryogenesis.(a) Micrographs of *C. elegans* embryo showing the migration of Cpaaa. Dorsal view, anterior to the left (A, anterior; P, posterior). Nuclei in green. Star indicates the ABarpaapp cell. Time 0 is the birth of Cpaaa. (b) The annotated image at a time point during HDRL training. Red, yellow, cyan and green indicate the migrating cell, subgoals, the destination and other cells in the embryo, respectively. (c) The rewards generated over training epochs with different rule settings. "HDRL" indicates HDRL training with the global feedback, subgoals, and local feedback. "No subgoals" indicates DRL training with the global feedback and the local feedback but no subgoals. "Local feedback" indicates training with only the local feedback. The earned rewards were averaged based on five runs and the shaded regions indicate one standard deviation. (d) Position of the migrating cell over time after training with different rule settings. "Observational data" indicates Cpaaa migration in image series. Lines represent the average of each group (20 runs for each rule setting and 10 wild-type embryos for observational data) and shaded regions indicate one standard deviation. Time 0 is 15 minutes after the birth of Cpaaa. 133

7.3 Modular organization of Cpaaa movement in HDRL and 3D time-lapse imaging.(a) The MI curve of successful migration scenarios. The average MI (black line) with one standard deviation (shaded region) are shown. Green and blue lines indicate the timing of achieving the first and second subgoal, respectively, where the vertical lines indicate the average time and horizontal line one standard deviation. Red lines indicate the timing of rosette formation identified from imaging experiments shown in (c). Time 0 is 15 minutes after the birth of Cpaaa. (b) The collection of all the subgoals in each of the directional movement phases. The group of subgoals for the first phase includes n1-n4 (green) and the group for the second phase n3-n6 (blue). Red circle, Cpaaa; cyan star, ABarpaapp. (c) Micrographs of *C. elegans* embryo showing sequential rosettes during Cpaaa migration. Dorsal view, anterior to the left (A, anterior; P, posterior). Nuclei in green and cell membranes in red. Time 0 is 15 minute after the birth of Cpaaa. Dashed lines encircle the eight cells involved. Arrows indicate the centers of three rosettes. Star: ABarpaapp; n1: ABarppapp; n2: ABarppppa; n3: ABarppapa; n4: ABarpppap; n5: ABarppaap; n6: ABarpppaa. 135

7.4 Migration of Cpaaa upon genetic perturbations.Micrographs of *C. elegans* embryo. See Fig. 7.2(a) for convention. White star marks the destination in the wild type (ABarpaapp). White dashed circles mark the ABarp cells involved. For n1-6, see Fig. 7.3. Red star, dashed circles and n' mark cells in the ABprp lineage that have adopted the corresponding ABarp fates. Time 0 is 15 minutes after the birth of Cpaaa. (a) A pal-1 (RNAi) embryo. (b) A wwp-1 (RNAi) embryo. (c) A glp-1 (RNAi) embryo. Red star: ABprpaapp; n'1: ABprppapp, n'3: ABprppapa, n'5: ABprppaap,. Note that the wild-type destination (ABarpaapp) is on the dorsal side of the plane shown and the white star indicates the projection of ABarpaapp position on the plane shown. n1/3/5 in the ABarp lineage are also on the dorsal side and not shown. 139

7.5 Validation and characterization of the TMM.(a) Test accuracy of the TMM and the Motion Model (MM). (b) The MI curve from the TMM in four embryos that were not included in the training set. (c) Example input images and the corresponding summary feature maps of the TMM at an early and a late time point of Cpaaa migration in three embryos. Input images on the left and feature maps on the right. For images, red shows nucleus of Cpaaa and green other cells. For feature maps, colors represent the value at each pixel of the summary map. Warmer color represents higher value. (d) Isocontour representation of the aggregated value of feature maps across embryos and time points. Closed blue circle marks the position of the migrating cell. Red asterisk marks the isocontour. The bottom inset shows a schematic of the migrating cell (shaded blue), current rosette neighbors (white cells in blue circle) as well as the other neighbors of the previous (yellow) and the next (red) rosette. (e) Upper panels: example ablated input image from Embryo 1 after ablating Cpaaa or Cpaaa's neighbors. White open circles marked the positions of the ablated cells. Lower panels: the corresponding summary difference map that shows changes in the summary feature map. (f) The ratio of overlapped area between the effective areas and the ablated cells to the effective area among a total of 74 timesteps in three embryos. See also Fig. A.4 for underlying data. 141

7.6 TMM classification and 3D time-lapse imaging of mu_int_R and CANL migration. (a,b) Micrographs of *C. elegans* embryo showing the migration of mu_int_R (a) and CANL (b). See Fig. 7.2(a) for convention. Time 0 is the birth of mu_int_R and CANL, respectively. (c,d) The MI curve from the TMM over mu_int_R (c) and CANL (d) migration in two embryos. Time 0 is the birth of mu_int_R and CANL, respectively. Red lines indicate the timing of rosette formation identified from imaging experiments shown in (e). (e) Micrograph of *C. elegans* embryo showing sequential rosettes during mu_int_R (marked by "mu") migration. Dorsal view, anterior to the left. Nuclei in green and cell membranes in red. The dash lines show the contour of rosettes. The arrows indicate rosette centers. Time 0 is the birth of mu_int_R. 144

A.1 MI curves of the successful migration scenarios using the Motion Model. Green and blue dashed lines indicate the time when the first and second potential subgoals (ABarp cells) are reached. 188

A.2 Feature maps of Cpaaa migration at one of the early migration timesteps in Embryos 1 (a), 2 (b), and 3 (c). Yellow represents highest pixel value and black lowest. All the maps reveal the size and orientation of the corresponding embryo. The yellow spots in half of the feature maps (32-39 out of 64) indicate the location of the migrating cell. 189

A.3 Examples of individual difference maps at a single timestep. Each of the 64 difference maps is shown for an ablation experiment after ablating Cpaaa (a) and Cpaaa's neighbor cells (b). 190

A.4 The number of pixels in the effective area of the summary difference map at each timestep in three embryos. Bars colored with blue/red represent the number of pixels where the ablated cells overlapped/not overlapped with the effective area. 192

A.5 Distribution for normalized minimal distance between neighbor cells. Statistics compiled from a collection of 50 wild-type embryos. X-axis indicates the level of cell overlapping. $\alpha = 0.3$ and $\beta = 0.8$ indicate the completely acceptable/unacceptable overlapping values, respectively. . 193

Chapter 1

Introduction

1.1 Motivation and challenges

In recent years, deep convolutional neural networks (CNNs) have enjoyed tremendous successes and have been widely used in different computer vision tasks , such as object classification [1, 2, 3, 4] and detection [5, 6, 7], image synthesis [8, 9, 10, 11], semantic segmentation [12, 13, 14], and biological image-based disease diagnosis [15, 16]. Improvements over the state of the art in these areas are usually achieved by designing deeper and wider CNNs [1, 17, 18, 2, 19]. However, the computing cost in these CNNs is usually expensive, which restricts their usage on resource-constrained devices such as robotics, mobile devices, and drones [20, 21, 22, 23, 24, 25, 26, 27].

To reduce the computation and storage cost of a CNN while maintaining its performance, many network compression approaches have been proposed, such as matrix decomposition [28, 29, 30], quantization [31, 32, 33], knowledge distillation [34, 35, 36, 37, 38], and network pruning [39, 40, 41, 42, 43, 44, 45, 46, 47, 48]. Among all these approaches, network pruning and knowledge distillation have become the most popular approaches because of its appealing properties of straightforward implementation and large pruning ratio with little performance degradation.

Network pruning aims to delete some unimportant parameters from a pre-trained network and finetuning the remaining network to recover its performance. It can be divided into two categories, i.e., weight pruning and channel pruning. Weight pruning uses for deleting single parameters in the neural network, resulting in unstructured sparsity in the pruned models. Although it can usually achieve high pruning ratios with little performance loss, specialized hardware and software are needed to deal with the unstructured sparsity to achieve the acceleration, which makes it less flexible. Compared to weight pruning [39, 49], channel pruning is a more straightforward approach because it removes the entire filter/channel rather than specific weights so that it can be implemented without the need for specialized software and hardware [50]. In our study, we mainly focus on channel pruning.

Knowledge distillation is another popular scheme with which a compact student network is trained by mimicking the softmax output (class probabilities) of a pre-trained deeper and wider teacher model [34]. By doing so, the rich information learned by the powerful teacher can be imitated by the student, which often exhibits better performance than solely training the student with a cross-entropy loss. Many variants have been developed to improve the vanilla KD approach by not only mimicking the softmax output but also matching extra elements in the teacher, such as the output of intermediate convolutional layers [51], attention maps [52], percentage of activated neurons [53], etc.

Although network pruning and knowledge distillation have achieved great successes, there are still some issues that need to be solved.

Problems in existing network pruning approaches. Traditional network pruning procedure includes three steps. First of all, an over-parameterized neural network is trained in a normal way until it can not be further improved. Then the unimportant parameters in the network are identified and removed. Finally, the remaining network is fine-tuned for some iterations to recover the performance. Usually, the second and third steps are conducted iteratively until the pruning rate is reached. There are many criteria to define what kind of parameters are less important than others in previous studies [41, 42, 54, 55]. For example, [41] considers the parameters with small magnitudes are less important because they contribute less to the calculation. Molchanov et al. use the Taylor series to

estimate the loss change after each filter's removal and prune the filters that cause minimal training loss change [42]. It has been taken for granted that dropping the least important filters ranked with the optimal criterion results in the minimal performance loss [41, 56].

However, our studies on channel pruning contradict this common belief. Using statistical modeling to measure the redundancy in each convolutional layer, we will theoretically show that pruning filters in the layer with the most redundancy outperforms pruning the least important filters across all layers, which is also supported by the subsequent empirical studies.

Problems in existing knowledge distillation approaches. In tradition, to train a compact neural network via KD, the following factors must be provided. (1) The teacher's training samples are accessible. (2) The teacher model is a white-box teacher model, i.e., the teacher's parameters are accessible. (3) the score-based outputs, i.e., class probabilities of the training samples are provided by the teacher. However, these factors are usually unrealistic in real-world applications.

In some cases, pre-trained models are not accessible because they may be the core competitiveness of a company or researcher and its parameters are not released. A more challenging situation is none of the three prerequisites mentioned above is available. A pre-trained model stored remotely may only provide APIs for inference, the model and its training samples are blind to the users. Worse than that, these APIs usually return hard-labels for the samples, instead of the probabilities of all the classes.

In our studies, we will focus on how to deal with these challenging scenarios with KD.

Efficient deep learning in real-world applications. In real-world applications, efficient deep learning is highly required. With the development of cutting-edge equipment and infrastructure, tremendous data are generated. However, how to efficiently leverage these data for further analysis becomes an issue [57, 58]. In this dissertation, we use a biological case study, i.e., cell migration modeling in the developmental system to demonstrate how to utilize efficient deep learning tools to improve the training efficiency and help to discover new biological regulatory mechanisms in practical scenarios [59, 60].

1.2 Dissertation overview

With the above concerns, in this dissertation, I make several contributions to design efficient deep learning algorithms to (1) improve the performance in traditional deep neural network compression tasks and (2) develop efficient deep learning tools to help improve the training efficiency in real-world applications, using biological cell migration modeling as an example. The main contributions of this dissertation include: a new network pruning approach from the perspective of structural redundancy reduction (Chapter 3); a knowledge distillation approach in the scenario with limited prior information (Chapter 4); a cell migration modeling system using efficient deep reinforcement learning (Chapter 5,6,7).

The rest of the dissertation is organized as follows. Chapter 2 presents the background of the related fields of the work presented in this dissertation, including network pruning, knowledge distillation, reinforcement learning, and agent-based modeling for cell migration. Chapter 3 introduces the proposed channel pruning approach, which identifies the most redundant layer in a pre-trained deep neural network and pruning filters from that layer, rather than removing the least important filters across all layers as most of the previous studies did. Chapter 4 introduces a knowledge distillation approach with a much more challenging scenario, that is, the training set is not accessible, and the pre-trained teacher model is black-box, with only the top-1 category returned. Chapter 5 introduces a machine learning-based approach to determine whether two agents in a developmental system are neighbors with each other at a given time, which is a basis for the modeling of cell migration introduced in the following Chapters. Chapter 6 introduces a novel cell migration approach with reinforcement learning. Chapter 7 improves the training efficiency of the approach described in Chapter 6 in a more challenging but realistic scenario. Finally, Chapter 8 concludes the dissertation.

Chapter 2

Background and Related Works

2.1 Neural network compression

2.1.1 Overview

Training compact deep neural networks (DNNs) [61] efficiently has become an appealing topic because of the increasing demand for deploying DNNs on resource-limited devices such as mobile phones and drones [62]. Recently, a large number of approaches have been proposed for training lightweight DNNs with the help of a cumbersome, over-parameterized model, such as network pruning [41, 46], quantization [31], factorization [63], and knowledge distillation (KD) [34, 64, 65, 66, 67, 37]. Among all these approaches, network pruning and knowledge distillation are two popular schemes.

2.1.2 Network pruning

Early works and weight pruning. Network pruning is a long-standing topic that can be traced back to the 1990s [68, 69]. In the era of deep learning, [39] is one of the most famous early works that prunes weights below a threshold. After that, various weight pruning approaches have been proposed [70, 71, 49]. As mentioned before, weight pruning

causes unstructured sparsities in a network, which is difficult to be used without specialized software and hardware [50].

Channel pruning. Channel pruning [41, 42, 46, 72] removes the entire filters in a network so that there is no need for specialized hardware. Among all channel pruning approaches, identifying and pruning the least important filters is one of the most popular branches, and can be further divided into three categories. (1) Ranking and pruning filters with a certain criterion. [41] and [40] prune the filters with small weight magnitudes or activation values in the corresponding feature maps. [55] uses the average percentage of zero (APoZ) activation neurons as the criterion and deletes the filters with small ApoZ. First and second-order Taylor expansion are used to estimate the loss change after each filter's removal and the filters that cause minimal loss change are removed [42, 73]. HRank [74] leverages the information in the feature maps to rank the filters. (2) Reconstruction error minimization. Thinet [44] and NISP [45] prune the filters whose removal leads to minimal reconstruction error of the next layer. (3) Similarity measurement. These approaches use various strategies, such as geometric median [46] and clustering [75, 76], to identify the most replaceable filters, or those functionally share the most similarity with others.

Pruning with reinforcement learning Recently, there are several studies leveraging reinforcement learning (RL) in the network pruning area. N2N learning [77] proposes to use a recurrent neural network trained with RL for network compression. [78] and [79] use RL to prune connections in ResNet and DenseNet, respectively. AMC [80] introduces a deep deterministic policy gradient (DDPG) [81] agent to optimize the best number of filters in each convolutional layer and achieve considerable acceleration on mobile devices. A "try-and-learn" scheme is described in [82], which trains an agent that takes filter weights as the input and outputs a binary decision on whether the filters should be pruned. There are also a number of automatic pruning approaches related to this line of research [83, 84, 85, 86]. AutoPruner [83] uses the gradient information during fine-tuning to rank the filters so that pruning and fine-tuning can be combined. [85] uses Viterbi inference and [87] introduces to achieve such a purpose. Related to network pruning, there are also several

works in the neural architecture search (NAS) area using RL to obtain compact CNNs [88, 89, 90].

Pruning as network structure optimization Recently, a number of empirical studies indicate that the network structure after pruning, rather than the removal of unimportant filters, plays a decisive role in maintaining the performance of a network. [91] trained several compact networks obtained by pruning approaches but with random initialization. Surprisingly, comparable or even better performance can be achieved compared with fine-tuning the pruned models. [92] reports that a network's performance can be recovered even after random pruning. Related to these works, we also find that pruning unimportant filters is not always essential. But beyond that, we theoretically show that pruning in the layers with large redundancy outperforms pruning the least important filters and propose to prune a network based on structural redundancy reduction.

2.1.3 Knowledge distillation

Knowledge distillation. Knowledge distillation is first introduced in [93] and generalized in [94, 34], which is a popular network compression scheme to train a compact student network by mimicking the softmax output predicted by a high-capacity teacher or ensemble of models. Besides transferring the knowledge of class probabilities, many variants have been proposed to add extra regulations or alignments between the teacher and the student to improve the performance [51, 95, 96, 53]. For example, FitNet [51] introduces an extra loss term that matches the values of the intermediate hidden layers of the teacher and the student, which allows fast training of deeper student models. [52] defines the attention of DNNs and uses it as the additional transferred knowledge. In [95], inner product between features from the two layers are calculated as an extra alignment between the teacher and the student to improve the performance of knowledge distillation. As mentioned before, the success of these methods relies on access to the training samples and the fully exposed white-box teacher.

Knowledge distillation with limited data. To mitigate the storage and transmission costs of large training datasets, several studies propose the concept of few-shot KD, which generates pseudo samples with the help of a small number of the original training samples [97, 98, 99]. Another study suggests that instead of the raw data, some surrogates with much smaller sizes (also known as metadata) can be used to distill the knowledge from the teacher. [100] leverages the statistical features of the activations of the teacher to train a compact student without access to the original data. However, releasing this kind of metadata along with the pre-trained teacher is usually not a common scenario.

Zero-shot knowledge distillation. To deal with the scenario when training data is not accessible, [101] proposes zero-shot knowledge distillation (ZSKD). The authors model the softmax output space of the teacher with a Dirichlet distribution and samples soft labels as the targets. Randomly generated noise inputs are optimized towards these targets via backpropagation and are used as the transfer set. [37] replaces the Dirichlet distribution with a multivariate normal distribution to model the softmax output space of the generated samples. Therefore, pseudo samples of different classes can be generated simultaneously rather than one after another as in [101]. Generative adversarial networks (GANs) [8] are leveraged in [102, 103] to solve this task so that pseudo sample synthesis and student network training can be conducted simultaneously. Another study [104] proposes to use the features in the batch normalization layers to generate pseudo samples. The key idea of these approaches is to generate informative pseudo samples that can capture the distributions of the training set. However, these methods still need access to the parameters of the teacher for backpropagation, which is unrealistic in many cases.

Black-box knowledge distillation. Although the vanilla KD is built with a black-box teacher [34], the whole training dataset is used for training. [98] investigates the possibility that a student is trained with limited samples and a black-box teacher. Other than zero-shot KD methods that generate pseudo inputs, [105] proposes to sample from a large pool (such as ImageNet) to get the transfer set to train the student. Therefore, there is no need to access the teacher's parameters. Different from previous few-shot knowledge distillation methods, this study generates extra pseudo samples without access to the teacher's parameters.

Although the prerequisites in these methods are relaxed, weak assumptions on the training samples and a score-based teacher that outputs class probabilities are still needed. Different from these studies, we consider a much more challenging case in which knowledge is transferred from a black-box teacher that only returns top-1 classes.

Self-knowledge distillation. Similar to our proposed scenario, in the absence of a pre-trained teacher, self-knowledge distillation aims to improve the performance of the student by distilling the knowledge within the network itself [106]. Born-again neural networks [107] proposes to use a multi-round self-distillation mechanism so that the student network's performance can be similar to the teacher with an identical structure.

2.2 Reinforcement learning

Reinforcement learning has developed substantially in recent years [108, 81, 109, 110]. By leveraging CNNs, RL algorithms achieve superhuman performance in the long-standing classic tasks with large search spaces such as playing video games [111, 108], the game of Go [112, 113], and robotic control [114]. Because of these successes, RL has attracted a huge amount of attention in various areas to optimize decision-making problems. For example, [115] uses RL to optimize the traffic signal control problem. These implementations inspire us to use RL to model the cell migration process.

2.3 Cell migration modeling

Agent-based modeling (ABM) [116] is a powerful approach to study systems consisting of self- and environment-ruling, interacting agents [117, 118, 119, 120], and it is well suited for multi-scale analysis of complex tissues and development. A desired framework would include players at multiple scales, namely how genes interact to give rise to cellular behavior and how cells interact to give rise to an organism. An ABM framework for complex tissues typically includes three scales: molecules (genes), cells, and tissues, in which agents represent interacting individual cells. Behaviors of an agent

are controlled by inherited biological information: known actions of gene; intercellular signals, physical limitations, environment factors and external interference; or statistical models based on experimental measurements if the underlying molecular mechanism is not known. Emergent properties, such as coordinated generation of the cell types and tissue morphology, can be examined by simulating gene actions and cell interaction. Among the earlier ABM-related approaches, the Cellular Potts Model (CPM) [121, 122] and statecharts [123, 124, 125, 126] are two representative ones that are worth noting. These models effectively simulate certain kinds of cell behavior, such as morphology, tissue development, and organogenesis. Such models focus on examining known/prescribed mechanisms, and are not aimed at, or even capable of handling, large amounts of observational/phenomenological data from the live microscope and 3D time-lapse imaging. However, these models requires a comprehensive of the regulatory mechanisms behind the modeling process, which is often missing practically.

Part I

Neural Network Compression with Channel Pruning and Knowledge Distillation

Chapter 3

Network Pruning via Structural Redundancy Reduction

3.1 Overview

As introduced in the previous chapters, convolutional neural networks (CNNs) [127] have developed substantially in recent years and are widely used in various applications. However, the over-parameterization problem of CNNs prevents them from being applied to resource-limited devices, such as mobile phones and robotics [21, 20]. Many approaches have been proposed to reduce the computation and storage cost of CNNs, such as quantization [31], matrix decomposition [29], network pruning [39, 41, 128, 48, 129, 46, 130], and knowledge distillation [34]. Network pruning is one of the most popular methods and attracts enormous attention.

Many of the existing channel pruning approaches rely on finding and pruning the least important filters, or the filters that share the most similarities with others across all layers [41, 42, 46, 131]. For example, [42] uses the Taylor series to estimate the loss change after each filter's removal and prune the filters that cause minimal training loss change. It has been a common belief that with a better filter ranking criterion, there is a better chance to drop the least important filters and get a compact network with less performance loss.

However, our studies on channel pruning contradict this common belief [132]. Using statistical modeling to measure the redundancy in each convolutional layer, we theoretically show that (in certain cases, even randomly) pruning filters in the layer with the most redundancy outperforms pruning the least important filters across all layers. To our best knowledge, this is the first study that theoretically analyzes the rationale behind network pruning from a redundancy reduction perspective. With this finding, we propose a layer-adaptive channel pruning approach based on structural redundancy reduction (SRR), which is achieved by establishing a graph for each convolutional layer of a CNN and using two quantities associated with the graph, i.e., ℓ-covering number and quotient space size, as the measurement of the redundancy in each layer. After that, unimportant filters in the identified layer(s) with the most redundancy, rather than the least important filters across all layers, are pruned.

We summarize the contribution of this chapter as follows.

- We theoretically analyze network pruning with statistical modeling from a perspective of redundancy reduction. We find that pruning in the layer(s) with the most redundancy outperforms pruning the least important filters across all layers.

- We propose a layer-adaptive channel pruning approach based on structural redundancy reduction, which builds a graph for each convolutional layer of a CNN to measure the redundancy existed in each layer. This approach prunes unimportant filters in the most redundant layer(s), rather than the filters with the least importance across all layers.

- We validate the proposed approach on various network architectures and datasets. Experiment results demonstrate that our approach achieves state-of-the-art performance compared with recent channel pruning methods. More specifically, our pruned ResNet50 model on ImageNet can reduce 44.1% FLOPs while losing only 0.37% top-1 accuracy.

3.2 A theoretical analysis on network pruning: a statistical perspective

In this section, we first statistically formulate the channel pruning problem from a structural redundancy reduction perspective. We will theoretically prove that pruning in the layer with the most redundancy outperforms pruning the least important filter across all layers. Then we empirically use several experiments to validate that the claim still holds in real-world scenarios.

3.2.1 Rethinking network pruning from the perspective of structural redundancy reduction

To illustrate the importance of structural redundancy reduction, we consider the following the simplest definition of structural redundancy and the simplest setup. We oversimplify layer redundancy as the number of filters in a convolutional layer. (We will later show that the structural redundancy can be better measured with other quantities in real applications in Section 3.3.) Suppose we have a two-layer CNN* with m and n filters, where $n \gg m$. Let $\{\xi_1, \xi_2, \cdots, \xi_m\}$ and $\{\eta_1, \eta_2, \cdots, \eta_n\}$ be one dimensional positive random variables (RVs) representing each filter's contribution to the network performance. For example, a filter's contribution can be represented as the absolute value of *training accuracy drop* or *training loss change* after pruning that filter. We call the two layers ξ layer and η layer for convenience. We first highlight our finding and then prove it from a statistical modeling perspective.

Claim: If a layer has much higher redundancy, pruning filters in that layer, either randomly or selectively, outperforms pruning the least important filters across all layers.

We choose positive constants $a, b > 0$, and use the random events $(\sum_{i=1}^{m} \xi_i \geq a)$ and $(\sum_{i=1}^{n} \eta_i \geq b)$ to describe the layers ξ and η "performing well". Then the performance of a system (i.e., the whole neural network) p is measured by the sum of probabilities of the two

*This configuration can be extended to a multi-layer network (number of layers ≥ 3) with no difficulty.

events (see Eq. (3.1)). We define one system (p_1) to perform better than another (p_2) if $p_1 > p_2$. A natural question is, if we prune a filter from the network, i.e., remove one variable from $\{\xi_1, \xi_2, \cdots, \xi_m, \eta_1, \eta_2, \cdots, \eta_n\}$, how does the system performance change? There are the following cases (the performances of the systems are listed in Eqs. (3.1)-(3.5)): (1) no pruning; (2) randomly pruning a filter in the η layer, without loss of generality, we assume the last one η_n is pruned; (3) pruning the least important filter $\underline{\eta} = \min\{\eta_1, ..., \eta_n\}$ in the η layer; (4) pruning the least important filter $\underline{\xi} = \min\{\xi_1, ..., \xi_m\}$ in the ξ layer; and (5) pruning the globally least important filter, i.e., $\min\{\underline{\xi}, \underline{\eta}\}$.

$$p_o = P(\sum_{i=1}^{m} \xi_i \geq a) + P(\sum_{i=1}^{n} \eta_i \geq b) \tag{3.1}$$

$$p_{\eta r} = P(\sum_{i=1}^{m} \xi_i \geq a) + P(\sum_{i=1}^{n-1} \eta_i \geq b) \tag{3.2}$$

$$p_{\underline{\eta}} = P(\sum_{i=1}^{m} \xi_i \geq a) + P(\sum_{i=1}^{n} \eta_i - \underline{\eta} \geq b) \tag{3.3}$$

$$p_{\underline{\xi}} = P(\sum_{i=1}^{m} \xi_i - \underline{\xi} \geq a) + P(\sum_{i=1}^{n} \eta_i \geq b) \tag{3.4}$$

$$\begin{aligned} p_g &= \frac{m}{m+n}\left[P(\sum_{i=1}^{m} \xi_i - \underline{\xi} \geq a) + P(\sum_{i=1}^{n} \eta_i \geq b)\right] \\ &+ \frac{n}{m+n}\left[P(\sum_{i=1}^{m} \xi_i \geq a) + P(\sum_{i=1}^{n} \eta_i - \underline{\eta} \geq b)\right] \\ &= \frac{m}{m+n} p_{\underline{\xi}} + \frac{n}{m+n} p_{\underline{\eta}} \end{aligned} \tag{3.5}$$

It is worth mentioning that we consider the network performance from a perspective of redundancy (or capacity). That is why we do not divide the probabilities in Eqs. (3.1)-(3.5) by m or n. a or b can be considered as a threshold. As long as the total contribution of the filters in a layer is greater than the threshold, there is no performance loss. If a layer has too much redundancy (too many filters in our context), then it's very likely that the total contribution of the filters can still be greater than the threshold after pruning some of them.

Note that $0 \leq \eta_n - \underline{\eta} \leq \eta_n$, we have

$$P(\sum_{i=1}^{n-1} \eta_i \geq b) \leq P(\sum_{i=1}^{n} \eta_i - \underline{\eta} \geq b) \leq P(\sum_{i=1}^{n} \eta_i \geq b), \tag{3.6}$$

which indicates $p_{\eta r} \leq p_{\underline{\eta}} \leq p_o$.

For the filters in the η layer, we naturally assume that the contribution of a filter to the network's performance cannot be infinite, i.e., the variances of filters' contributions are uniformly bounded (Eq. (3.7)).

$$\exists C_1 > 0, \quad \text{s.t.} \quad \mathbb{D}\eta_i \leq C_1, i = 1, 2, \cdots, n. \tag{3.7}$$

By Chebyshev's inequality, for any real number $\epsilon > 0$,

$$P(\frac{1}{n}|\sum_{i=1}^{n}(\eta_i - \mathbb{E}\eta_i)| \geq \epsilon) \leq \frac{\mathbb{D}(\sum_{i=1}^{n}\eta_i)}{\epsilon^2 n^2}. \tag{3.8}$$

With Eq. (3.7), it is obvious that we have $\mathrm{Cov}(\eta_i, \eta_j) \leq \sqrt{\mathbb{D}\eta_i \cdot \mathbb{D}\eta_j} \leq C_1$.

We further define that there are $C_2 n$ $(0 \leq C_2 \leq 1)$ pairs of correlated filters in the η layer, i.e., $\#\{(i,j) : \mathrm{Cov}(\eta_i, \eta_j) \neq 0, i \neq j,\ i, j = 1, \cdots, n.\} \leq C_2 n$. Then we have,

$$\begin{aligned}\mathbb{D}(\sum_{i=1}^{n}\eta_i) &= \sum_{i=1}^{n}\mathbb{D}\eta_i + \sum_{i\neq j}\mathrm{Cov}(\eta_i, \eta_j) \\ &\leq C_1 n + C_1 C_2 n = C_1(1 + C_2)n.\end{aligned}$$

By Eq. (3.8),

$$P(\frac{1}{n}|\sum_{i=1}^{n}(\eta_i - \mathbb{E}\eta_i)| \geq \epsilon) \leq \frac{C_1(1+C_2)}{\epsilon^2 n} \to 0.$$

This means $\frac{1}{n}\sum_{i=1}^{n}(\eta_i - \mathbb{E}\eta_i)$ converges in probability to zero, i.e., $\frac{1}{n}\sum_{i=1}^{n}(\eta_i - \mathbb{E}\eta_i) \xrightarrow{P} 0$. Suppose the number of filters in the η layer n is large enough, say $n > \frac{2b}{\epsilon_0}$.

We consider that a filter's contribution needs to be positive, but it could be infinitely small, i.e., the expectation of filters' contributions have a uniform positive lower bound

(Eq. (3.9)).

$$\exists \epsilon_0 > 0, \quad \text{s.t.} \quad \mathbb{E}\eta_i \geq \epsilon_0, i = 1, 2, \cdots, n. \tag{3.9}$$

With Eq. (3.9), we have,

$$\begin{aligned}
&P(\frac{1}{n}\sum_{i=1}^{n}(\eta_i - \mathbb{E}\eta_i) > -\frac{\epsilon_0}{2}) = P(\sum_{i=1}^{n}\eta_i > \sum_{i=1}^{n}\mathbb{E}\eta_i - \frac{\epsilon_0}{2}n) \\
&= P\Big(\sum_{i=1}^{n}\eta_i > \frac{\epsilon_0}{2}n + \sum_{i=1}^{n}(\mathbb{E}\eta_i - \epsilon_0)\Big) \\
&\leq P(\sum_{i=1}^{n}\eta_i > \frac{\epsilon_0}{2}n) \leq P(\sum_{i=1}^{n}\eta_i > b).
\end{aligned}$$

Letting $n \to +\infty$, taking the limit and note that $\frac{1}{n}\sum_{i=1}^{n}(\eta_i - \mathbb{E}\eta_i) \xrightarrow{P} 0$, we have

$$\lim_{n\to\infty} P(\sum_{i=1}^{n}\eta_i > b) \geq \lim_{n\to\infty} P(\frac{1}{n}\sum_{i=1}^{n}(\eta_i - \mathbb{E}\eta_i) > -\frac{\epsilon_0}{2}) = 1,$$

Similarly,

$$\lim_{n\to\infty} P(\sum_{i=1}^{n}\eta_i - \eta_r > b) = \lim_{n\to\infty} P(\sum_{i=1}^{n}\eta_i - \underline{\eta} > b) = 1,$$

and then we have $p_{\eta r} \approx p_{\underline{\eta}} \approx p_o$ for n large enough. Note that $p_{\underline{\xi}} \leq p_o \approx p_{\underline{\eta}}$ and observe that p_g is the weighted average of $p_{\underline{\xi}}$ and $p_{\underline{\eta}}$. Hence we have $p_{\underline{\xi}} \leq p_g \leq p_{\underline{\eta}}$. It is worth mentioning that we cannot imply $p_g \approx p_\eta$ from Eq. (3.5) by letting $n \to \infty$ because we do not assume $m/n \to 0$.

In summary, we have $p_{\underline{\xi}} \leq p_g \leq p_{\eta r} \leq p_{\underline{\eta}} \leq p_o$, which indicates that (even randomly) pruning a filter in the layer with much larger redundancy outperforms pruning the least important filter across all layers. Here we consider the CNN as a black-box and we do not assume any prior distribution for the RVs to achieve a good generalization. So the conclusion holds no matter how the RVs are distributed. Indeed, the conclusion relies on the assumption $n \to +\infty$. However, the assumption can be relaxed in real world applications such that $p_{\underline{\eta}} \geq p_g$ still holds on average (though not in every filter selection step). In the next subsection, we present some intuitive examples on a number of networks,

which empirically provides evidence to validate the analysis above. As we can see in the following, for more sophisticated architectures that contain less redundancy, such as ResNet, with a well designed metric to measure the layer redundancy, our proposed approach that prunes the least important filters in the layer(s) with larger redundancy outperforms recent pruning approaches that removes the least important filters across all layers.

3.2.2 Empirical study of the theoretic analysis

In this section, we provide a number of intuitive examples to support our theoretical analysis in Section 3.2.1. For this purpose, we first manually create a redundant layer in several benchmark structures by increasing the number of filters in certain layers. Following the hypothesis in Section 3.2.1, here we simply use the number of filters in a layer as the redundancy measurement. Therefore, our naive structural redundancy reduction pruning approach just removes filters from the layers with the most number of filters in the neural network. In the following, we describe our experimental settings and compare the performance of this straightforward approach with several popular channel pruning methods.

Experiment configurations. We employ AlexNet and VGG16 as the backbones. We and enlarge the width of one of the last four convolutional layers for each architecture because deeper layers are usually considered as more redundant [42]. Specifically, we increase number of filters for each layer by a factor of four, except for the fourth layer from the last for AlexNet (i.e., the second convolutional layer), in which we increase number by a factor of six because the initial number of filters of this layer in the vanilla AlexNet is relatively small. We prune the enlarged AlexNets and VGG16s with the CIFAR-10 and Birds200 datasets, respectively. The architectures employed for the experiments are summarized in Table 3.1.

We train each enlarged network for 100 epochs with an SGD optimizer, using a learning rate started from 0.001. If the test accuracy does not increase for ten consecutive epochs,

Table 3.1: The manually augmented network architectures used for the experiments.

AlexNet on CIFAR-10					
Architecture	L1	L2	L3	L4	L5
Vanilla AlexNet	64	192	384	256	256
AlexNet Aug2		1152	384	256	256
AlexNet Aug3		192	1536	256	256
AlexNet Aug4		192	384	1024	256
AlexNet Aug5		192	384	256	1024

VGG16 on Birds200					
Architecture	L1-9	L10	L11	L12	L13
Vanilla VGG16	L1-2: 64x2 L3-4: 128x2 L5-7: 256x3 L8-9: 512x2	512	512	512	512
VGG16 Aug10		2048	512	512	512
VGG16 Aug11		512	2048	512	512
VGG16 Aug12		512	512	2048	512
VGG16 Aug13		512	512	512	2048

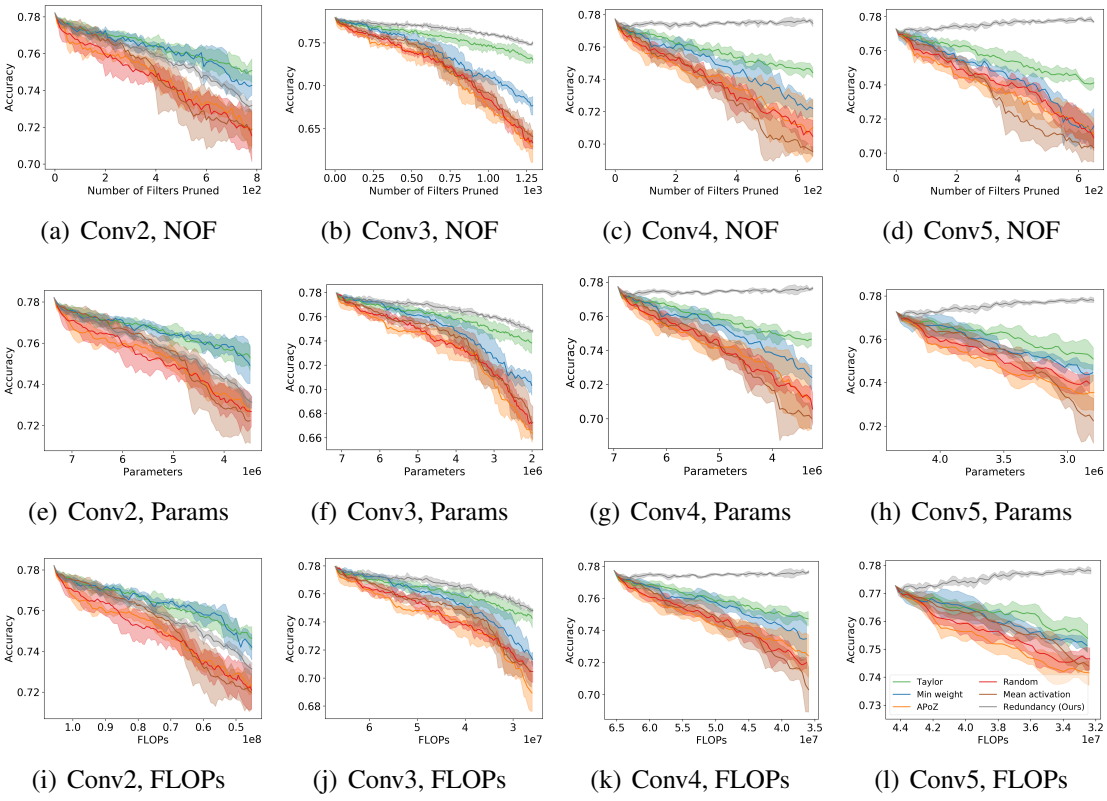
we decay the learning rate by a factor of 2. For enlarged AlexNets, each network is trained from scratch. For enlarged VGGs, we first train a vanilla VGG16 on the Birds200 dataset via transfer learning, building upon the pretrained model on ImageNet. After that, we use its weights as the initialization of each enlarged network, except for the enlarged layers, in which the weights are only copied to the first N_i filters (where N_i is the number of the filters in the i-th layer), leaving the rest random initialized. Except for the enlarged layer, we fix the rest of the weights and train the network for the first 25 epochs. Then we allow the loss to backpropagate across all the layers for the following training process.

We use progressive pruning for demonstration so that the performance of each approaches can be continuously observed. We compare the performance of our approach based on naive structural redundancy reduction with several popular channel pruning methods under the same configuration. Specifically, for the AlexNets, we prune 10 filters each time, which is followed by a 100 batch update of the slimmed network (with a batch size of 64) for the finetuning purpose. For the VGGs, we prune 50 filters each time and finetune the remaining network for 500 batches with a batch size of 32. The pruning & finetuning procedure repeats until the number of filters in the enlarged layer is less than any of the other layers in the network. All the finetuning are conducted with a learning rate of 0.0001.

We use two pruning strategies with our approach: (1) randomly pruning filters from the layer with the most number of filters, and (2) ranking the filters from the layer with the most number of filters with the Taylor expansion approach [42] and pruning the lowest ranked (recognized as the least important) filters. We compare our approach with a number of popular pruning approaches, including minimum weight [41], mean activation [40], average percentage of zeros (APoZ) [55], and Taylor expansion [42]. We also present the results of random pruning (across all layers) for better illustration.

Performance comparison.

Enlarged AlexNets on CIFAR-10. The experiment results of the four enlarged AlexNet networks trained/pruned with the CIFAR-10 dataset is presented in Fig. 3.1. It is observed that our approach achieves competitive performance against other methods on



(a) Conv2, NOF (b) Conv3, NOF (c) Conv4, NOF (d) Conv5, NOF

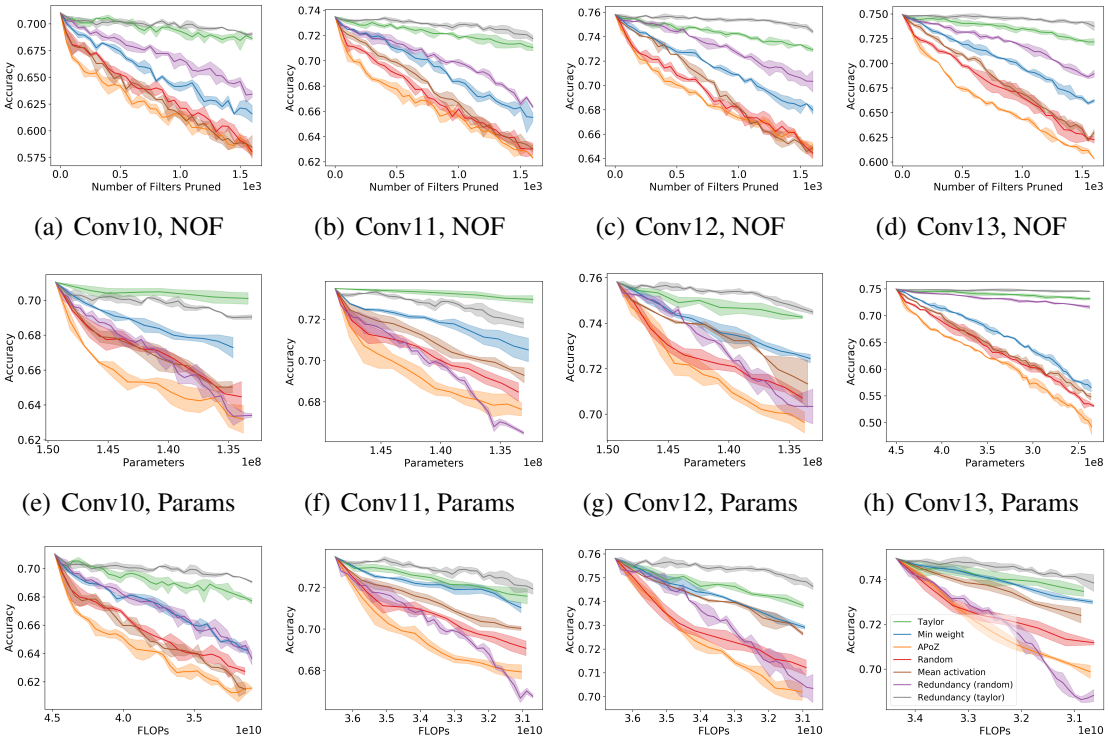
(e) Conv2, Params (f) Conv3, Params (g) Conv4, Params (h) Conv5, Params

(i) Conv2, FLOPs (j) Conv3, FLOPs (k) Conv4, FLOPs (l) Conv5, FLOPs

Figure 3.1: AlexNet on CIFAR-10. The shaded regions indicate one standard deviation.

the enlarged AlexNets, especially with the networks in which the third, fourth, and fifth layers are enlarged, respectively. Since our approach just randomly remove filters from the layer with the most number of filters rather than designing sophisticated mechanism to find unimportant filters across all layers in other compared approaches, these results are surprising and contradict common beliefs. Specifically, when pruning from the last convolutional layer, the test accuracies of the network, as the number of filters in this layer decreases from 1024 to 374, even increases from 77.27% to 77.69%, 77.81%, and 77.80% in terms of number of filters, parameters, and FLOPs, respectively, which is around 0.5% performance gain. For other approaches, the best test accuracy achieved by Taylor expansion is % after pruning, which is approximately 2% to 3% performance loss after pruning the same number of filters, parameters, and FLOPs. In the fourth and third convolutional layers, our approach also performs the best test accuracy among all approaches, even though the performance gain against others decreases as the pruning is conducted to shallower layers. It is observed that in the fourth convolutional layer, the network performance after pruning still increases with our approaches. However, all the approaches begin to lose performance as pruning filters in the third convolutional layer. In the second convolutional layer, our approach is outperformed by the Taylor expansion and minimum weight strategies, but is still competitive compared to APoZ, mean activation and random pruning. These results indicate that deep layers are usually with more redundancy given the number of filters within a similar range. More effective measurements on structural redundancy is needed since in this case we only use a very naive strategy, i.e., the number of filters in a convolutional layer, to measure the redundancy in each layer.

Enlarged VGGs on Birds 200. We also conducted experiments with several enlarged VGGs on the Birds200 dataset. The results are shown in Fig. 3.2. Similar to the AlexNet experiments, our approach achieves superior performance compared to other methods, with the simple strategy that prunes filters from the widest layer. In this series of experiments, however, we discover that the performance gap between our approach and other methods is less than those with AlexNet, indicating that VGG16 is a more effectively designed network



(a) Conv10, NOF (b) Conv11, NOF (c) Conv12, NOF (d) Conv13, NOF

(e) Conv10, Params (f) Conv11, Params (g) Conv12, Params (h) Conv13, Params

(i) Conv10, FLOPs (j) Conv11, FLOPs (k) Conv12, FLOPs (l) Conv13, FLOPs

Figure 3.2: VGG16 on Birds200. The shaded regions indicate one standard deviation.

with a better redundancy balancing among layers, even after manually enlarging a specific layer.

Specifically, we observe that randomly pruning from the layer with the most number of filters is no longer the best solution in these cases. But it is still worth mentioning that this strategy is still proved effective for layer redundancy identification since it performs much better than randomly pruning filters arbitrarily in the network. However, when we rank the filters' importance via the Taylor expansion approach in the most redundant layer and prune the least important filters in the single layer, its performance outperforms other popular approaches that prune least important filters across all layers in terms all three criteria (number of filters, parameters, and FLOPs). Taking the second last convolutional layer (Conv12) as an example, when the number of filters in this layer decreases from 2048 to 508, the performance loss is only 1% with our approach. On the other hand, the performance degradation of the Taylor expansion approach is around 2% after pruning the same amount of filters/parameters/FLOPs. For other approaches such as APoZ and mean activation, the test accuracies drop severely to around 66% after the pruning and finetuning, which is around 10% loss compared to the network's performance before pruning. These results validate the theoretic analysis in Section 3.2.1.

We also find similar phenomena of performance degradation as in the AlexNet experiments for the pruning results in the shallower layers. For example, when pruning in the ninth and tenth convolutional layers, our approach performs slightly worth than the Taylor expansion approach in terms of the number of parameters. This might be resulted from the fact that the filters in different layers contain different number of parameters and this factor is not considered for this calculation of structural redundancy in our naive pruning strategy.

In summary, we verify the effectiveness of the redundancy-based pruning approach with a simple strategy and several enlarged architectures in which redundancy are manually created in certain layers. The experiment results thus validate our theoretical claim that if a layer has much higher redundancy than others, pruning filters in that layer outperforms

pruning the least important filters across all layers. Beyond that, these experiments also indicate that more sophisticated strategy is need for better structural redundancy measurement, especially dealing with more effectively designed modern neural networks, which is usually with less structural redundancy, such as ResNet. In the next Section, we will propose a more systematic approach to measure the redundancy by establishing a graph for each convolutional layer according to the filters' weights and discover the redundancy in each graph as a representation of the level of layer redundancy.

3.3 Methodology

In this section, we present our proposed channel pruning approach. We first introduce the notations, preliminaries of network pruning and graph-related concepts used in our approach. Then we describe the proposed approach based on structural redundancy reduction.

3.3.1 Notations and preliminaries

Network pruning. Suppose a CNN has L layers. For the i-th layer, the number of the input and output channels are represented by N_i and N_{i+1}. Therefore, the CNN's parameters $\mathbf{W}$ can be represented as $\{\mathbf{W}^{(i)} \in \mathbb{R}^{N_{i+1} \times N_i \times h_i \times w_i},\ i = 1, 2, \cdots, L\}$, where h_i and w_i are the filter's width and height. Channel pruning is formulated to find a set of parameters $\mathbf{W}'$ optimized on certain objective functions such that $||\mathbf{W}'||_0 < K$, where $|| \cdot ||_0$ denotes the ℓ_0 norm and K limits the number of non-zero filters in $\mathbf{W}'$. Depending on different configurations, the objective functions can be minimizing a CNN's cost function, drop of training accuracy, or the reconstruction error, etc.

Graph theory. Let X be a finite set. An undirected graph is a pair (X, E), where E is a symmetric subset of $X \times X \setminus \{(x, x) : \ x \in X\}$. We call $x \in X$ a vertex (or a node) and $(x, y) \in E$ an edge. For $x, y \in X$, a path from x to y is a finite sequence $\{x_0, x_1, \cdots, x_n\} \subset X$ such that $x_0 = x,\ x_n = y$ and $(x_i, x_{i+1}) \in E$. In general, the above

path may not be unique if such a path exists. Denote $d(x, y)$ the minimal length of paths from x to y if x and y can be connected by a path; $d(x, y) = 0$ if $x = y$; and $d(x, y) = +\infty$ if x and y cannot be connected by a path. Then it is clear that $d(x, y)$ is an integer value metric. Recall that the degree of a vertex $x \in X$ is the total number of edges connected to x, i.e., $\deg(x) = \#\{(x, y) : (x, y) \in E\}$ ($\#A$ is the total number of elements in A).

3.3.2 Overall framework

We showed that pruning filters in the layer with larger redundancy outperforms pruning the least important filters across all layers. Our approach focuses on measuring how much redundancy exists in each layer and pruning filters from the most redundant layer(s) (Fig. 3.3). To measure the structural redundancy in a network, for each layer, we build an undirected graph in which each vertex represents a filter and the edges are defined with the distances between filter weights. We use two quantities associated with the graph, i.e., quotient space size and ℓ-covering number, as a measurement of how much redundancy exists in each graph, which is considered as the redundancy exists in each layer. At each time step, after the graph establishment and redundancy quantification, we randomly remove a vertex and its associated edges from the graph identified as with the most redundancy. Then we recalculate the redundancy after graph reconstruction for the next iteration. This process continues until a target is reached (e.g., a certain number of filters are pruned). Finally, we prune the filters in each layer according to the remaining number of vertices in each graph with a certain filter selection criterion. Note that in the filter pruning phase, filters are ranked separately in each layer, rather than globally across all layers. Since the redundancy identification phase has selected a different number of filters in each layer, our approach is a layer-adaptive approach.

We present the details of our approach as follows: graph establishment, calculation of quotient space size and ℓ-covering number, intuition and quantification of graph redundancy, and filter selection.

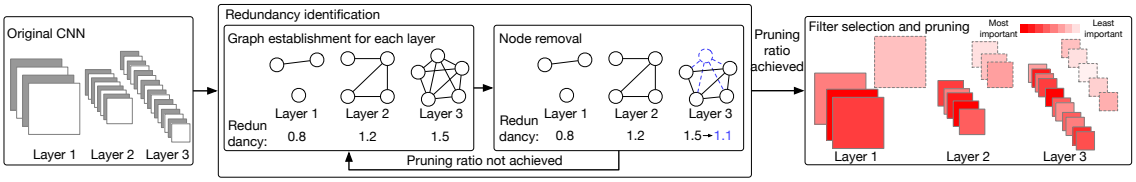


Figure 3.3: Overall workflow of the proposed approach. The number below each graph refers to the measurement of redundancy, which are just used for the illustration purpose and do not reflect the real measurements of the graphs.

3.3.3 Graph establishment

To illustrate how to build a graph for a convolutional layer, we use X to represent the filter weights of a certain layer $\mathbf{W}^{(i)}$ for simplicity. We first flatten and normalize the filter weights, which changes their lengths to 1. After that X becomes a finite subset of n dimensional unit sphere $\mathbb{S}^n = \{x \in \mathbb{R}^n : |x| = 1\}$ in $\mathbb{R}^n$, where $n = N_i \times h_i \times w_i$ and $|x|$ is the length of x in $\mathbb{R}^n$. We define a graph on X as follows (assuming the elements in X are distinct). We choose a positive real number $\gamma > 0$ and define an edge set on X as

$$E = \{(x, y) \in X \times X \setminus \Delta : |x - y|/\sqrt{n} \leq \gamma\},$$

where $\Delta = \{(x, x) : x \in X\}$ is the diagonal of X, and $|x - y|$ is the Euclidean distance on $\mathbb{R}^n$. Then we get a graph (X, E). By definition $(x, y) \in E$ implies x and y are approximately equal if γ is small.

3.3.4 ℓ-covering number

Recall that (X, d) is a metric space, where d is the graph metric defined previously. Let $\ell > 0$ be a fixed natural number, a subset $X_0 \subset X$ is called an ℓ-cover set of X, if $X \subset \bigcup\{B(x', \ell) : x' \in X_0\}$, where $B(x', \ell) = \{x \in X : d(x', x) \leq \ell\}$ is the ball centered at x' with radius ℓ. This means X is covered by the balls $\{B(x', \ell) : x' \in X_0\}$. We call the following quantity the ℓ-covering number of X:

$$N_\ell^c = (N_\ell^c(X) =) \min\{\#X_0 : X_0 \text{ is an } \ell\text{-cover set of } X\}.$$

3.3.5 Decomposition of a graph

We call a graph connected if for any $x \neq y$, there exists a path from x to y. In this case $d(x, y) < \infty$ for all $x, y \in X$. For an unconnected graph (X, E), we define the notation "$\sim$" on X as follows: $x \sim y$ if and only if there exists a path from x to y. Then it is clear that "$\sim$" is an equivalence relation. Let $X/\sim = \{X_1, X_2, \cdots, X_k\}$ be the quotient space. This mathematical concept means that: using an equivalence relation, we can decompose

the set X as a disjoint union $X = X_1 \cup X_2 \cup \cdots \cup X_k$ such that the elements in the same X_i are equivalent. We call the number k (the total number of equivalence classes) the quotient space size. Intuitively, k is the number of unconnected sub-graphs of (X, E). We give a straightforward example of the calculation of the above two quantities (Fig. 3.4). First of all, there are 3 unconnected sub-graphs in total, so $k = 3$. For each sub-graph, 1, 1, and 2 balls (centered at the darker nodes) need to be selected as the 1-cover set to cover all the vertices in the graph, so $N_1^c = 4$.

3.3.6 Graph redundancy, intuition and quantification

Intuitively, larger values of the quotient space size and ℓ-covering number indicate a more complicated set of data (with less redundancy). In fact, $x \in B(x', \ell)$ if and only if $d(x, x') \leq \ell$, so x and x' are approximate equal. Hence the covering number can be approximately considered as the total number of vectors in X that are linearly independent. In our implementation we simply use $\ell = 1$, with the consideration of both performance and computation efficiency. Based on the above analysis, we define the graph (layer) redundancy as in Equation (3.10).

$$R(X) = \frac{N}{w_1 k + w_2 N_1^c}, \tag{3.10}$$

where $\{w_1, w_2\}$ is a probability weight that balances the importance of k and N_1^c, N is the number of filters. Besides the graph redundancy, we also investigate other criteria (i.e., the number of filters and principal component analysis (PCA)) to measure the structural redundancy in the ablation study.

3.3.7 Estimate of the 1-covering number.

Since the calculation of ℓ-covering number is NP-hard and time-consuming in practice [133], we propose a lightweight method to estimate N_1^c. Let X_0 be the 1-cover set of a graph X, such that $\#X_0 = N_1^c$. We estimate $\#X_0$ as follows. Fix an integer $\ell (= 1 \text{ or } 2)$

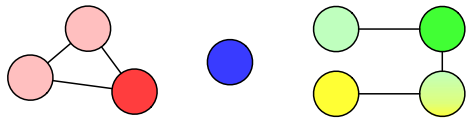


Figure 3.4: An example of the calculation of ℓ-covering number and quotient space size. Darker color nodes indicate the selected ball centers and lighter color nodes represent the elements within the corresponding balls.

and let $x_1^{(\ell)} \in X$, s.t.

$$\deg(x_1^{(\ell)}) = \max\{\deg(x) : \ x \in X\}.$$

We define a finite sequence $\{x_1^{(\ell)}, x_2^{(\ell)}, \cdots, x_{n_\ell}^{(\ell)}\}$ by induction: If we have defined $x_k^{(\ell)}$, then there are two possible cases: (i) $X = \bigcup_{i=1}^{k} B(x_i^{(\ell)}, \ell)$, i.e., the family of balls $\{B(x_i^{(\ell)}, \ell) : \ 1 \le i \le k\}$ is an ℓ-cover of X. Then we stop the construction of the sequence and get $\{x_1^{(\ell)}, x_2^{(\ell)}, \cdots, x_{n_k}^{(\ell)}\}$. (ii) Otherwise, choose (any) $x_{k+1}^{(\ell)} \in X \setminus \bigcup_{i=1}^{k} B(x_i^{(\ell)}, \ell)$, s.t.

$$\deg(x_{k+1}^{(\ell)}) = \max\{\deg(x) : \ x \in X \setminus \bigcup_{i=1}^{k} B(x_i^{(\ell)}, \ell)\}.$$

We repeat the above process eventually, and get the sequence

$$\{x_1^{(\ell)}, x_2^{(\ell)}, \cdots, x_{n_\ell}^{(\ell)}\}, \quad l = 1 \text{ or } 2.$$

It is obvious that we have $N_1^c = \#X_0 \le n_1$ because the family $\{B(x_k^{(1)}, 1) : \ 1 \le k \le n_1\}$ is a 1-cover of X. Moreover, for any $i \ne j$ $(i, j \le n_2)$, we have $d(x_i^{(2)}, x_j^{(2)}) \ge 3$. On the other hand, for each $x_0 \in X_0$, and any $x, y \in B(x_0, 1)$, we have $d(x, y) \le d(x, x_0) + d(x_0, y) \le 2$. Recall that X_0 is a 1-cover set of X, then for any $x_i^{(2)}$, there exists (may not unique) $x_0 \in X_0$ such that $x_i^{(2)} \in B(x_0, 1)$. Moreover, for $i \ne j$, $x_i^{(2)}$ and $x_j^{(2)}$ cannot be in the same ball $B(x, 1)$ (otherwise $d(x_i^{(2)}, x_j^{(2)}) \le 2$, a contradiction). We see $n_2 \le \#X_0 = N_1^c$. Hence $n_2 \le N_1^c \le n_1$.

We can use $\tilde{N_1^c} = \frac{1}{2}(n_1 + n_2)$ as an estimation of N_1^c, if $|n_1 - n_2| \le \epsilon$ and ϵ is acceptably small. Although we cannot theoretically find an upper bound of ϵ, extensive empirical study on various networks indicate that $\tilde{N_1^c}$ is well enough as an estimation of N_1^c, and the computation time of $\tilde{N_1^c}$ is negligibly small (see Section 3.5 for empirical details).

3.3.8 Filter selection strategy

After identifying the layers with large redundancy, we prune unimportant filters from these layers. We can either train a pruned network architecture from scratch with random

initialization or prune certain filters from the pretrained network and do fine-tuning. There are various approaches for unimportant filter selection. In our study, we use a very common and simple strategy, i.e., pruning the filters with smaller absolute weights [41]. This method avoids feeding a large number of training samples into the CNN to get filter rankings, which is usually computationally intensive [40, 42]. But in general, our approach can be used together with any filter selection criterion.

3.4 Experiments

3.4.1 Setup

We first evaluate our approach with the single-shot pruning scheme (pruning a large number of filters at one time), with two-widely used benchmark datasets (CIFAR-10 [134] and ImageNet ILSVRC-2012 [135]) on ResNet. We also present the results with the progressive pruning scheme (pruning a small number of filters and fine-tuning the remaining network for multiple times).

For single-shot pruning, we use ResNet{20,56} on the CIFAR-10 dataset and Resnet50 on ImageNet to evaluate the performance, in terms of accuracy drop and FLOPs reduction. We used the widely-used ResNet architecture as described in [2]. For the CIFAR-10 experiments, the models are trained following the setup in [46]. For the ImageNet experiments, pre-trained models from torchvision are used. We first evaluate the layer redundancy in the pre-trained models and identify the number of filters to be pruned in each layer, with our proposed approach. Then we prune the filter in each layer with the corresponding numbers identified and fine-tune the slimmed network. We follow the fine-tuning strategy in [54]. For CIFAR-10, we fine-tune each pruned network for 200 epochs, with a learning rate starting from 0.1, which is divided by 10 at the epochs 60, 120, and 160. For ImageNet, we fine-tune each pruned network for 150 epochs, with a learning rate starting from 0.1, which is divided by 10 every 30 epochs. For all the models, we use an SGD optimizer with a momentum of 0.9, a weight decay of $2e^{-5}$, and a batch size of

256. For the graph associated parameters, we use $w_1 = 0.35$, $w_2 = 0.65$ to emphasize the importance of ℓ-covering number. We choose $\gamma = 0.034$ to achieve the best performance. We implement the experiments with Pytorch 1.3 [136].

We compare the performance of our approach with several recent channel pruning methods, namely, minimum weight (MW) [41], Taylor expansion [42], average percentage of zero activation neurons (APoZ) [55], soft filter pruning (SFP) [54], discrimination-aware channel pruning (DCP) [137], neuron importance score propagation (NISP) [45], slimmable neural networks (SNN) [128], autopruner (AP) [83], generative adversarial learning (GAL) [138], geometric median (GM) [46], transformable architecture search (TAS) [90], cluster pruning (CUP) [76], ABC [139], trained rank pruning (TRP) [140], soft channel pruning (SCP) [141], and high-hank (HRank) [74].

Progressive pruning usually takes much more time than single-shot pruning. We validate the performance of our approach with AlexNet on the CIFAR-10 dataset and VGG16 on the Birds-200 dataset. We re-implement all of the methods with the same configuration. We prune 10 and 50 filters for AlexNet and VGG16 in each iteration and fine-tune the remaining network for 500 mini-batches with a learning rate of 0.0001. During experiments, we discover that as more filters are removed, there exists less redundancy in the graphs. Slightly increasing γ results in better performance. In our experiments we set $\gamma_{i+1} = \gamma_i \times 1.01$ at each time step i. We plot the accuracy after each step of pruning and fine-tuning, in terms of the number of filters, parameters, and FLOPs pruned from the original network.

For progressive pruning, we compare our proposed approach with the following popular channel pruning methods, with the same pruning settings as our approach: [41], Taylor expansion [42], average percentage of zero activation neurons (APoZ)[55], mean activation [40], and random pruning filters from the network.

3.4.2 Performance evaluation of single-shot pruning

CIFAR-10. Results of pruning ResNet20 and ResNet56 on CIFAR-10 are presented in Table 3.2. Our approach prunes a large percent of FLOPs from both architectures without performance degradation, which outperforms the previous state-of-the-art with an obvious margin. For ResNet20, we prune 45.8% FLOPs and the test accuracy is increased by 0.21%. Our pruned ResNet56 model reduces 53.8% FLOPs and achieves a test accuracy of 93.75%, which outperforms the baseline by 0.37%.

ImageNet. Results of ResNet50 on ImageNet are shown in Table 3.3. Our pruned model with 44.1% FLOPs reduction only loses 0.37% top-1 accuracy and 0.19% top-5 accuracy. When pruning comparable FLOPs, the top-1 accuracy of the previous state-of-the-art approaches usually drop by more than 1%. As the pruning ratio increases to 55.1%, the proposed approach can still achieve a promising test accuracy (1.02% and 0.51% drop for top-1 and top-5 accuracy), which is the best performance compared with recent works. These results verify the effectiveness of our approach on the single-shot pruning scheme. It is worth mentioning that the MW approach can be considered as a baseline of our approach because we add a redundancy identification stage before using MW for filter pruning. It is observed that pruning filters uniformly with MW results in unsatisfactory results (71.24% top-1 accuracy). After identifying the redundancy and pruning the corresponding number of filters in each layer, the performance is significantly improved by around 4%.

3.4.3 Performance evaluation of progressive pruning

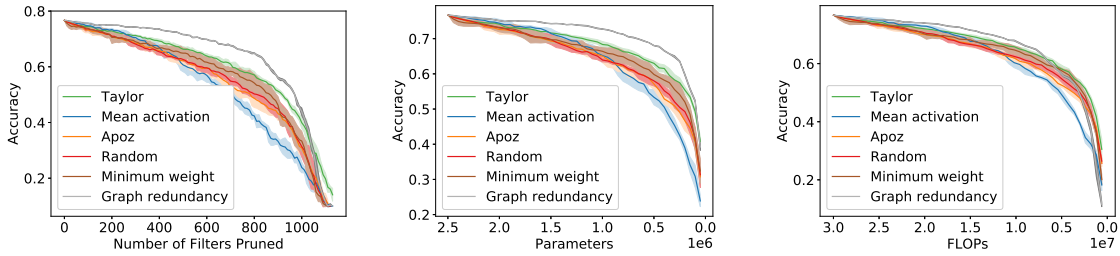
CIFAR-10. We first pre-train an AlexNet on the CIFAR-10 dataset and achieve an accuracy of 76.67%. Results in Fig. 3.5 show that our approach significantly outperforms other methods when reducing the same number of filters or parameters. For example, when pruning 600 (out of 1152) filters from AlexNet, we achieve an accuracy of 69.85%, while the highest performance among the other method is 63.89% (Taylor). Although improvement in terms of FLOPs is not as obvious as the previous two metrics, our approach still achieves better performance than other methods.

Table 3.2: Results of ResNet on CIFAR-10. MW results is with our own implementation. GR is graph redundancy.

Model	Approach	Acc. before p.	Acc. after p.	Acc. drop	FLOPs drop
ResNet20	MW	92.35%	90.93%	1.42%	41.0%
	SFP	92.20%	90.83%	1.37%	42.4%
	GM	92.20%	91.09%	1.11%	42.2%
	TAS	-	92.88%	0.00%	45.0%
	SRR-GR	92.27%	92.48%	**-0.21%**	**45.8%**
ResNet56	MW	93.51%	92.90%	0.61%	51.5%
	NISP	-	93.01%	-	35.5%
	GAL	93.26%	93.38%	-0.12%	37.6%
	DCP	93.80%	93.49%	0.31%	49.8%
	HRank	93.26%	93.17%	0.09%	50.0%
	SFP	93.59%	92.26%	1.33%	52.6%
	GM	93.59%	92.93%	0.66%	52.6%
	TAS	-	93.69%	0.77%	52.7%
	SRR-GR	93.38%	93.75%	**-0.37%**	**53.8%**

Table 3.3: ResNet50 results on ImageNet. Results of MW, APoZ, and Taylor are based on our own implementation.

Approach	Top1 acc. baseline	Top5 acc. baseline	Top1 acc. after p.	Top5 acc. after p.	Top1 acc. ↓	Top5 acc. ↓	FLOPs ↓
MW	76.13%	92.86%	71.24%	90.38%	4.89%	2.48%	41.8%
SFP	76.15%	92.87%	74.61%	92.06%	1.54%	0.81%	41.8%
GM	76.15%	92.87%	75.03%	92.40%	1.12%	0.47%	42.2%
GAL	76.15%	92.87%	71.95%	90.94%	4.20%	1.93%	43.0%
TAS	-	-	76.20%	93.07%	1.26%	0.48%	43.5%
HRank	76.15%	92.87%	74.98%	92.33%	1.17%	0.54%	43.8%
SNN	-	-	74.90%	-	1.10%	-	43.9%
SRR-GR	76.13%	92.86%	75.76%	92.67%	**0.37%**	**0.19%**	**44.1%**
TRP	-	-	74.06%	92.07%	-	-	44.4%
AP	76.15%	92.87%	74.76%	92.15%	1.39%	0.72%	51.2%
GM	76.15%	92.87%	74.13%	91.94%	2.02%	0.93%	53.5%
ABC	76.01%	92.96%	73.86%	91.69%	2.15%	1.27%	54.0%
SCP	75.89%	92.98%	74.20%	92.00%	1.69%	0.98%	54.3%
CUP	-	-	-	-	1.47%	0.88%	54.5%
GAL	76.15%	92.87%	71.80%	90.82%	4.35%	2.05%	55.0%
SRR-GR	76.13%	92.86%	75.11%	92.35%	**1.02%**	**0.51%**	**55.1%**



(a) Number of filters (b) Parameters (c) FLOPs

Figure 3.5: Progressive pruning results of AlexNet on CIFAR-10.

Birds-200. For the experiments of VGG16 on Birds-200, the pre-trained model achieves an accuracy of 76.70%. Fig. 3.6 shows the results after pruning. Our approach exhibits the best performance for nearly the entire range in terms of the number of filters and parameters pruned. We observe that in terms of FLOPs, most pruning criteria perform even worse than randomly pruning filters from the network. This is because progressive pruning approaches remove a fixed number of filters in each iteration. These approaches (including ours) tend to prune the filters in the deeper layers, which are usually with fewer FLOPs than the shallower layers. However, FLOPs is not jointly considered when selecting filters. We leave this issue as future work.

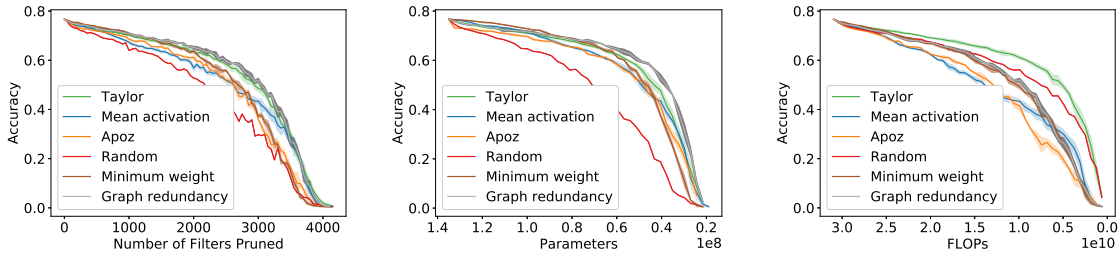
3.5 Analysis and ablation study

In this section, we conduct several experiments to analyze the key components of our proposed approach.

3.5.1 ℓ-covering number estimate and computation time

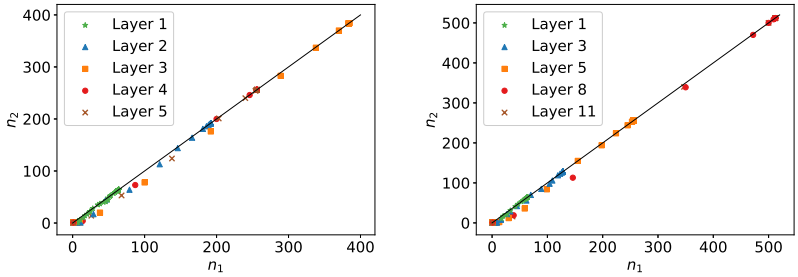
We first show the effectiveness of our approach for l-covering number estimate. We build a series of graphs for each layer of a pre-trained AlexNet and VGG16 by changing γ from 0.001 to 0.3 and visualize n_1 and n_2 (defined in the Methodology section) for the illustration purpose (Fig. 3.7). Obviously, $n_1 \approx n_2$ in nearly all cases. The same trend is also observed in ResNet. Actually, in all of our experiments, we do not observe any large deviations between n_1 and n_2. It is with negligible influence to use $\tilde{N}_1^c = \frac{1}{2}(n_1 + n_2)$ as an estimate of N_1^c, whose values is between n_1 and n_2.

We also evaluate the computing time for estimating the 1-covering number N_1^c (Table 3.4). To obtain N_1^c, a complete search of all combinations of vertices has to be done to see if all the vertices are covered by the selected balls, We name this approach as the oracle approach. We measure the computation time of the oracle approach and our proposed lightweight approach for calculating 1-covering number. It is clear that with the oracle



(a) Number of filters (b) Parameters (c) FLOPs

Figure 3.6: Progressive pruning results of of VGG-16 on Birds-200.



(a) AlexNet (b) VGG16

Figure 3.7: The value of n_1 and n_2 by changing γ from 0.001 to 0.3. Black solid line refers to $n_1 = n_2$.

Table 3.4: Average computation time of the oracle and our proposed method for the 1-covering number calculation/estimate (in secs).

Num of filters	1-covering number	Time with oracle method	Time with our method
64	1	0.001	0.0015
	2	0.045	0.0013
	3	1.384	0.0020
	4	28.21	0.0026
192	1	0.023	0.0014
	2	0.974	0.0027
	3	90.29	0.0028

method, the computation time grows drastically as the number of filters and the actual value of N_1^c increase. It is even not temporally feasible to use the oracle approach when $N_1^c > 4$. In contrast, the time used with our method for $\tilde{N}_1^c$ is negligibly short and is merely influenced by the number of filters in the layer and the value of $\tilde{N}_1^c$. These results indicate that our proposed approach for the estimate of N_1^c is valid and efficient. Therefore, the real running time of the proposed approach is almost the same as existing methods with the same filter selection criteria.

3.5.2 Filter selection criteria

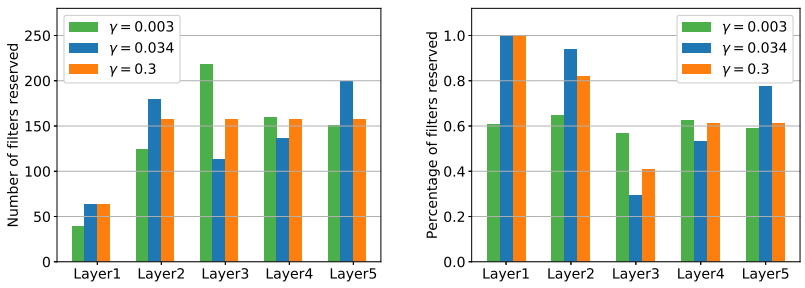
In previous experiments, we use the minimum weight criterion to prune filters. We further investigate whether different filter selection criteria have any influence on the performance. We use AlexNet on CIFAR-10 for illustration, by pruning 30% FLOPs and fine-tuning the remaining networks for 100 epochs with a learning rate of $1e^{-4}$. Results (Table 3.5) indicate that choosing filters with the minimum weight strategy achieves the best accuracy (75.99%). However, using other filter selection criteria results in a similar performance, and the accuracy only drops 0.17% even we randomly prune filters in the layers identified as with large redundancy by our approach. Therefore, our proposed approach is not sensitive to filter selection criteria, which further validates the fact that pruning filters in the layers with large redundancy is more essential than identifying unimportant filters.

3.5.3 The value of gamma

We change the distance threshold γ for graph establishment to analyze its influence on the performance. We keep using AlexNet on CIFAR-10 as an example by pruning 40% of the filters with $\gamma = 0.003, 0.034$, and 0.3. Results (Fig. 3.8) shows that with a large γ, the last four layers remain the same number of filters, which indicates that the layer with the largest number of filters are identified as the redundant layer at each time (Fig. 3.8(a)). When γ is small, nearly the same percent of filters are removed from all layers (Fig. 3.8(b)). With a suitable value of γ (0.034), our approach identified Layer 3 as the most redundant

Table 3.5: Performance with different filter selection criteria after pruning 30% FLOPs of AlexNet.

Approach	MW	MA	Taylor	Random
Accuracy	75.99%	75.84%	75.95%	75.82%



(a) # of filter reserved (b) % of filter reserved

Figure 3.8: The network structure comparison when 40% filters are pruned from AlexNet using different γs.

layer. Layer 4 and 5 are also redundant to some extent but Layer 4 is with a little more redundancy. Different from other existing works, our approach suggests that Layer 1 should not be pruned if we only aim to remove 40% of the filters from AlexNet. These results are consistent with the definition of layer redundancy (in the Methodology section). For a layer with n filters, when $\gamma \to 0$, it is clear that $k = n$, $N_1^c = n$, and $R(X) = 1$. Therefore, all layers have the same level of redundancy and the approach becomes a uniform pruning. When $\gamma \to +\infty$, $k = 1$, $N_1^c = 1$, and $R(X) = n$, which indicates that the layer with the most number of filters is with the largest redundancy. Our approach can be considered as a dynamic architecture search approach controlled by γ.

3.5.4 Other criteria for structural redundancy identification

Since there are few studies that consider network pruning from the perspective of structural redundancy reduction, we further investigate the effectiveness of structural redundancy reduction for channel pruning with the following layer redundancy measurement metrics. (1) SRR-NOF: This strategy simply uses the number of filters in the convolutional layers as the measurement of layer redundancy. The layer with more filters is considered as with more redundancy. In the redundancy identification phase, for each iteration, a filter from the layer with the most number of filters are removed. If there exist more than one layer containing the same number of the most filters, a filter is removed from a randomly chosen layer. When the requirement is reached, the network is pruned with the minimum weight criterion according to the remaining number of filters in each layer. (2) SRR-PCA: This strategy uses principal component analysis (PCA) on the intermediate feature maps of a network to measure the correlation between filters. We first feed training samples to the CNN and record the flattened feature maps of each convolutional layer. Then we fit PCA on these flattened feature maps and get a list of percentage of variance explained by each of them for all convolutional layers. We select the N smallest percentage of variances across all layers and count how many items are selected in each convolutional layer. Finally, we prune the filters in each layer accordingly, with the minimum weight ranking criterion. (3)

SRR-GR: This strategy uses graph redundancy as described in the Methodology section. The training and pruning configuration are the same as in the Experiment section.

Experiment results (Table 3.6) show that by pruning 44% FLOPs from ResNet50, even with a naive layer redundancy measurement (i.e., the number of filters in the layer), the performance is comparable to recent studies. PCA identifies the layer redundancy better than NOF and with SRR-PCA, the drops of top-1 and top-5 accuracy further decrease to 0.94% and 0.38%. With the graph redundancy-based approach, the pruning performance is significantly improved. These results validate that (1) structural redundancy reduction is an efficient approach for channel pruning, and (2) the proposed graph redundancy-based approach is a promising way for layer redundancy measurement.

3.6 Conclusion

In this chapter, we theoretically studied the rationale behind network pruning from a perspective of redundancy reduction via a statistical modeling and discovered that pruning filters in the layer(s) with the most structural redundancy plays a more essential role than pruning the least important filters across all layers. We proposed to identify the level of redundancy existed in each convolutional layer of a CNN via a graph establishment for each layer and two graph-related quantities as the measurement of the redundancy. After that, filters are pruned from the selected layer(s) by a simple filter selection criterion. Experimental results validated that our approach improved the state-of-the-art on image classification tasks. We believe that the proposed approach can be effective on more complicated tasks such as object detection and image synthesis.

Table 3.6: Performance of the pruned ResNet50 networks with different redundancy reduction measurements.

Approach	Top1 acc. baseline	Top5 acc. baseline	Top1 acc. after p.	Top5 acc. after p.	Top1 acc. ↓	Top5 acc. ↓	FLOPs ↓
SRR-NOF	76.13%	92.86%	74.88%	92.27%	1.25%	0.59%	44.0%
SRR-PCA			75.19%	92.48%	0.94%	0.38%	44.1%
SRR-GR			75.76%	92.67%	0.37%	0.19%	44.1%

Chapter 4

Zero-Shot Knowledge Distillation from a Decision-Based Black-Box Model

4.1 Overview

In the last chapter, we have introduced a network pruning approach to reduce the size and computational cost of deep neural networks. In this chapter, we study another approach for neural network compression, i.e., knowledge distillation (KD).

KD is a popular scheme that trains a smaller model (student) to mimic the softmax outputs of a pre-trained over-parameterized model (teacher) [34]. With this approach, the performance of the student model can be improved compared to training the model solely with the cross-entropy loss. The success of KD relies on three factors: (1) access to the teacher's training dataset, (2) the white-box teacher model, i.e., access to the teacher's parameters, and (3) the score-based outputs, i.e., class probabilities of the training samples outputted by the teacher.

In real-world applications, however, these prerequisites are usually unrealistic. Due to storage costs of large training datasets (such as ImageNet [135]) or privacy issues (such as sensitive patient data or personal photos), accessing the training samples are sometimes not feasible. With this concern, the concept of zero-shot knowledge distillation (ZSKD) [101,

102, 104, 37] is proposed. ZSKD generates pseudo training samples via backpropagation with access to the parameters of the white-box teacher, which are then used as the transfer set for training the student model via KD. However, we argue that this scenario is still not realistic under certain circumstances.

In some cases, training samples are publicly available, but pre-trained models are not. For example, YouTube's recommendation system [142] is trained with tons of videos that can be accessed by any user. However, the trained model is a core competitiveness of the company and its parameters are not released. One can argue that a surrogate teacher can be trained locally with the accessible training set, but due to the limitations such as computing resources, its performance is usually not satisfactory compared to the provided powerful model with much more parameters and complicated architectures.

Moreover, a much more challenging scenario is that, in many real-world applications, none of the three factors mentioned above is available. A pre-trained model stored on the remote server may only provide APIs for inference, neither the model parameters nor the training samples are accessible to the users. Worse than that, these APIs usually return a category index for each sample (i.e., hard-label), rather than the class probabilities over all classes. For example, speech recognition systems like Siri and Cortana are trained with internal datasets and only return the results to users [143]. Cloud-based object classification systems like Clarifai [144] just give the top-1 classes of the identified objects in the images uploaded by users.

With these concerns, we propose the concept of decision-based black-box knowledge distillation (DB3KD), i.e., training a student model by transferring the knowledge from a black-box teacher that only returns hard-labels rather than probability distributions. We start with the scenario when the training data is available. Our key idea is to extract the class probabilities of the training samples from the DB3 teacher. We claim that the decision boundary of a well-trained model distinguishes the training samples of different classes to the largest extent. Therefore, the distance from a sample to the targeted decision boundary (the boundary to the samples of a certain class) can be used as a representation of a sample's robustness, which determines how much confidence of a specific class is assigned to the

sample. Based on this, the soft label of each training sample can be constructed with the value of sample robustness and used for training the student via KD.

We further extend DB3KD to the scenario when training data are not accessible. As the decision boundary makes every effort to differentiate the training samples of all classes, samples used for training the teacher tend to be with longer distances to the boundary than others. We propose to optimize randomly generated noises away from the boundary to obtain robust pseudo samples that simulate the distribution of the training samples. This is achieved by iteratively estimating the gradient direction on the boundary and pushing the samples away from the boundary in that direction. After that, pseudo samples are used for training the student via DB3KD. To our best knowledge, this is the first study of KD from a DB3 teacher, both with and without access to the training set.

The contribution of this study is summarized as follows.

- We propose the concept of decision-based black-box knowledge distillation for the first time, with which a student is trained by transferring knowledge from a black-box teacher that only returns hard-labels.
- We propose to use sample robustness, i.e., the distance from a training sample to the decision boundaries of a DB3 teacher, to construct soft labels for DB3KD when training data is available.
- We extend the DB3KD approach to a more challenging scenario when accessing training data is not feasible and name it zero-shot decision-based black-box knowledge distillation (ZSDB3KD). This is achieved by generating pseudo samples distinguished by the teacher's boundaries to the largest extent and using DB3KD for soft label construction.
- Extensive experiments validate that the proposed approaches achieve competitive performance compared to existing KD methods in more relaxed scenarios.

4.2 Knowledge distillation

We first formulate KD in its standard form and present our approach that creates soft labels of the training samples with a DB3 teacher. Finally, we extend our approach to the scenario in which the training set is not accessible.

KD is used for training a compact student by matching the softmax outputs of a pre-trained, cumbersome teacher [34] (Fig. 4.1(left)). For an object classification task, denote $F_t(x)$ and $F_s(x)$ the teacher and the student DNNs, respectively, which take an image x as the input, and output a vector $P \in [0,1]^L$, i.e., $F_t(x) = P_t = \text{softmax}(a_t)$, $F_s(x) = P_s = \text{softmax}(a_s)$, where L is the number of classes and a is the pre-softmax activation. In a KD procedure, a temperature τ is usually introduced to soften the softmax output, i.e., $P^\tau = \text{softmax}(a/\tau)$, which is proved to be efficient to boost the training process. The student is trained by minimizing the loss function in Eq. (4.1).

$$\mathcal{L} = \mathcal{L}_{CE}(P_s, y) + \lambda \mathcal{L}_{KD}(P_t^\tau, P_s^\tau), \tag{4.1}$$

where y is the ground truth label, $\mathcal{L}_{CE}$ and $\mathcal{L}_{KD}$ are the cross-entropy loss and the distillation loss. A scaling factor λ is used for balancing the importance of the two losses.

4.3 Decision-based black-box knowledge distillation

As mentioned, in many real-world applications, users are prohibited from querying any internal configuration of the teacher except for the final decision (top-1 label). Denote $F_t^B(x)$ the DB3 teacher, then $F_t^B(x) = l, l \in \{1, 2, \cdots, L\}$. In this case, P_t cannot be obtained and the student cannot be trained with Eq. (4.1). We claim that a sample's robustness against a specific class can be used as a representation of how much confidence should be assigned to this class, with proper post-operations. Therefore, we extract the sample's robustness against each class from the DB3 teacher and convert it to a class distribution $\hat{P}_t$ as an estimate of P_t (Fig. 4.1(bottom)). In the following, we propose three

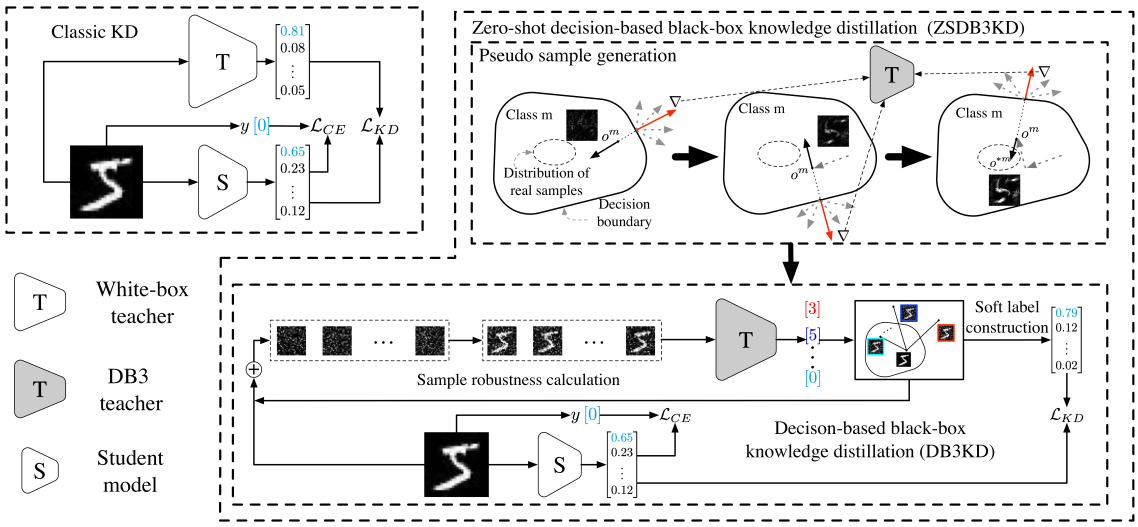


Figure 4.1: The overall workflow of the proposed approach. Left: classic KD. Bottom: decision-based black-box KD (DB3KD). Samples are iteratively fed to the DB3 teacher to compute the sample robustness, which is transformed as soft labels for training the student via KD. Right: Zero-shot DB3KD (ZSDB3KD). Pseudo samples are generated by moving random noises away from the decision boundary and approaching the distribution of the original training samples, which are used as the transfer set for training the student via DB3KD.

metrics to measure sample robustness and present how to construct class distributions with the sample robustness measurements. Intuitively, if a sample is closer to some points in the region of a specific class, it is more vulnerable to this class and thus should be assigned higher confidence.

4.3.1 Sample robustness

Sample Distance (SD). The most straightforward way to quantify the sample robustness is to compute the minimal ℓ_2-norm distance from a sample to those of other classes (Fig. 4.2(left)). Denote $x_0^m \in \mathbb{R}^{C\times W\times H}$ a sample of the m-th class, $\mathbf{x}^n = \{x_1^n, x_2^n, \cdots, x_S^n\}$ a batch of S samples from the n-th class, where $n \neq m$, C, W, H are the number of channels, width and height of the sample, respectively. The robustness of x_0^m against class n is computed with Eq. (4.2).

$$r_0^{m,n} = \min_{1\leq i\leq S} ||x_i^n - x_0^m||_2. \tag{4.2}$$

The advantage of using SD is it can be implemented without querying from the teacher. However, SD is a rough estimate of sample robustness since it does not mine any information from the teacher. Therefore, we introduce two advanced strategies to measure sample robustness.

Boundary Distance (BD). To obtain better representation of sample robustness, we propose to leverage the distances from a sample to the targeted decision boundaries of the teacher (Fig. 4.2(middle)). For each $x_i^n \in \mathbf{x}^n$, we implement a binary search in the direction $(x_i^n - x_0^m)$ and find the corresponding point $\bar{x}_i^n$ on the decision boundary (Eq. (4.3)).

$$\begin{aligned} \bar{x}_i^n = \min_{\alpha}(x_0^m + \alpha \cdot \frac{x_i^n - x_0^m}{||x_i^n - x_0^m||_2}), i = 1, 2, \cdots, S, \\ \text{s.t. } F_t^B(\bar{x}_i^n + \epsilon) = n, \quad ||\epsilon||_2 \to 0. \end{aligned} \tag{4.3}$$

We then compute the sample robustness with Eq. (4.2) in which x_i^n is replaced by $\bar{x}_i^n$.

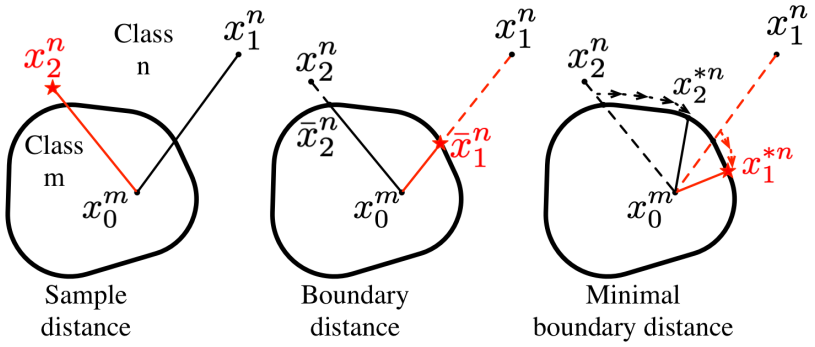


Figure 4.2: Strategies for computing sample robustness.

Minimal Boundary Distance (MBD). Inspired by recent studies of decision-based black-box adversarial attack [145, 146, 147, 148], we further optimize $\bar{x}_i^n$ by moving it along the decision boundary to the point x_i^{*n} where $||x_i^{*n} - x_0^m||_2$ is minimized (Fig. 4.2(right)). Starting from $\bar{x}_i^n$, we first estimate the gradient of the boundary $\nabla F_t^B(\bar{x}_i^n)$ via zeroth order optimization [149], which is achieved by sampling Q Gaussian random vectors $\mathbf{u}_q \in \mathbb{R}^{C\times W\times H}$ $(q = 1, 2, \cdots, Q)$ and averaging them (Fig. 4.3, Eq. (4.4)).

$$\nabla F_t^B(\bar{x}_i^n) = \frac{1}{Q}\sum_{q=1}^{Q} \text{sign}(\bar{x}_i^n + \epsilon_g \mathbf{u}_q)\mathbf{u}_q, \tag{4.4}$$

where ϵ_g is a very small scalar, and $\text{sign}(x_i^n + \epsilon_g \mathbf{u}_q)$ is a sign function, i.e,

$$\text{sign}(x_i^n + \epsilon_g \mathbf{u}_q) = \begin{cases} +1, & F_t^B(\bar{x}_i^n + \epsilon_g \mathbf{u}_q) = n, \\ -1, & \text{Otherwise.} \end{cases} \tag{4.5}$$

Once the gradient is determined, we get a new sample outside the decision boundary $\hat{x}_i^n \leftarrow \bar{x}_i^n + \xi_d \nabla F_t^B(\bar{x}_i^n)$ with a step size ξ_d. Then we conduct the same binary search procedure (Eq. (4.3)) in the direction $(\hat{x}_i^n - x_0^m)$ and obtain an updated $\bar{x}_i^n$. Since the search is within a very small region, the decision boundary in such a region is smooth. Therefore, the new $\bar{x}_i^n$ has a smaller distance to x_0^m (Fig. 4.3). We repeat the procedure above to get the optimal solution $x_i^{*n} = \bar{x}_i^n$ until $||\bar{x}_i^n - x_0^m||_2$ cannot be further minimized or the query limit is reached. Finally, we compute the sample robustness with Eq. (4.2) in which x_i^n is replaced by x_i^{*n}.

4.3.2 Soft label construction

After obtaining all the samples' robustness on all classes, we construct the soft labels for them with proper manipulations. We start with the pre-softmax activations for better illustration. Suppose the pre-softmax activation of a sample x_s^m is $\mathbf{a_s^m} = \{a_{s,1}^m, a_{s,2}^m, \cdots, a_{s,L}^m\}$. Then the pre-softmax activation and the sample robustness should be in correlation with the following conditions. (1) $\text{argmax}_i a_{s,i}^m = m$. It is obvious that $a_{s,m}^m$ should be the largest number to ensure that the sample is assigned to the correct class. (2) If $r_s^{m,j} > r_s^{m,k}$, then

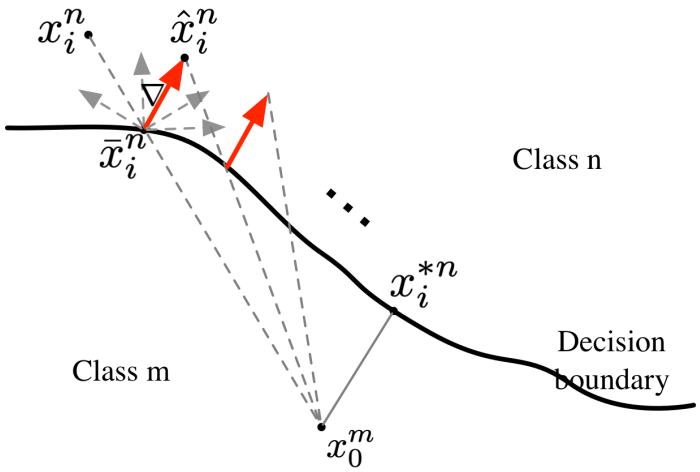


Figure 4.3: The iterative procedure for the optimization of MBD.

$a^m_{s,j} < a^m_{s,k}$. This is because bigger sample robustness indicates longer distance to the targeted decision boundary, which means that the sample is more robust against the certain class and should be assigned a lower confidence. (3) If $\sum_{j=1}^{L} r_s^{m,j} > \sum_{j=1}^{L} r_p^{m,j}, j \neq m$, then $a^m_{s,m} > a^m_{p,m}$. This is because when the sum of a sample's distances to its targeted decision boundaries is larger, the probability mass of this sample is more concentrated in its top-1 class. Otherwise, the mass is more dispersed among all elements.

With the above design philosophy, to meet requirement (1) and (2), we define $\hat{a}^m_{s,n}(n = 1, 2, \cdots, L)$ in Eq. (4.6).

$$\hat{a}^m_{s,n} = \begin{cases} \frac{1}{r_s^{m,n}}, & \text{for } n \neq m, \\ \sum_{i=1}^{L} \frac{1}{r_s^{m,i}}, i \neq m, & \text{for } n = m. \end{cases} \tag{4.6}$$

$\hat{a}^m_{s,n}$ is then divided by $(\sum_{i=1}^{L} \frac{1}{r_s^{m,i}})^2$ to meet requirement (3), as presented in Eq. (4.7).

$$a^m_{s,n} = \frac{\hat{a}^m_{s,n}}{(\sum_{i=1}^{L} \frac{1}{r_s^{m,i}})^2}, \quad i \neq m, \text{ for } n = 1, 2, \cdots, L. \tag{4.7}$$

Finally, we get $\hat{P}_t = \text{softmax}(\mathbf{a}^m_s)$ for sample x^m_s.

4.3.3 Training of student model

Once the soft labels of all the training samples are constructed with the above approach, we can train the student with standard KD, using the objective function in Eq. (4.1).

4.4 Zero-shot decision-based black-box knowledge distillation

In zero-shot KD, pseudo samples are usually generated by optimizing some noise inputs via backpropagation towards some soft labels sampled from a prior distribution, which are then used as the transfer set. However, with a DB3 teacher, backpropagation cannot

be implemented and the prior distribution cannot be obtained, which makes ZSDB3KD a much more challenging task. Since the teacher is trained to largely distinguish the training samples, the distance between a training sample to the teacher's decision boundary is usually much larger than the distance between a randomly generated noise image to the boundary. With this claim, we propose to iteratively push random noise inputs towards the region that is away from the boundary to simulate the distribution of the original training data (Fig. 4.1(right)).

Denote o_0^m and $\mathbf{o}^{\bar{m}} = [o_1^{\bar{m}}, o_2^{\bar{m}}, \cdots, o_T^{\bar{m}}]$ a random noise input of the m-th class and a batch of T random noises with any other class, respectively. Similar but slightly different from Eq. (4.3), for each $o_i^{\bar{m}} \in \mathbf{o}^{\bar{m}}$, we first identity its corresponding points on the boundary $\bar{o}_i^m$ with Eq. (4.8).

$$\begin{aligned} \bar{o}_i^m = \min_{\alpha}(o_0^m + \alpha \cdot \frac{o_i^{\bar{m}} - o_0^m}{||o_i^{\bar{m}} - o_0^m||_2}), i = 1, 2, \cdots, T, \\ \text{s.t. } F_t^B(\bar{o}_i^m + \epsilon) \neq m, \quad ||\epsilon||_2 \to 0. \end{aligned} \tag{4.8}$$

Similarly, the MBDs of o_0^m, i.e., o_i^{*m}, can be iteratively estimated with Eq. (4.4) and (4.5). Let o^{*m} be the one of o_i^{*m} $(i = 1, 2, \cdots, T)$ such that $||o^{*m} - o_0^m||_2$ attains its minimal value, i.e., $||o^{*m} - o_0^m||_2 = \min_i ||o_i^{*m} - o_0^m||_2$. We then estimate the gradient at the boundary $\nabla F_t^B(o^{*m})$ with Eq. (4.4) and update o^m as $o^m \leftarrow o^m - \xi_o \nabla F_t^B(o^{*m})$ with the step size ξ_o. The new o^m is usually with longer distance to the boundary. We repeat the above process until $||o^{*m} - o^m||_2$ cannot be further maximized or the query limit is reached. Finally, we used the generated pseudo samples with the DB3KD approach to train the student as described in Section 4.3.

4.5 Experiments

In this section, we first demonstrate the performance of DB3KD when training samples are accessible. Then we show the results of ZSDB3KD under the circumstance that training data is not accessible.

4.5.1 Experiment setup of DB3KD

We demonstrate the effectiveness of DB3KD with several widely used DNNs and datasets as follows. (1) A LeNet-5 [150] with two convolutional layers is pre-trained on MNIST [150] as the teacher, following the configurations in [100, 102]. A LeNet-5-Half and a LeNet-5-1/5 are designed as the student networks, which contains half and 1/5 number of convolutional filters in each layer compared to LeNet-5, respectively. (2) The same teacher and student networks as in (1) are used but are trained and evaluated on the Fashion-MNIST dataset. (3) An AlexNet [1] pre-trained on CIFAR-10 [134] is used as the teacher. An AlexNet-Half and an AlexNet-Quarter with half and 25% filters are used as student networks. (4) A ResNet-34 [2] pre-trained on the high-resolution, fine-grained dataset FLOWERS102 [151] is used as the teacher, and the student is a ResNet-18.

We evaluate our approach with the three strategies for sample robustness calculation as described in Section 4.3.1, represented as DB3KD-SD, DB3KD-BD, and DB3KD-MBD, respectively. For DB3KD-SD, we use 100 samples from each class to compute the sample robustness r for MNIST, Fashion-MNIST, and CIFAR-10. Since there are only 20 samples in each class of FLOWERS102, we use all of them. Starting with these samples, ϵ is set to $1e^{-5}$ as the stop condition of the binary search in DB3KD-BD. In DB3KD-MBD, we use 200 Gaussian random vectors to estimate the gradient and try different numbers of queries from 1000 to 20000 with $\xi_d = 0.2$ to optimize the MBD and report the best test accuracies. The sample robustness are calculated in parallel with a batch size of 20 with FLOWERS102, and 200 with the other datasets.

With the constructed soft labels, we train the student networks for 100 epochs, using an Adam optimizer (learning rate $5e^{-3}$), for all the datasets except for FLOWERS102, which is trained for 200 epochs. The scaling factor λ is set to 1 for simplicity. Since Eq. (4.7) has the similar functionality with the temperature τ, τ is not need to be as large as in previous studies [34].With a hyperparameter search, we find that smaller τs between 0.2 and 1.0 leads to good performance. We use $\tau = 0.3$ in our experiments. All experiments are evaluated for 5 runs with random seeds.

4.5.2 Performance evaluation of DB3KD

The performance of DB3KD is presented in Tables 4.1 and 4.2. To understand the proposed approach better, we also present the performance of the following training strategies. (1) The teacher and the student networks trained solely with the cross-entropy loss. (2) The standard KD with Eq. (4.1) [34]. (3) Training the student network via KD with a surrogate white-box teacher (Surrogate KD in Tables 4.1 and 4.2), which is used for simulating the scenario in which one can train a smaller but affordable surrogate model with full access to its parameters compared to the powerful DB3 teacher. Here the surrogate has the same architecture with the student. The performance of surrogate KD is considered as the lower bound of DB3KD. (4) Training with the soft labels constructed with randomly generated sample robustness (Noise logits in Tables 4.1 and 4.2), which is used for verifying the effectiveness of DB3KD for soft label construction.

We observe from the results that DB3KD works surprisingly well. With the most straightforward strategy SD, our approach still achieve competitive performance on all experiments compared to standard KD and outperform surrogate KD. When using MBD to compute sample robustness, DB3KD-MBD outperforms standard KD on all the experiments except for FLOWERS102. On FLOWERS102, the performance of DB3KD is slightly worse due to the complexity of the pre-trained teacher model. However, DB3KD still outperforms the surrogate KD with a clear margin. These results validate the effectiveness of DB3KD and indicates that sample robustness with proper post-operation provides an informative representation of a sample's probabilities over all classes and can be used as an alternative to the softmax output when only a DB3 teacher is provided.

We also observe the following phenomena in the experiments. (1) Training with noise logits via KD does not work, but even results in worse performance than training with cross-entropy. It indicates noise logits cannot capture the distribution of class probabilities, but are even harmful due to the wrong information introduced. (2) Training a student with a surrogate teacher not only results in unsatisfactory performance, but is also a difficult task due to the low capacity of the surrogate model. Also, the performance is sensitive to

Table 4.1: Performance evaluation of the proposed DB3KD approach on small datasets (MNIST, Fashion-MNIST, and CIFAR-10).

Algorithm	MNIST		Fashion-MNIST		CIFAR10	
	LeNet5 -half	LeNet5 -1/5	LeNet5 -half	LeNet5 -1/5	AlexNet -half	AlexNet -quarter
Teacher CE	99.33%	99.33%	91.63%	91.63%	79.30%	79.30%
Student CE	99.11%	98.77%	90.21%	88.75%	77.28%	72.21%
Standard KD	99.33%	99.12%	90.82%	89.09%	77.81%	73.14%
Surrogate KD	99.13%	98.85%	90.27%	88.72%	77.49%	72.49%
Noise logits	99.01%	98.72%	89.81%	88.20%	77.04%	72.06%
DB3KD-SD	99.15%	98.98%	90.86%	89.31%	77.66%	72.78%
DB3KD-BD	99.51%	99.19%	90.68%	89.47%	77.92%	72.94%
DB3KD-MBD	99.52%	99.22%	91.45%	89.80%	78.30%	73.78%

Table 4.2: Performance evaluation of the proposed DB3KD approach on high-resolution, fine-grained dataset (Flowers-102).

Algorithm	Accuracy
Teacher CE	95.07%
Student CE	92.18%
Standard KD	94.05%
Surrogate KD	92.93%
Noise logits	91.99%
DB3KD-SD	93.18%
DB3KD-BD	93.30%
DB3KD-MBD	93.77%

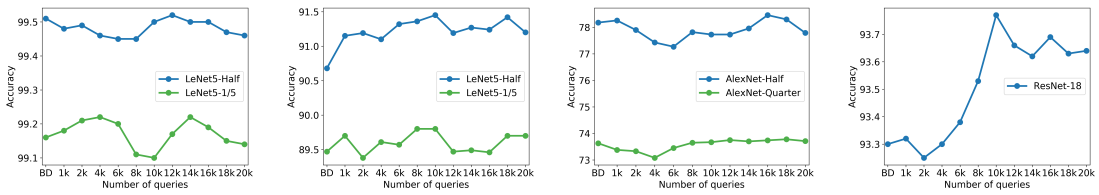
hyperparameter selection (λ, τ, learning rate, etc.). Therefore, training an extra affordable surrogate teacher is not an optimal solution compared to DB3KD.

We notice that in some experiments, surprisingly, DB3KD even works better than standard KD, though the models are trained with a more challenging setting. A reasonable hypothesis is that, for some problems, the distance between a training sample to the decision boundary may provide more information than the softmax output. These results provide future research directions that the dark knowledge behind the teacher's decision boundary is more instructive compared to the teacher's logits in certain cases.

4.5.3 Ablation studies and analyses of DB3KD

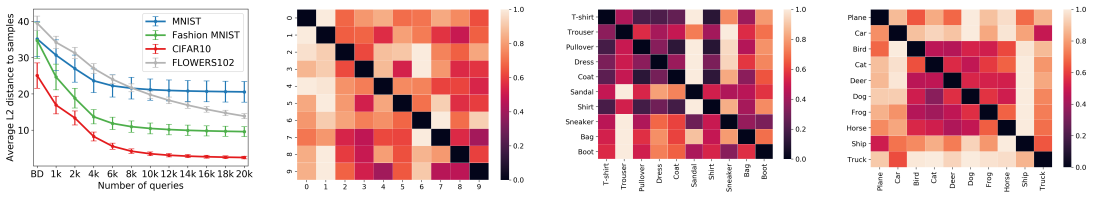
We conduct several ablation studies and analyses for further understanding of the effectiveness of DB3KD.

Number of queries in label construction. We first investigate whether different numbers of queries used for computing sample robustness has any influence on the performance. For each dataset, we query from the teacher for a variety of times from 1000 to 20000 to compute the sample robustness (Fig. 4.4). It can be observed that with more queries, the student models perform slightly better, especially for deeper architectures (ResNet) and high-resolution datasets (FLOWERS102). In general, the student models perform well with various numbers of queries. Even using a binary search with around 100 queries (DB3KD-BD), the performance are satisfactory on all student models. This is because the quality of a sample's soft label is largely related to its robustness against different classes. Moreover, the MBD used for computing sample robustness shows a highly positive correlation with the number of queries (Fig. 4.5(a)). The ratios of sample robustness against different classes remain stable against the number of queries. Therefore, it is not necessary to optimize the MBD with a large number of queries, which indicates that DB3KD is query efficient. It is also worth noting that the performance is not linearly correlated with the query numbers. This is because for all experiments, we use the same set of hyperparameters for fair comparison, which may not be optimal as the query number increases. However,



(a) LeNet5-MNIST (b) LeNet5-FMNIST (c) AlexNet-CIFAR10 (d) ResNet-FLOWERS

Figure 4.4: Performance comparison with different numbers of queries for computing sample robustness.



(a) (b) MNIST (c) Fashion-MNIST (d) CIFAR10

Figure 4.5: (a) The average minimal boundary distances over number of queries. Error bar indicates one standard deviation. (b-d) Normalized average minimal boundary distances of the samples of different classes. Darker colors indicate smaller distances between two classes.

we'd like to emphasize the performance is not sensitive to query numbers and is satisfactory with a wide range of numbers (from 2k to 20k).

Although the boundary may be complex in the pixel domain and the boundary sample may be fragile, what we actually care about is the minimal boundary distance (MBD). It actually measures how fragile a training sample is against other classes and is a robust measurement. As supplementary evidence, the standard deviations of the MDBs are relatively small (shown with the error bars in Fig. 4.5(a)), indicating the robustness of the proposed approach.

Correlation between sample robustness and class probability. To further analyze the effectiveness of DB3KD for constructing soft labels, we visualize the normalized average MBDs of the samples with different classes (Fig. 4.5(b-d)). It is observed that classes semantically closer with each other are with smaller distances to their decision boundary. For example, in MNIST, the distance between '8' and '9' is smaller than '8' and '1' because '8' looks more like '9' than '1'. Therefore, a sample of '8' is assigned higher confidence in class '9' than '1'. Similarly, in Fashion-MNIST, 'T-shirt' looks more like 'shirt' than 'sneaker' so that their distance are smaller. In CIFAR-10, samples of the 'dog' class are with smaller distances to the boundary with 'cat' than 'truck' since 'dog' and 'cat' are semantically closer. These analyses confirm the consistency between sample robustness and class probability distribution.

4.5.4 Experiment setup of ZSDB3KD

We evaluate ZSDB3KD with (1) a LeNet-5 and a LeNet-5-Half (on MNIST and Fashion-MNIST), and (2) an AlexNet and an AlexNet-Half (on CIFAR-10) as the teacher and the student. The networks are the same as in Section 4.5.1.

We optimize the pseudo samples for 40 ($\xi_o = 0.5$) and 100 iterations ($\xi_o = 3.0$) for the two LeNet-5 and the AlexNet experiments, respectively. The query is limited to 5000 when iteratively searching for the MBD. We generate 8000 samples for each class with a batch size of 200 for all the experiments. We use data augmentation to enrich the transfer

set. We use 5000 queries for computing the sample robustness since we have shown the number of queries is trivial. Other parameters are the same as the DB3KD experiments. We compare the performance of ZSDB3KD with several popular KD approaches in more relaxed scenarios, including FSKD [97], BBKD [98], Meta KD [100], DAFL [102], ZSKD [101] and DFKD [37].

4.5.5 Performance comparison of ZSDB3KD

The performance of ZSDB3KD on MNIST and Fashion-MNIST, and CIFAR-10 presented in Table 4.3 and 4.4 show that ZSDB3KD achieves competitive performance. The accuracies of the student networks are 96.54% and 72.31% on MNIST and Fashion-MNIST, which are quite close to other KD approaches with more relaxed scenarios (training data or the teacher's parameters are accessible). On CIFAR-10, our AlexNet-Half model achieves an accuracy of 59.46% without accessing any training samples and the softmax outputs of the teacher. It is worth noting that using random noise as the input results in very poor performance with a DB3 teacher. These results indicate that the samples generated with our proposed approach indeed capture the distribution of the samples used for training the teachers.

4.5.6 Ablation studies and analyses of ZSDB3KD

In this subsection, we perform several studies to understand the effectiveness of ZSDB3KD, using LeNet-5-Half trained on MNIST as an example.

Iteration of sample generation. We first evaluate the performance of the student with pseudo samples generated with different iterations (Fig. 4.6(upper right)). As expected, the performance is improved as the samples are optimized away from the decision boundaries with more iterations. As shown in Fig. 4.6(left), with more steps, more pixels in the pseudo samples are activated, with sharper edges and recognizable digits, which indicates that the samples become more robust as we keep moving them to the opposite of the gradient direction on the decision boundaries.

Table 4.3: Result of ZSDB3KD with MNIST and Fashion-MNIST. S: score-based teacher. D: decision-based teacher.

Algorithm	Data	Model	MNIST	FMNIST
Teacher CE	Yes	White	99.33%	91.63%
Student CE	Yes	White	99.11%	90.21%
Standard KD	Yes	Black-S	99.33%	90.82%
FSKD	Few	White	86.70%	72.60%
BBKD	Few	Black-S	98.74%	80.90%
Meta KD	Meta	White	92.47%	-
DAFL	No	White	98.20%	-
ZSKD	No	White	98.77%	79.62%
DFKD	No	White	99.08%	-
ZSDB3KD	No	Black-D	96.54%	72.31%

Table 4.4: Result of ZSDB3KD on AlexNet with CIFAR-10.

Algorithm	Data	Model	Accuracy
Teacher CE	Yes	White	79.30%
Student CE	Yes	White	77.28%
Standard KD	Yes	Black-S	77.81%
FSKD	Few	White	40.58%
BBKD	Few	Black-S	74.60%
DAFL	No	White	66.38%
ZSKD	No	White	69.56%
DFKD	No	White	73.91%
Noise input	No	Black-S	14.79%
Noise input	No	Black-D	13.53%
ZSDB3KD	No	Black-D	59.46%

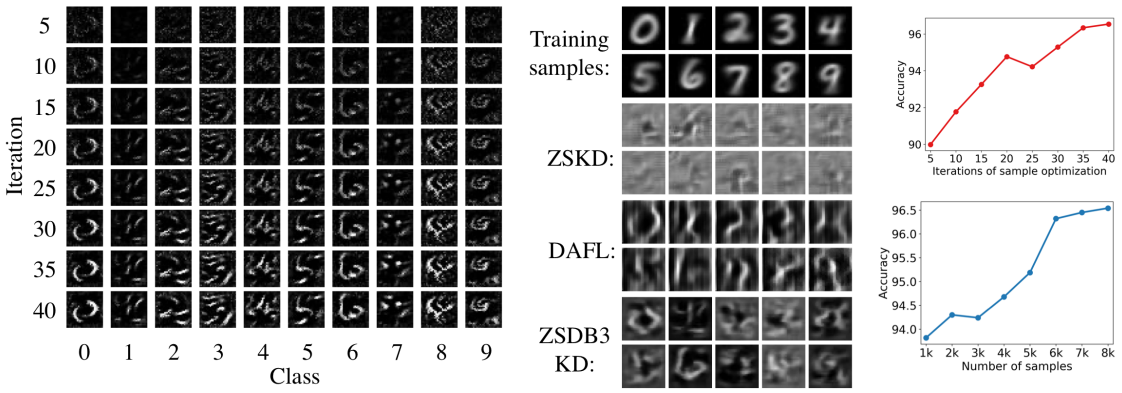


Figure 4.6: Analysis and ablation study of ZSDB3KD with MNIST. Left: evolution of pseudo images over iterations. Middle: averaged images compared to other white-box zero-shot KD approaches. Upper right: the accuracies with different iterations of sample generation. Bottom right: the accuracies with different numbers of samples used for training the student.

Number of samples used for training. We then investigate the effect of the number of pseudo samples used for training on the performance of the student network. The results of training the student network with different numbers of generated samples (from 1k to 8k per class) are presented in Fig. 4.6(bottom right). Not surprisingly, with more samples, the test accuracy increases. Even with a small number of samples (1k per class), the student network can still achieve a competitive performance of 94% test accuracy. With 8k samples per class, the student's performance gets saturated and is comparable to the performance of standard KD.

Visualization of generated samples. As mentioned above, we have shown the evolution of individual samples over iterations (Fig. 4.6(left)), which gradually exhibits clear digits. To have a further visualization of the generated pseudo samples, we further average 1k samples for each class as shown in Fig. 4.6(middle). Even though generated with a DB3 teacher, the samples are with a satisfactory quality compared with the averaged samples generated with ZSKD and DAFL that use white-box teachers.

4.6 Conclusion

In this chapter, we introduced KD from a decision-based black-box teacher for the first time. We proposed DB3KD to deal with this problem, which uses sample robustness to construct the soft labels for the training samples by iteratively querying from the teacher. We also extend DB3KD to a much more challenging scenario in which the training set is not accessible and named it Zero-shot DB3KD (ZSDB3KD). Experiments on various networks and datasets validated the effectiveness of the proposed approaches.

Part II

Modeling Cell Migration with Efficient Deep Reinforcement Learning

Chapter 5

Neighbor Relationship Determination in Metazoan Embryos

5.1 Overview

Cell neighbor determination is a significant component in the simulation of a metazoan embryo system since it influences a number of fundamental biological processes, such as cell signaling, migration, and proliferation. Therefore, before introducing modeling cell migration, we first present a neighbor relationship determination model that can determine whether two cells are neighbors with each other given a certain embryonic circumstance, which is used a lot during the cell migration simulation process.

Recent breakthroughs in cutting-edge live microscopy and 3D time-lapse imaging technologies allow biologists to observe the dynamics in diverse organisms and gather a tremendous amount of data with cellular resolution. Such data explosion promotes the research in specific developmental aspects, including but not limited to cell-cell signaling, cell migration, and proliferation. Recently, the regulatory mechanism of cell asymmetric division was discovered in [152]. A study analyzed the loss of function phenotypes of genes that functioning substantially during significant signaling processes during embryogenesis, and a multi-scale model was generated including a regulatory

landscape of lineage differentiation [153, 154, 155]. In the above studies, to determine the neighbors of certain cells is a fundamental prerequisite. Signaling processes, especially those during early embryogenesis, are accomplished by a contacted interaction from one cell to another. The moving/dividing speed and direction of a cell are influenced to some extent by its neighbors that generate squeezing forces to it during its migration and division processes. Therefore, to obtain a correct neighbor relationship is the core of the success of the *in silico* simulation and *in vivo* experiment focusing on the above problems.

Voronoi diagram [156] is wildly used for cell neighbor determination and its upstream applications, such as cell volume calculation and embryo segmentation in recent years [157, 158, 159]. By assigning every grid to the target point of (weighted) minimum distance with proper resolution, Voronoi diagram provides biologists with a sufficiently approximate solution to the above problems with the consideration of the complexity of a tissue. However, obtaining a Voronoi tessellation is not a computationally simple task, since it is difficult to derive an analytical solution. For a metazoan embryo (or similar situations), if we consider the asymmetric division of a cell, which leads to different volumes of the following generations, a weighted 3D Voronoi diagram has to be implemented, which makes things much worse. As a result, traversal approaches [160] are largely used for such kind of problems. The task becomes extremely time-consuming as the number of cells in the system grows exponentially.

Machine learning, especially its popular branch, deep learning, has been developing at a tremendous speed in recent years and opened new doors to a variety of fields [161, 1, 162, 112]. We are provided with unprecedented opportunities to consider the traditional neighbor determination problem with such completely new point of view. As mentioned above, cellular resolution, minute level embryogenesis data contain rich information that allows us to reduce the computational redundancy of the Voronoi approach. With properly extracted features and well-designed framework, we can train an efficient classifier to achieve real-time cell neighbor determination with very little accuracy loss. In this chapter, we transform the neighbor determination problem to a classification task. We preprocess the raw embryogenesis data as neighbor samples and label them, extract appropriate

features and train the classifier. An accuracy of 99.66% is acquired and by adding certain constraints in specific real world problems, the accuracy further increases to 99.98%.

5.2 Methodology

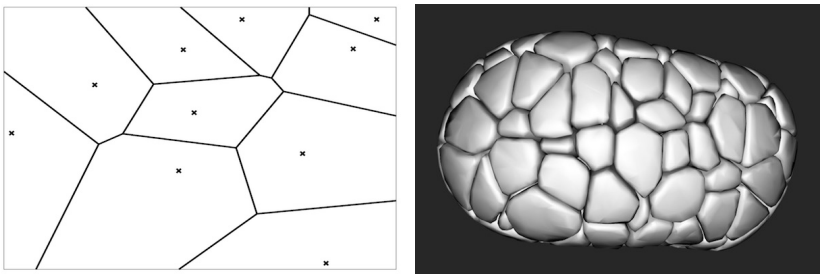
5.2.1 Voronoi diagram

Voronoi diagram (Fig. 5.1(a)) is a unique space partition that contains certain points and a corresponding region with every point [163]. Suppose there are a set of target points $P = \{p_1, p_2, ..., p_n\}$ in the space. For each point x belongs to the region $dom(p_i)$, x is closer to p_i than any other ps in the space (Eq. (5.1)). Voronoi diagram and its variants, such as weighed Voronoi diagram, are wildly used in various fields, such as forest simulation [164], wireless sensor networks (WSNs) [165], and embryo segmentation (Fig. 5.1(b)) [166].

$$dom(p_i) = \{x \in R^m | d(x, p_i) < d(x, p_j), i \neq j\}, \tag{5.1}$$

where $d(x, p)$ is the Euclidean distance between x and p.

Voronoi diagrams are usually generated with heuristic algorithms. Space is divided into small grids with proper resolution and the distances between a grid and each point in the space are calculated and compared to assign it to its nearest point. Even though several preprocessing approaches that can reduce the computational complexity have been proposed [167], they have the following defects. (1) These algorithms focus on certain sub-problems derived from Voronoi diagrams, such as nearest neighbor query, path planning, and collision detection. Unfortunately, none of them solves our neighbor determination problem in the embryo system. (2) These algorithms require a preprocessing step as a prerequisite before reducing the computational complexity and a fixed Voronoi diagram is needed. However, in a metazoan embryo system, cells migrate, divide and squeeze with each other over time, which leads to highly dynamics of the Voronoi tessellation. Therefore, these approaches are not feasible under such situations. The most classical approach to obtain the Voronoi neighbor is Delaunay triangulation, which connects all the neighbor



(a) General case. (b) Embryo segmentation

Figure 5.1: Voronoi diagrams.

pairs in the whole diagram. Unfortunately, for a 3D weighted Voronoi diagram, there does not exist a way to obtain its Delaunay triangulation besides the heuristic approaches [168].

In conclusion, the Voronoi diagram, as an approach for neighbor determination, is not able to achieve fast processing. In a metazoan embryogenesis process, which contains hundreds to thousands of cells over hours to days of development, simulation time is much more valuable than that we spend to construct other pre-trained/processed built-in modules. As a result, a training/preprocessing expensive, but testing cheap framework for cell neighbor determination is required.

5.2.2 Framework

The whole framework is described in Fig. 5.2. First of all, proper features are extracted from the raw embryogenesis data, and corresponding labels are assigned, indicating that under such circumstance, whether the two cells are neighbors (using 1) or not (using 0). Then we design/select an appropriate classifier and feed it with the samples. The performance is evaluated during the testing process and feedback is given to the classifier to adjust its hyperparameters. Finally, the well-trained model is built into specific real world problems to model more complicated tasks, such as cell-cell signaling and cell division.

5.2.3 Feature extraction

To select proper features to train the classifier is the key for acquiring high accuracy. For the cell neighbor determination problem, an appropriate set of features can accurately provide sufficient information to describe the neighbor relationship between cells without redundancy. Here we list some of the features that can be the candidates for the neighbor determination problem.

- Distance between two cells.
- Radius (or volume) of each cell.
- Length of each body axis (or volume) of the embryo.

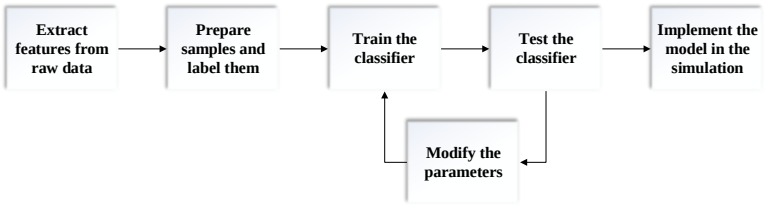


Figure 5.2: The framework of the proposed cell neighbor determination model.

- Distance between each cell to the eggshell.
- Total number of cells in the embryo.
- The distribution of the generations of the cells in the embryo.

5.2.4 Classification

Two classical classifiers, namely, k-nearest neighbor (kNN) and support vector machine (SVM) [169], are implemented in our model. These algorithms are linear models that achieve a fast implementation for cell neighbor determination. In Section 5.4, we will discuss the potential probability to carry out some advanced non-linear models, such as the convolutional neural network (CNN), to acquire higher accuracy.

The kNN classifier aims to find the nearest k points to a testing sample. The result of the number of nearest neighbors is transferred as a maximum posterior probability (Eq. (5.2)) to determine whether the two cells are neighbors with each other given the situation described in the sample.

$$P(\omega_i|x) = \frac{p(x|\omega_i)P(\omega_i)}{p(x)} = \frac{K_i}{K}, \tag{5.2}$$

where $p(x|\omega_i) = K_i N_i/V$ is the probability density function (pdf) for each class, $P(\omega_i) = N_i/N$ is the prior probability, and $p(x) = K/(NV)$ is the normalization constant. K, K_i, N, N_i, V denote the total number of points in the hypersphere, the total number of points that belongs to ω_i in the hypersphere, the total number of samples, the total number of samples that belongs to ω_i, and the volume of the hypersphere, respectively.

The SVM classifier, also known as the large margin classifier, finds the best decision boundary that linearly separates the samples in the different classes. Such boundary holds largest minimum distances to both of the two classes by selecting proper support vectors. With SVM, the cell neighbor determination problem is transferred to an optimization problem that tries to minimize the loss function L (Eq. (5.3)).

$$L = C\sum_{n=1}^{N}\max(0, 1 - y_n w^T x_n) + \frac{1}{2}||w||^2, \tag{5.3}$$

where C is the hyperparameter, N is the total number of training samples, $y_n \in \{-1, 1\}$ depends on the specific label of the sample, w is the parameter of the classifier. The second term in Eq. (5.3), $1/2||w||^2$, is a regularization penalty and the Euclidean distance is used for this case.

5.3 Experiments

5.3.1 Simulation setup

Our simulation uses Nematode *C. elegans* as a study model. Because of its invariant cell lineage, fast developmental speed (only a few hours) from a zygote to hundreds of cells, *C. elegans* is a very suitable model for large-scale systematic quantitative analyses [154] and the existing data have been well formatted for further study. In our experiments, *C. elegans* embryogenesis data from [170] is used as the raw dataset. We cut off the dataset and reserve those from 0 to 350-cell stage. About 2 million samples are generated from 3D weighted Voronoi diagram as "correct answers" of neighbor relationship. For both classifiers (kNN and SVM), 20% of the data are used for training and the rest for testing. A 5-fold cross validation is implemented to guarantee the validity of the result. The runtime performance is tested on the hydra supercomputer from UTK's EECS department [171].

Besides the distance of two cells and the total number of cells in the embryo as the selected features, we use the concept of "relative radius" as the feature of each cell in the sample. The advantage of introducing this concept is that we can reduce the complicated implementation of estimating the volume of the cell, and the lengths of body axes of the eggshell, since for a single organism with consistent format data, these features are exactly the same for every sample. Although we implement this simplified version of feature selection in our current model, we strongly believe that such model is without loss of

generality, and can achieve as good accuracy as this simulation study by adding the missing features when dealing with the situation that embryo sizes vary.

During the very early stage of embryogenesis, asymmetric division commonly exists, especially in the germline cells. In Table 5.1, a list of the predefined ratio of volumes after asymmetric cell division in our simulation is provided. For other cells, relative radii are calculated based on symmetric division.

5.3.2 Runtime performance and accuracy

Running time is a significant factor when we build *in silico* models. The Voronoi approach iterates every grid in the 3D space, which is extremely time-consuming. What makes things worse is that such iteration has to be done three times with different orders of the axes to eliminate the miss of a neighbor pair where the Voronoi boundary is exactly along an axis. kNN compares the distance between a testing sample to all the training samples to find a certain number of nearest neighbors, while SVM optimizes a set of parameters and only a matrix multiplication is needed for a testing sample. Therefore, both kNN and SVM run at a much faster speed than the Voronoi approach. Due to the features of the two classifiers, as the number of cells increases, SVM performs better than kNN on the running time.

Fig. 5.3 shows the running time comparison of the approaches using the Voronoi diagram, kNN, and SVM. The Voronoi approach is extremely time-consuming as the number of cells grows exponentially. But for the other two methods, the running times are controlled in an appropriate time. At the 350-cell stage, our proposed approaches increase the running speed by 10^4 times. Although SVM is more training expensive than kNN (training time at 350-cell stage: 370s vs. 18s), we care more about the simulation time, i.e., the testing time in our case, and SVM saves half time with the comparison to kNN (testing time at 350-cell stage: 72s vs 146s).

In our setup, the results generated from the Voronoi diagram are considered as the "correct answer" of the neighbor relationship, so the accuracy for the Voronoi approach is 1 at each stage as a reference. During early stages (0 to 100-cell), though kNN performs

Table 5.1: Volume ratios after asymmetric cell division during *C. elegans* early embryogenesis.

Cell Name	Daughter 1	Daughter 2	Ratio
P_0	AB	P_1	55:45
P_1	EMS	P_2	54:46
P_2	C	P_3	53:47
P_3	D	P_4	52:48

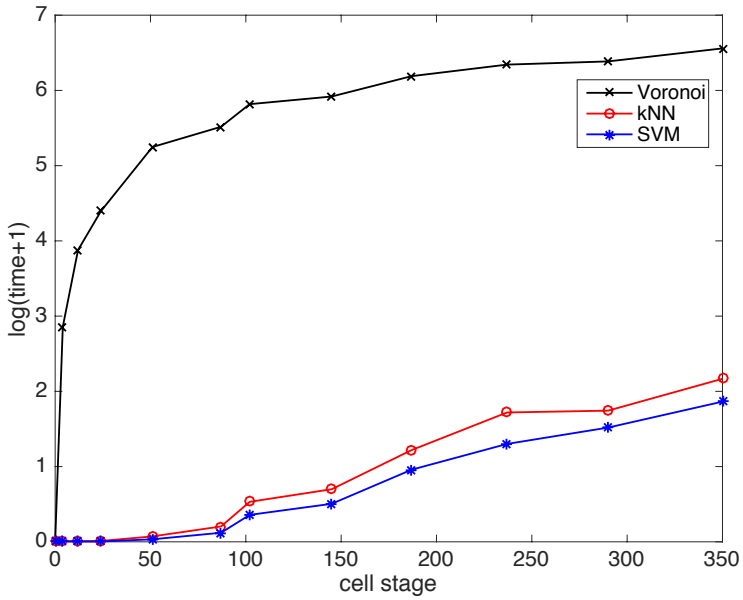


Figure 5.3: Running time comparison from 4 to 350-cell stage.

slightly better than SVM, neither classifier gives a satisfactory accuracy. But as the number of cells increases, the accuracy increases correspondingly, and SVM performs as good as kNN. When the simulation ends at 350-cell stage, the accuracies with kNN and SVM is 99.62% and 99.66%, respectively (Fig. 5.4).

In conclusion, at the early stage (0 to 100-cell), there is no big difference of the running time, but kNN achieves higher accuracy. At late stage (100 to 350-cell), SVM performs as good as kNN on the accuracy, but can save half of the running time.

It is worth noting that sometimes there exist multiple sample points in which certain cell pairs develop in their cell cycle simultaneously. In specific real world problems, to regard certain cell pairs as neighbors or not from just one-time classification is superficial. Modifications can be made under specific situations to increase the accuracy. For example, during a cell-cell signaling process, if we set a proper threshold, and a neighbor relationship is not confirmed until they contact over certain time intervals. The risk that the classifier generates a wrong result is then decreased under extreme circumstances. Therefore, in certain applications, we can modify the definition of accuracy as follows. If there exist more than 10% of the sample points that the classifier makes mistakes, then a wrong result of cell neighbor relationship is confirmed. Based on this improvement, the accuracy of our model further increases to 99.98%.

5.3.3 Verification and visualization

To verify the validity of our proposed approach in specific problems, we implement it in *C. elegans* Notch signaling pathways, and the cell-cell squeeze model for cell division direction. The Notch signaling pathways play significant roles in *C. elegans* embryogenesis [172]. During four Notch signaling processes, certain cells interact in a contactable way to activate certain gene expressions, which leads to different cell fates in the AB-lineage. We test the neighbor relationships of the involved cells in the four Notch signaling pathways listed in Table 5.2. More specifically, signaling and receiving cells are used for testing the neighbor relationship, while others listed in the "unsignaled cell" column are used for

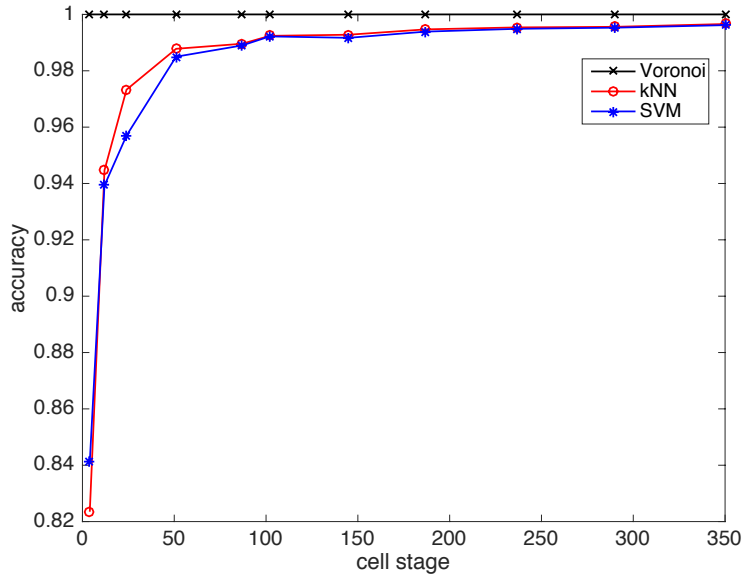


Figure 5.4: Accuracy comparison from 4 to 350-cell stage.

Table 5.2: Cells involved in the four Notch signaling pathways in *C. elegans embryogenesis*.

#	Signaling cell	Receiving cell	Unsignaled cell*
1st	P_2	ABp	ABa
2nd	MS	ABalp,ABara	ABala,ABarp
3rd	ABalapp	ABplaaa	ABpraaa
4th	MSapa,MSapp	ABplpapp	ABprpapp

* The cells required not to be the neighbors of the signaling cell during certain Notch pathways.

testing the algorithm that there does not exist a neighbor relationship between those and the signaling cells, which is exclusively required by each of the four Notch processes.

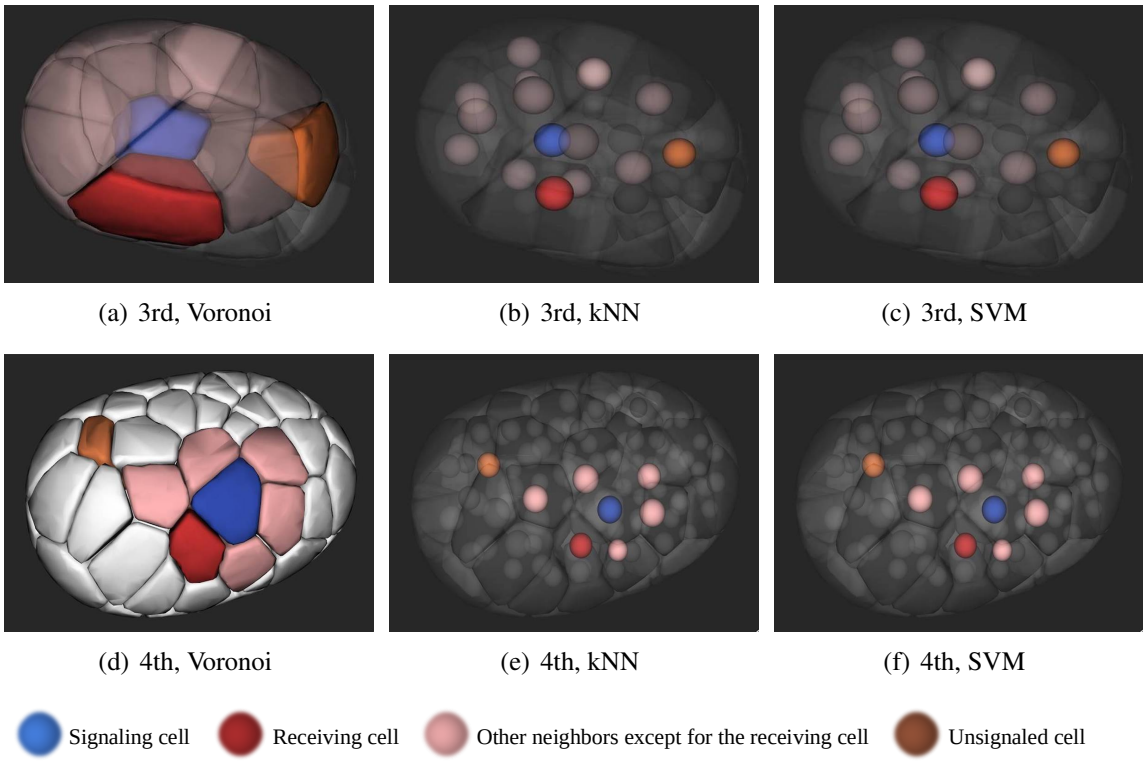
The neighbor relationship of the involved cells in the third and fourth Notch interactions are visualized via WormGUIDES [166] with the approach of Voronoi diagram, and our learning based approaches, respectively (Fig. 5.5). The results from Voronoi diagram are pre-examined to be consistent with the real world situations, and our approaches successfully capture the neighbor relationship in the Voronoi diagram (data not shown for the first two Notch cases).

Division/proliferation is one of the most fundamental cell behaviors, and the squeeze force between cells is a significant component that influences the division direction of a cell. Our model can be used for the determination of the neighbors of a dividing cell. A single cell-cell squeeze direction is then calculated by comparing the sum of the estimated cell radii and their spatial distance. Finally, we combine all the direction vectors associated with a squeeze distance k to obtain the composition direction. We test the model with *C. elegans* embryogenesis data from 4 to 350-cell stage and the whole system works well with our learning based cell neighbor determination approach. Some samples of the result (two daughters of ABal at AB8 stage) of the cell-cell squeeze direction during division process is presented in Table 5.3. The modeling framework for other division direction components, and the composition of them is beyond the scope of this paper.

5.4 Discussions

5.4.1 Analysis of the accuracy rate

Accuracy is the most important issue we care about given that the proposed cell neighbor determination framework has largely increased the speed for calculating it. We consider the reasons for the errors in the following aspects. (1) The Voronoi diagram itself is an approximate approach for embryo segmentation. The real situation is that there exist gaps between cells, and the boundaries are not as sharp as those from Voronoi diagram, either.



(a) 3rd, Voronoi (b) 3rd, kNN (c) 3rd, SVM

(d) 4th, Voronoi (e) 4th, kNN (f) 4th, SVM

Figure 5.5: The neighbor relationship from the Voronoi, kNN and SVM approaches during the 3rd and 4th Notch signaling pathways of *C. elegans* embryogenesis.

Table 5.3: Cell-cell squeeze direction of ABala and ABalp with their neighbors at AB8 stage.

Dividing cell	Location	Radius	Cell stage
ABala	(321, 181, 105)	92.52	16
ABalp	(334, 250, 90)	92.52	16

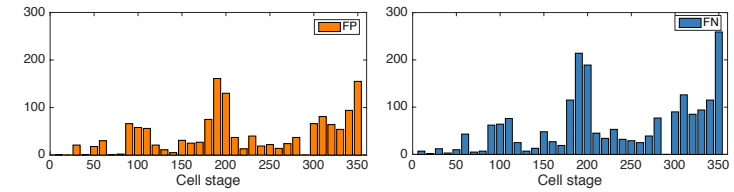
Dividing cell	Neighbor	Location	Distance	Radius	Squeeze dist
ABala	ABalp	(334, 250, 90)	71.80	92.52	113.24
	ABara	(254, 194, 65)	79.11	92.52	105.93
	ABarp	(220, 198, 130)	105.43	92.52	79.61
	ABplaa	(299, 238, 140)	70.41	46.26	68.35
ABalp	ABara	(254, 194, 65)	100.80	92.52	84.24
	ABala	(321, 181, 105)	71.80	92.52	113.24
	MSa	(276, 276, 55)	72.56	88.78	108.74
	ABplp	(251, 312, 120)	107.86	92.52	77.18
	ABplaa	(299, 238, 140)	62.20	46.26	76.56
	ABplap	(280, 245, 145)	77.24	46.26	61.52

(2) The linear classification models have their own limitations, which make the decision boundaries not separate the samples well enough.

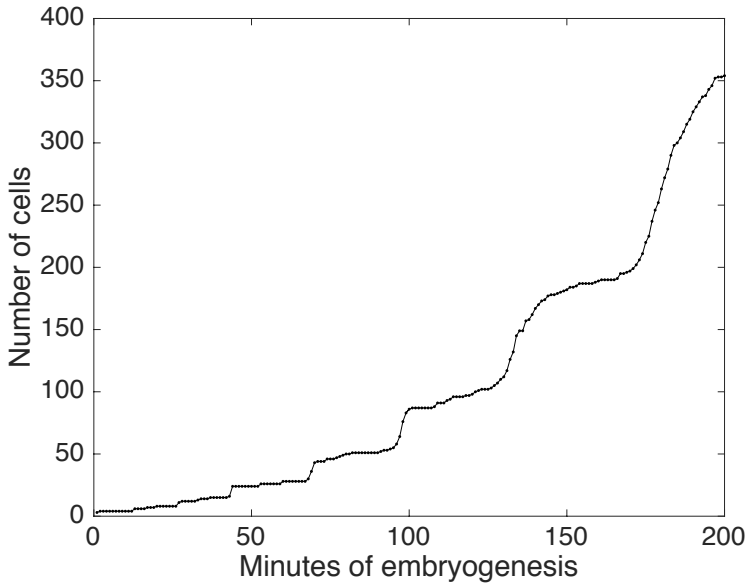
We divide the error into two categories, namely, false positive (FP, the two cells are not neighbors but the classifier believes they are.) and false negative (FN, the two cells are neighbors but the classifier believes they are not.) errors, and analyze them respectively. By counting the number of recognition errors during certain times (represent as cell stage), we find that the time intervals in which errors occur more frequently are centered around only a few cell stages, namely, 90, 190, and 350-cell stage (Fig. 5.6(a) and 5.6(b)). With further analysis of the number of cells in the embryo over time (Fig. 5.6(c)), we find that such time intervals are exactly the periods that the number of cells grows exponentially when the cells from the AB lineage divide extensively at similar times. As a result, the 3D geometry structure changes intensively, which leads to more errors.

We count the number of neighbor determination errors in every cell pair in their cell cycles (Fig. 5.7(a)), and most errors happen less than three times (793 out of 859, 93% in FP results, 947 out of 1069, 89% in FN results). To further validate the assumption we make in Section 5.3.2, the frequency of these errors are presented in Fig. 5.7(b). We find out that most of the errors in certain cell pairs occur at a rate of less than 10% (696 out of 859, 81% in FP results, 839 out of 1069, 78% in FN results), which follows that our assumption to use 10% of the time steps as a threshold to determine the neighbor relationship in an overall time window is reasonable.

An interesting spot in Fig. 5.7(b) is that there exists a slight increase of the frequency of errors in the interval from 90% to 100%. Therefore, we list these cell pairs, the number of errors, and the total number of time steps they exist simultaneously in the embryo, respectively (Table 5.4). It turns out that these high frequencies of errors occur when the two cells are not from the same generation, which means that one cell is just born, and the other is at the very end of her cell cycle. Therefore, the high frequency of errors results from the short time interval they exist in the embryo simultaneously, rather than the extensive errors generated from the classifier.

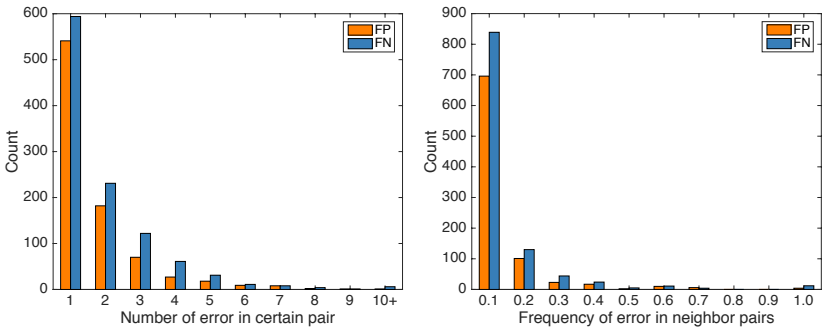


(a) (b)



(c)

Figure 5.6: Distribution of the FP and FN results during embryogenesis simulation.



(a) (b)

Figure 5.7: The number and frequency of errors of all cell pairs in FP and FN results.

Table 5.4: Cell neighbor pairs with a high frequency of errors (90% to 100%) in FP and FN results.

Cell pair	Gens*	Error°	Existence†	Cat
Cpp,Cpaa	6,7	3	3	FP
Dap,Dppp	7,8	1	1	FP
P_4,MSppap	4,8	1	1	FP
ABalapapa, ABalpppaap	9,10	1	1	FP
MSppaaa, ABalaapaaa	9,10	3	3	FN
Cpap,Cpaaa	7,8	1	1	FN
ABalpp, ABplapa	6,7	1	1	FN
ABprpa, ABarapp	6,7	1	1	FN
ABplaaaaa, ABplaaaapp	9,10	3	3	FN
ABalapp, ABalapaa	7,8	1	1	FN
ABplaaap, ABalapapa	8,9	1	1	FN
Daa,Dppa	7,8	1	1	FN
Daa,Dppp	7,8	1	1	FN
ABarappaa, ABprpapaaa	9,10	2	2	FN
Caaaa,Cpaaaa	8,9	1	1	FN
ABaraaa, ABarapap	7,8	1	1	FN

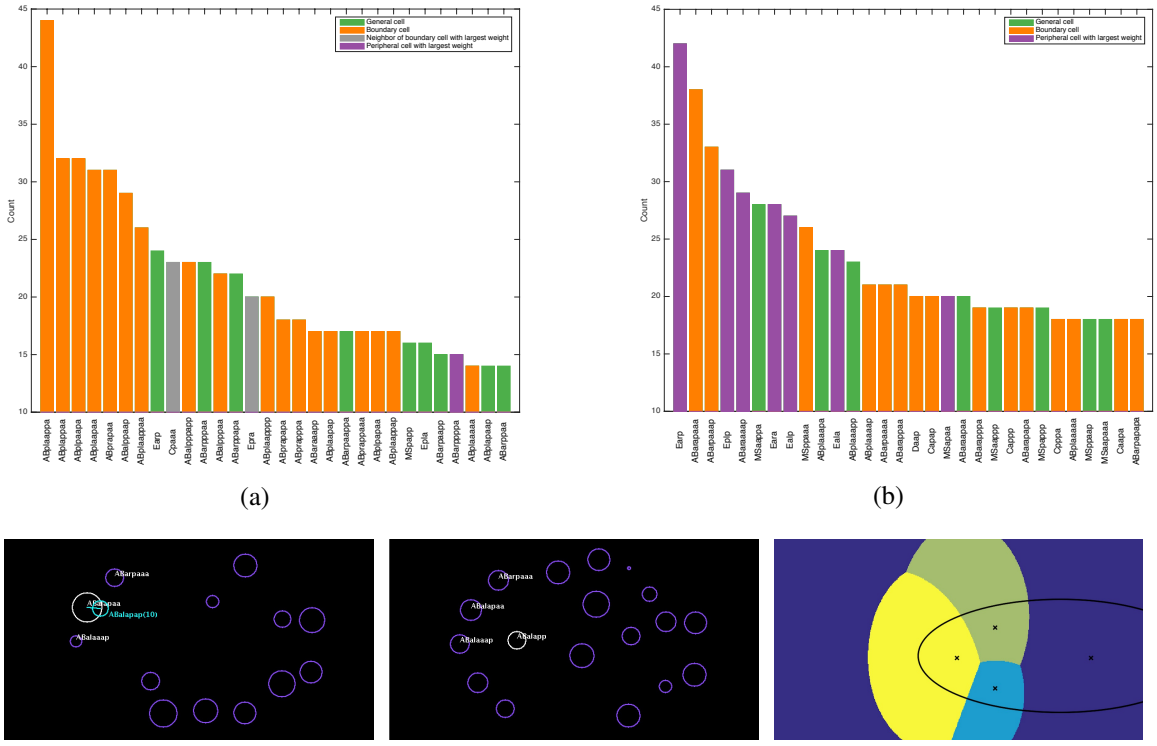
* Generations of both cells.

° Number of errors occurring in their life cycles.

† Number of time steps of the existence in their life cycles.

It is worth noting that in Fig. 5.7(a), the total number of FN results is larger than that of FP ones (1069 vs. 859). Moreover, for the frequency interval of 10% to 40%, FN results is more than FP ones (141 vs. 198). To study the reason behind it, we count the thirty cells with the most number of errors in FP and FN results, respectively (Fig. 5.8(a) and 5.8(b)). Besides the fact that boundary cells close to the eggshell are more likely to be classified inaccurately in both cases, there exist a large number of errors that occur in peripheral cells with the largest weight around, which are the biggest ones near the eggshell, but are not the neighbors of the boundary cells, in the FN results. Further study indicates that the weighted Voronoi approach has a certain probability to generate wrong neighbor labels outside the embryo under the circumstance that the boundary of the eggshell is not estimated appropriately.

Analysis from AceTree [173] validates that for some of the boundary cells and peripheral cells, Voronoi approach gives a wrong segmentation, which leads to FN cases from the classifier, but is actually true results. Fig. 5.5(a) and 5.5(b) show an example in which the boundary cell ABalapaa and the peripheral cell ABalapp are not neighbors (at least they are separated by ABalapaa's posterior sister ABalapap). Under the situation that the peripheral cell ABalapp holds the largest weight near by, the Voronoi approach provides the classifier with a wrong label, while the classifier gives the correct answer, but is considered as an FN result. Further study of the 3D weighted Voronoi diagram shows that for the points with a large weight, encapsulated Voronoi tessellation is generated to nearby cells because of the distances calculated with the ratio of their different weights, even though they are actually not neighbors (Fig. 5.5(c)). Therefore, when the boundary of the eggshell is not estimated appropriately, a number of non-existent neighbor pairs are generated outside the embryo. On the other hand, learning-based approaches consider the distances between the features, which results in FN results that contradict those from the Voronoi approach, but is actually true.



(a) (b)

(c) (d) (e)

* Boundary cells: cells that are close to the eggshell.

° Peripheral cells: cells that are near the eggshell, but are not the neighbors of the boundary cells.

Figure 5.8: Top 30 cells with the most errors in FP and FN results, and analysis from AceTree.

5.4.2 Customization on different purposes

To assign different penalties on FP and FN results can improve the model, since they have different influences on the downstream applications. In the cell-cell squeeze model, it is reasonable to assume that the risk from FP results is much smaller than those from FN ones. This is because when we take the false neighbor pair from the FP results to calculate the squeeze force afterward, the sum of the radii of the two cells is very likely to be larger than the distance between them, and no extra force is generated to influence the division direction. Even if the force is generated unfortunately, since the embryo is a highly dynamic system, the extra vector from wrong results leads to a movement towards the direction of it, which is likely to be neutralized at the next time step when there does not exist FP result.

For the neighbor determination during signaling process, if various of cells contain certain receptor that can be activated by the same ligand through specific pathways, both FP and FN results lead to irreversible consequences. For example, during the first Notch signaling process, the germline cell P_2 expresses the Notch ligand APX-1/Delta, which can be received by both of the daughters of AB (ABa and ABp) with the receptor GLP-1/Notch. ABa and ABp are originally identical, but because of the anterior-posterior relationship, the posterior cell ABp is a neighbor of P_2 while ABa is not. As a result, the expression of the ref-1 family in ABp is induced and this leads to different cell fates of ABa and ABp. Suppose that both FP and FN results are wrongly generated from the model on the neighbor relationship of (P_2,ABa) and (P_2,ABp), respectively. It is obvious that a fate exchange will occur in the ABa and ABp cells. Under such circumstance, FP and FN results should be assigned with the same penalty. Therefore, for the signaling process, bad results on the neighbor relationship may have very large influences on the following processes. For the situations that regulatory mechanisms remain unknown, the operation to assign different penalties on FP and FN results should be treated cautiously.

5.5 Conclusion

In this chapter, we present a cell neighbor determination algorithm to discover the neighbor relationship in the metazoan embryo system. Our idea transfers the traditional Voronoi diagram approach into a classification problem, and takes kNN and SVM as the classifiers, which are trained by the "correct answers" generated from Voronoi. Simulation results show that our approach increases the running speed by 10^4 times. The accuracy for single time step recognition is 99.66%, and such result can be further increased to 99.98% by adding specific restrictions. Our approach successfully models the neighbor relationship in *C. elegans* Notch signaling pathways and cell-cell squeeze forces prove its validity for the cell neighbor determination problem in metazoan developmental systems.

Chapter 6

Cell Migration Modeling with Deep Reinforcement Learning

6.1 Overview

Recent developments in cutting-edge live microscopy and image analysis provide an unprecedented opportunity to systematically investigate individual cells' dynamics and quantify cellular behaviors over extended period of time. Systematic single-cell analysis of *C. elegans* has led to the highly desired quantitative measurement of cellular behaviors [174, 153, 154, 175]. Based on 3D time-lapse imaging, the entire cell lineage can be automatically traced, and quantitative measurements can be made on every cell to characterize its developmental behaviors [176, 157, 177, 178]. These massive recordings, which contain hundreds to thousands of cells over hours to days of development, provide a unique opportunity for cellular-level systems behavior recognition as well as simulation-based hypothesis testing.

Agent-based modeling (ABM) is a powerful approach to analyze complex tissues and developmental processes [123, 179, 180]. In our previous effort, an observation-driven, agent-based modeling and analysis framework was developed to incorporate large amounts of observational/phenomenological data to model the individual cell behaviors

with straightforward interpolations from 3D time-lapse images [181, 182, 183]. With the ultimate goal being to model individual cell behaviors with regulatory mechanisms, tremendous challenges still remain to deal with the scenarios where regulatory mechanisms lag data collection and potential mechanistic insights need to be examined against complex phenomena.

Directional cell movement is critical in many physiological processes during *C. elegans* development, including morphogenesis, structure restoration, and nervous system formation. It is known that, in these processes, cell movements can be guided by gradients of various chemical signals, physical interactions at the cell-substrate interface and other mechanisms [184, 185, 186]. It remains an open and interesting challenge as to What and how one could learn about the rules and mechanisms of cell movement from the movement tracks recorded in live imaging.

This chapter presented a new approach to study cell movement by adopting deep reinforcement learning approaches within an agent-based modeling framework. Deep reinforcement learning is good at dealing with high-dimensional inputs and can optimize complex policies over primitive actions [111], which naturally aligns with the complex cell movement patterns occurred during *C. elegans* embryogenesis, especially where regulatory mechanisms are not completely studied. More specifically, deep neural networks can be adopted to characterize the cell movement within an embryonic system. The neural network takes information from 3D time-lapse images as direct inputs, and the output is the cell's movement action optimized under a collection of regulatory rules. Since deep reinforcement learning can optimize the cell migration path over considerable temporal and spatial spans in a global perspective, it overcomes the local optimization problem encountered by traditional rule-based, agent-based modeling that uses greedy algorithms.

We tested our model through two representative scenarios during *C. elegans* embryogenesis: *the anterior movement of Cpaaa via intercalation* and *the rearrangement of the superficial left-right asymmetry*. In the first case, we proposed two hypotheses for the intercalation of Cpaaa, and simulation results indicated that Cpaaa experienced an active directional movement towards the anterior, which is caused by the continuous effects from a

longer distance, rather than a passive process in which it is squeezed to the target location by its neighbors' movements. In the second case, the frequently occurring "leader-follower" mechanism was also supported by the simulation results of the asymmetry rearrangement. In summary, this framework presented a reverse engineering perspective to investigate regulatory mechanisms behind a certain developmental process: By formulating the reward functions as the representation of regulatory mechanisms, different hypotheses can be tested via reinforcement learning procedures. By comparing the extent of similarities between the simulation cell migration paths and the observation data, such hypotheses can either be supported or rejected, which can facilitate new explanations of certain cell movement behaviors. The model can also be used to study cell migration paths in mutants or other metazoan embryo/tissue systems when related data are given.

6.2 Modeling approach

In our modeling framework, an individual cell is modeled as an agent that contains a variety of information on its fate, size, division time, and group information. For a wild-type *C. elegans* simulation, the cell fate and division information can be directly derived from predefined observation datasets. For more complicated cases that involve gene mutation and manipulation, the developmental landscape can be incorporated for the purpose of modeling [154]. More detailed design information on the agent-based model can be found in [181]. In this study, the cellular movements are treated as results of inherited and genetically controlled behaviors regulated by inter- or intracellular signals, and these cell movements are also constricted by the neighbor cells and the eggshell.

We further assume that the migration path of an individual cell is the optimal path that a cell can use to migrate under a collection of regulation networks and/or constraints within a physical environment. Then we can transform the cell movement problem into a neural network construction and learning problem using observational and/or predefined rules. Therefore, neural networks can be constructed inside each cell to represent its behaviors, and the reinforcement learning method can be used to train the neural networks from

3D time-lapse imaging (with information on locations of cells, their neighbor lists, and other cell interactions after automated cell lineage tracing [187]). After training, the neural networks can determine a feasible and optimal cell migration path in a dynamic embryonic system, but the migration path is still controlled and constrained by the underlying regulation networks and the physical environment.

While the regulation networks can be defined at cellular, group, tissue, or even embryonic levels, only the individual cell movement and group movement are examined and modeled in this study.

6.2.1 Individual cell movements

Two basic kinds of individual cell movements are investigated. The first movement pattern is directional movement, in which the regulation network presents strong signals (such as morphogen gradient or planar cell polarity [188]) and results in directional individual cell movements. The second type of cell movement, defined as passive cell movement, represents the scenarios in which no explicit movement patterns are observed when the signals from regulation networks are weak or canceled out.

Directional cell movement

At this stage, with strong regulation signals from regulation networks, cell movement is mainly controlled by the potential destination and physical pressures from neighbor cells or the eggshell. The destination of cell movement can be defined as a spatial location or region within the embryonic system when regulatory mechanisms are not well studied, or it can be defined as a location next to a specific cell.

Passive cell movement

At this stage, without strong overall regulation mechanisms, cell movement is mainly controlled by the physical pressures between neighbor cells or the eggshell. Therefore, it is defined as passive cell movement with a high level of randomness.

6.2.2 Collective cell migration

In a *C. elegans* embryonic system, individual cells can also be a part of functional group with group-specific communication and regulation mechanisms. In collective cell migration, all the cell movements are directional. However, depending on the role of cell movement, the cells in collective migration can be further categorized as leading cells and following cells.

6.3 Methods

6.3.1 ABM framework

An ABM platform was adopted to present fundamental cell behaviors, including cell fate, division, and migration for a wild-type *C. elegans* in which all cell fates are predefined. The framework, which retains two fundamental characteristics (cell movement and division) for *C. elegans* early embryogenesis is illustrated in (Fig. 6.1). We use the terminologies "intelligent cell" and "dumb cell" to represent the cell that learns its migration path, and those move based on the observation dataset, respectively. At each time step, each cell first moves to its next location, based on either the observation data (that contain cells' locations at different time points) or the output action from the neural network, depending on their specific identities (dumb cells or the intelligent cell). After that, if it is at the right time for division, a new cell is hatched. The global timer is updated when all the cells have acted at a single time step, and such a loop repeats until the end of the process.

6.3.2 Cell movement via deep Q-network

As mentioned in the Modeling Approach section, cell movement has been modeled as a reinforcement learning process [189] in which an agent (cell) interact with the environment (other cells in the embryo and the eggshell) to achieve predefined goals. In an individual cell movement case, an intelligent cell always tends to seek the optimal migration path

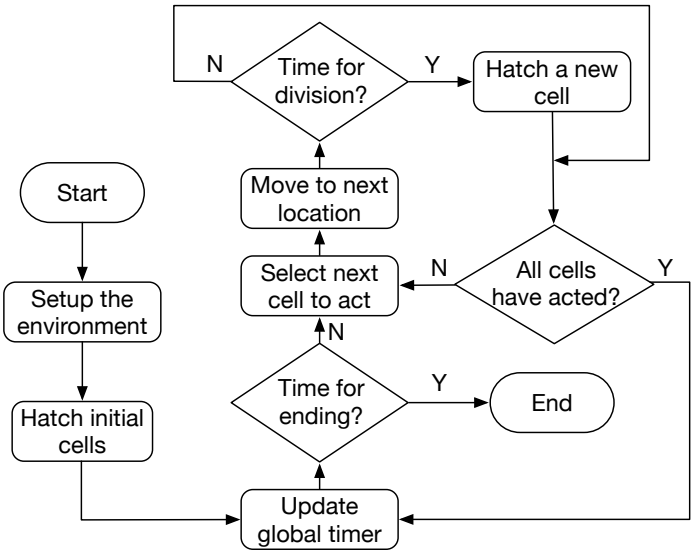


Figure 6.1: The ABM framework. Cells move at each time step based on reading the observed locations (dumb cells) or the output of the neural network (intelligent cell). After a cell's movement, if it is at the right time for division, a new cell is hatched. Such a process repeats until the end of the simulation.

towards its destination based on the regulatory rules. At each discrete time step t, the cell senses its environmental state $S_t \in \mathcal{S}$ from the embryo and selects an action $A_t \in \mathcal{A}$, where the set of $\mathcal{A}$ includes the candidate actions at that state. The embryo returns a numerical reward $R_t \in \mathcal{R}$ to the cell as an evaluation of that action based on the state. Finally, the cell enters the next state S_{t+1} and repeats the process until a terminal condition is triggered. The intelligent cell's objective is to maximize the overall rewards collected during the process. The whole process is demonstrated in Fig. 6.2.

Traditionally, tabular-based Q-learning approaches were largely used for reinforcement learning tasks with modest amounts of input states. However, a dynamic agent-based embryogenesis model usually contains hundreds of cells that act at high temporal and spatial resolutions. Millions of different states are generated during a single embryogenesis process, which cannot be handled by traditional tabular-based Q-learning algorithms. Recent breakthroughs in reinforcement learning that incorporate deep neural networks as mapping functions allow us to feed in high-dimension states and obtain the corresponding Q-values that indicate a cell's next movement [111, 108]. Such a deep Q-network (DQN) outperforms most of the previous reinforcement learning algorithms.

Framework

We implemented a DQN customized for cell movement modeling. It contains two main loops: a cell migration loop and a network training loop (Fig. 6.3). At each time step in the cell migration loop, a state tracker is used for collecting the input state as a representation of the environmental conditions (details in Section 6.3.4). An ϵ-greedy strategy is implemented to balance the exploration and exploitation. Specifically, ϵ is a hyperparameter in $[0, 1)$. A random number x is sampled from a uniform distribution $U(0, 1)$ each time before the selection of an action. If $x \in [\epsilon, 1)$, the intelligent selects a random action, obtains a reward and moves to the next location. Otherwise, the movement action is calculated by feeding the input state to the neural network. Such a process repeats until a terminal condition is triggered. For the training loop, the DQN is established based

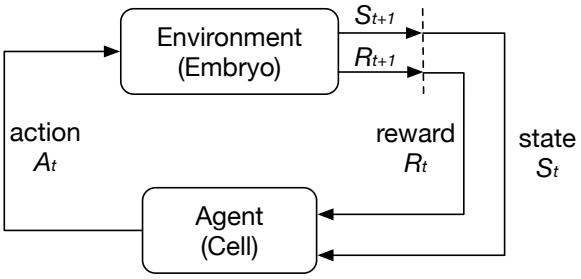


Figure 6.2: The reinforcement learning framework. A cell interacts with the embryo. At each time step, the cell receives a state S_t, selects an action A_t, gets a reward R_t and enters the next state S_{t+1}. The cell's objective is to maximize the total rewards received.

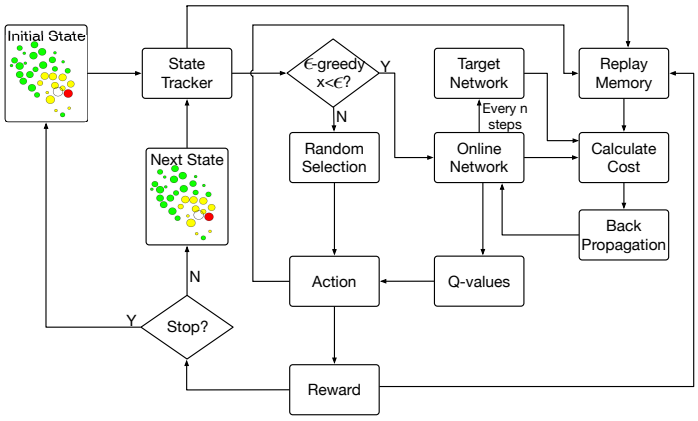


Figure 6.3: The deep Q-network framework for cell movement, which cotnains a cell migration loop and a network learning loop. The intelligent cell's movement is selected via the ϵ-greedy mechanism, from either a random sampling of all the possible actions or the output of the neural network. Then it gets a reward, moves to the next location, and repeats this process. The samples generated from the cell migration loop are used to update the parameters of the neural network via backpropagation. Experience replay and target network are implemented to improve the performance.

on traditional Q-learning algorithms. Rather than searching a Q-table to find the maximal value of $Q(S_t, A_t)$, Q-values are obtained through a neural network parameterized by a set of weights θ. The inputs of the neural network are the tuples (S_t, A_t, R_t, S_{t+1}) gathered from the migration loop. The update process (Eq. (6.1)) can be achieved by minimizing the loss function $\mathcal{L}$ (Eq. (6.2)) and backpropagating the loss through the whole neural network to update θ by $\theta_{t+1} = \theta_t - \alpha \nabla_\theta \mathcal{L}(\theta_t)$ [190]. Therefore, the intelligent cell will gradually choose better actions as the training process proceeds.

$$\begin{aligned} Q(S_t, A_t|\theta_t) \leftarrow & \, Q(S_t, A_t|\theta_t) \\ & + \alpha \left[R_t + \gamma \max_a Q(S_{t+1}, A_{t+1}|\theta_t) - Q(S_t, A_t|\theta_t) \right], \end{aligned} \tag{6.1}$$

$$\mathcal{L}(S_t, A_t|\theta_t) = \left[R_t + \gamma \max_a Q(S_{t+1}, A_{t+1}|\theta_t) - Q(S_t, A_t|\theta_t) \right]^2, \tag{6.2}$$

where α is the learning rate and $\gamma \in (0, 1)$ is the discount factor, which determines the present value of future rewards [189].

In order to improve the system's performance, we utilized two mechanisms: experience replay [111] and target network [108] in the framework. Experience replay cuts off the correlation (which is one of the sources of instabilities) between samples by storing the movement tuples (S_t, A_t, R_t, S_{t+1}) in a replay memory and sampling them randomly during the training process. This is because the capacity of the replay buffer is much larger than the number of samples generated in a single process (from the beginning to a terminal state), and the randomly selected samples for training at each time will come from various processes, which are much less related with each other than those consecutive samples from a single process. In a DQN with a single neural network, the target for gradient descent is always shifting as θ is updated at each time step. Therefore, rather than calculating the future maximal expected reward $\max_a Q(S_{t+1}, A_{t+1}|\theta_t)$ and updating the weights in a single neural network, a target network, which has the same architecture as the original network (called the online network in the new scenario) but parameterized with θ_t^-, was implemented for the calculation of $\max_a Q(S_{t+1}, A_{t+1}|\theta_t^-)$. The weights θ_t^- remains

unchanged for all n iterations until they are updated with θ_t from the online network. This mechanism reduces the oscillations and improve the stabilities of the framework. The improved process is represented in Eq. (6.3).

$$\begin{aligned} Q(S_t,A_t|\theta_t) \leftarrow & \, Q(S_t, A_t|\theta_t) \\ & + \alpha \left[R_t + \gamma \max_a Q(S_{t+1}, A_{t+1}|\theta_t^-) - Q(S_t, A_t|\theta_t) \right] \end{aligned} \tag{6.3}$$

The neural network, which is fed with the embryo state and outputs a Q-value for each action, contains three hidden layers, with 512, 1024, and 1024 nodes, respectively. The Rectified Linear Units (ReLU) was implemented as the activation function after all the hidden layers except for the output layer.

Regulatory mechanisms and reward settings

In the reinforcement learning scenario, the regulatory mechanisms that guide cell movements can be transformed to reward functions as an evaluation of how well a cell moves during a certain period of time based on those mechanisms. For the physical constraints of the cell movements, we defined the following two rules:

- *Collision*: Cells cannot squeeze too much with each other. The closer two cells are, the larger penalty (negative reward) they receive.
- *Boundary*: Cells cannot break through the eggshell. The closer the cell is to the eggshell, the larger penalty (negative reward) it receives.

For both of the above rules, as a threshold of distance is reached, a terminal condition is triggered and the process restarts. For the directional cell movement, an explicit destination is given as a simplified third rule when other regulatory mechanisms are missing:

- *Destination*: A cell always seeks the optimal path towards its target location.

This rule can be replaced as more specific regulatory mechanisms are discovered (e.g., following a leading cell or becoming the neighbor of a certain cell), or new hypotheses are formulated.

6.3.3 Behaviors of the dumb cells

The automated cell lineage tracing technology was utilized to obtain the information of cells' identities and locations from 3D time-lapse microscopy images. These information were used to model the non-intelligent cells' (dumb cells') movement. Because the temporal resolution of our observation data is one minute, and an ABM simulation often requires a much smaller tick interval, a linear interpolation is implemented between two consecutive samples to calculate the next locations of these cells. Additionally, we added a random noise for each movement by sampling it from a normal distribution whose mean value and standard deviation were averaged from the locations of the cells of 50 wild-type *C. elegans* embryos [170].

6.3.4 Behaviors of the intelligent cell

For the intelligent cell, an ϵ-greedy strategy was implemented, which makes it not only act based on past experiences to maximize the accumulated rewards most of the time but also gives it a small chance to randomly explore unknown states. Usually, the value of ϵ is set to increase (the probability of random exploration decreases) as the training process proceeds. This is because the demands of exploration narrows down as the intelligent cell moves towards the destination. In the following sub-sections, we give a description of the settings of the intelligent cell's input states and output actions.

Input states

Representing the input state accurately and efficiently is a key issue for the deep reinforcement learning framework of cell movement. Besides the location of the intelligent cell, which is indispensable, an intuitive assumption is that its neighbors, which represent the environment, should be incorporated to form the input state. We implemented a neighbor determination model (which takes a set of features of two cells, such as the distance between them, their radii, etc., and determines whether they are neighbors with each other with machine learning algorithms) [191] in a conservative manner for this

purpose. Specifically, we extracted a number of candidate cells that might influence the intelligent cell with a relatively loose condition, so that more cells would be selected to guarantee that the input state was sufficiently represented. This was done by running the agent-based model in a non-reinforcement learning mode (all cells move based on the observation data) and recording the neighbors of the intelligent cell at each time step. Finally, we combined the locations of all these cells (selected accumulatively in the whole process) in a fixed order as the input for the neural network.

Output actions

It is intuitive to give the intelligent cell as many candidates of actions as possible (or a continuous action space) so that it can make the most eligible choice during the simulation. The diversity of the action includes different speeds and directions. However, the number of output nodes grows exponentially as we take looser strategies to select the action. Based on our extensive experiments, we discovered that an enumeration of eight directions of action, with 45° between each of them, is good enough for this scenario. Moreover, we fixed the speed based on an estimation of the average movement speed during the embryogenesis, which was measured from the observation data.

Finally, we give an example of a specific evaluation step for a single action selection process (Fig. 6.4). We collect all the locations of the selected cells by the neighbor determination model, concatenate them to form a vector in a fixed order, and feed it into the neural network. The output of the neural network are the Q-values (i.e., a probability for selecting each action). The action that corresponds to the maximal probability (or a random action as the ϵ-greedy suggested) is selected as the intelligent cell's next movement.

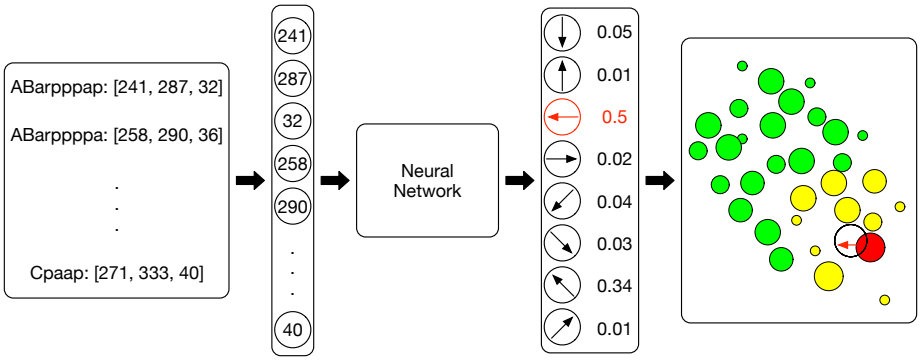


Figure 6.4: An example of a specific evaluation step for a single action. A list of cells are pre-selected as the state cells via the cell neighbor determination model. Their locations are concatenated and sent to the neural network, and the output action with the maximal probability is selected as the intelligent cell's next movement.

6.4 Experiments

6.4.1 Computational environment and platform

The agent-based model was implemented with Mesa, which is an ABM framework in Python 3+. We used Python's GUI package Tkinter for the purpose of visualization. The cell movement behavior model was built with 3D coordinates, and certain slice of the whole embryo was visualized in a 2D manner to illustrate where emergent behaviors specifically happen. We used Pytorch to achieve reinforcement learning algorithms with the advantage of GPU acceleration during the training process. The reinforcement learning architecture was integrated as part of the agent-based model. All the computations were executed in a DELL® Precision workstation, configured with a 3.6 GHz 4-core Intel® Xeon® CPU, 64 GB main memory, and a 16-GB NVIDIA® Quadro® P5000 GPU.

6.4.2 Model setup

Live 3D time-lapse images of *C. elegans* embryogenesis data were used to study cell movement. Cell lineage [192] was traced by Starrynite II [193] and manually corrected in Acetree [173]. Acetree was also used to visualize the observation data.

Two special *C. elegans* biological phenomena, *the intercalation of Cpaaa* and *left-right asymmetry rearrangement*, were investigated. The first case is a remarkable process during *C. elegans* early morphogenesis of dorsal hypodermis. Cpaaa is born at the dorsal posterior. About 10 minutes later after its birth, Cpaaa moves towards the anterior and intercalates into two branches of ABarp cells, which will give rise to left and right seam cells, respectively. The intercalation of Cpaaa is consistent among wild-type embryos. It leads to the bifurcation of ABarp cells and the correct positioning of seam cells. The second case is *left-right asymmetry rearrangement*. It is a significant development scenario: At the 4-cell stage, the left-right symmetry is broken after the skew of ABa/ABp spindle. The right cell ABpr is positioned more posterior than the left cell ABpl. At the AB64 (64 AB cells, 88 total cells) stage, the movement of ABpl and ABpr cells start to restore the spatial

symmetry, i.e., ABpl cells move towards the posterior and ABpr cells move towards the anterior. By 350-cell stage, ABpl and ABpr cells are again in symmetry on the AP axis. This asymmetry rearrangement achieves a superficially symmetric body plan [194].

The embryo was considered to be an ellipsoid for the volume estimation. The mounting technique aligns the DV axis in the embryo with the z-axis of the data [187, 195], and the lengths of the other two axes (AP and LR) were obtained by finding the minimum and maximum cell positions along them [170]. For the estimation of the cell radius, the ratio of the cell volume to the entire embryo was determined based on its identity. Then, the radius was estimated by considering a cell as a sphere [191].

We utilized linear functions to define the rewards in our simulations. Specifically, for the *Collision* rule, a penalty (negative reward) was exerted as the distance between two cells reached a threshold. As their distance became smaller, the penalty linearly grew until a terminal threshold was reached (Eq. (6.4)). Similarly, for the *Boundary* rule, the penalty was calculated based on the distance between the intelligent cell and the eggshell. Finally, for the *Destination* rule, bigger positive rewards were given as the cell moved towards the destination.

$$r = \frac{d - d_l}{d_h - d_l} \times (r_h - r_l) + r_l, \tag{6.4}$$

where d is the distance between two cells and d_h and d_l represent the highest and lowest bounds of the distance between two cells where a penalty is generated. r_h and r_l indicate the range of the penalty.

6.4.3 An agent-based deep reinforcement learning framework for *C. elegans* embryogenesis

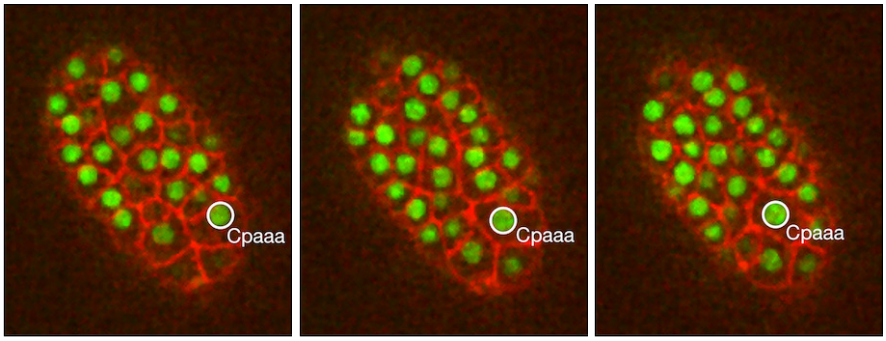
The ABM environment was initialized with the observation data from live imaging with automated cell lineage tracing. We first tested the performance of out ABM framework. The ABM platform was configured to track the movements of the intercalation cell, namely, Cpaaa, for the purpose of illustration. Although the embryo we measured had a length of

30 μm in the dorsal-ventral axis, we only considered the space that is 5-9 μm to the dorsal side, where Cpaaa's intercalation happens. The entire space was visualized by projecting all cells in this space to the center plane (7 μm to the dorsal side). Based on the result (Fig. 6.5) we found that the movement path of Cpaaa is consistent with that in the 3D time-lapse images. The visualized cell sizes are largely consistent with the observation data, except the fact that a few of them, especially located in the planes that are far away from the center plane, have slightly different sizes visually. However, those differences have an insignificant impact on cell movement modeling.

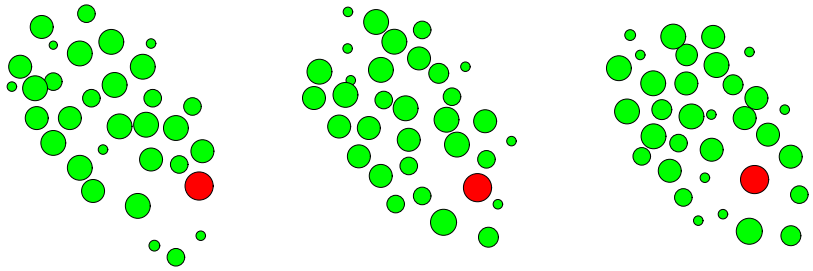
Unlike supervised learning tasks, such as classification and regression, evaluating the performance is quite challenging in deep reinforcement learning tasks. We followed the evaluation metric in [111] to quantify the general performance of the system. The total rewards a cell collects in a single movement path generally goes upward, but tends to be quite noisy since very tiny changes in the weights of the neural network results in large changes in the actions a cell chooses [111] (Fig. 6.6(a)). Training loss tends to oscillate over time (Fig. 6.6(b)), and the reason behind this is the implementation of the experience replay and the target network, which cut off the correlation between training samples. Finally, we extracted a set of states by running the model in a non-reinforcement learning way and collecting the state cells' locations. We then fed these predefined states to the neural network during the training process. It turns out that the average action values of these states grows smoothly during the training process (Fig. 6.6(c)). We did not encounter any divergence problems, though the convergence of DQN is still an active research area. Sometimes, we experienced a few unstable training scenarios, but these problems could be solved by implementing a learning rate decay strategy.

6.4.4 Regulatory mechanisms of individual cell movements

We examined our hypotheses of individual cell movement in the *Cpaaa intercalation* case (see Section 6.2.1). Specifically, we tested (1) whether Cpaaa's intercalation results from an active directional movement or a passive movement, and (2) whether a passive movement

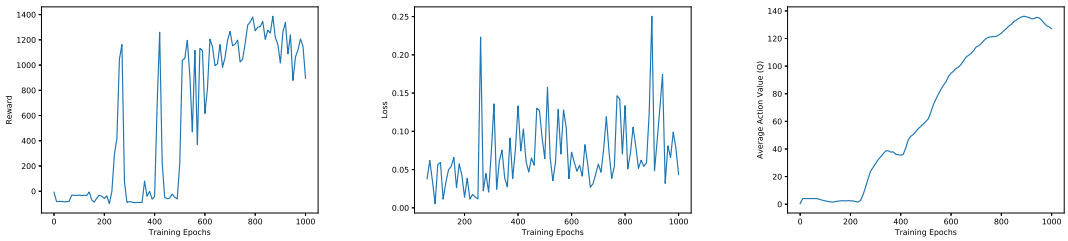


(a)



(b)

Figure 6.5: Comparison between (a) the 3D time-lapse images and (b) the visualizations of the ABM simulation results. Simulation results highly reproduce the observed patterns.



(a) (b) (c)

Figure 6.6: Performance evaluation of the deep reinforcement learning algorithm for cell movement modeling. (a) The accumulated rewards generally goes upward, but tends to be noisy. (b) The loss tends to oscillate because of the implementation of the experience replay and the target network. (c) The average action value grew smoothly over time.

mechanism is sufficient for explaining the migration path of Cpaaa's neighbors. In this case, the observed fact is that during the first four minutes of the process, the intercalating cell Cpaaa moves randomly. After extensive divisions of the ABarp cells, Cpaaa changes its behavior to a directional movement until the end of the process. The signal triggering the switch may come from the newborn ABarp cells.

In the directional cell movement process, unexpected regularization signals or irregular movement patterns have to be considered. In our study, we defined the possibility of selecting a directional movement from the neural network by a ratio between 0 and 1. The value of zero means a completely random movement, and the value of one means a completely directional cell movement.

Regulatory mechanisms in the *Cpaaa intercalation* case

We trained individual neural networks (parameters were initialized by random sampling from a Gaussian distribution.) for directional and passive movements with different sets of regulatory mechanisms. Specifically, we trained the neural network for passive movement with the *Collision* and *Boundary* rules, and the one for directional movement with an addition of the *Destination* rule. The different behaviors of Cpaaa (random movement for the first four minutes and directional movement after that) were controlled by manipulating the probability of random movement ϵ in the action selection procedure. The results of the simulation of Cpaaa with the *Destination* rule (Fig. 6.7(b)) show that during the first four minutes, the intelligent cell didn't have an explicit destination and, to a large extent, acted randomly. After that, Cpaaa switched its behavior and began to move directionally to the destination, as well as kept proper distances from its neighbors and the eggshell. The whole migration path largely reproduced that in the live microscopy images (Fig. 6.7(a)). However, when we trained Cpaaa without the *Destination* rule, it failed to identify the migration path and fell into a suboptimal location where it kept proper distances with its neighbors (Fig. 6.7(c)). We also trained a neighbor of Cpaaa, namely, Caaaa, as a passive movement cell during the process (Fig. 6.7(d)), and its migration path in this scenario also

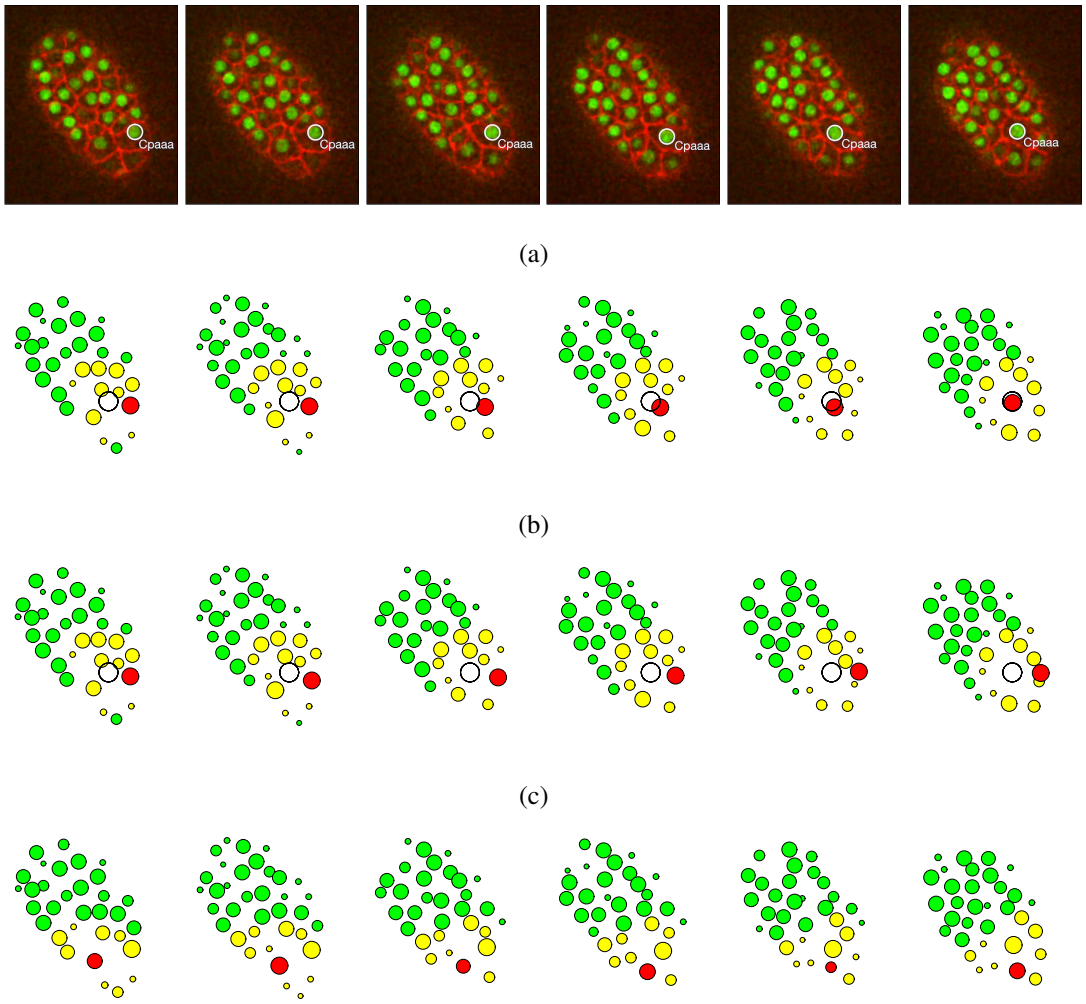


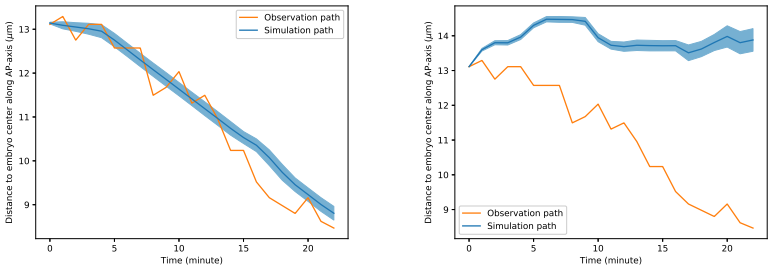
Figure 6.7: Results of the *Cpaaa intercalation* case. (a) Observation results visualized by Acetree from 3D time-lapse images. (b) Simulation results of the intercalating cell Cpaaa with the *Destination* rule. (c) Simulation results when training Cpaaa only with the *Boundary* and *Collision* rules, without the *Destination* rule, which indicate that Cpaaa fell into a suboptimal location. (d) Simulation results of the cell Caaaa, a neighbor of Cpaaa. Red, yellow, and green circles represent the intelligent cell, input state cells, and non-related cells, respectively. The white circle indicates the destination of the intelligent cell. All four sets of data were collected at the following time steps: 0, 4, 8, 12, 17, and 22 (minutes from the beginning of the simulation).

reproduced that in the images, which indicated that Caaaa played a passive role during Cpaaa's intercalation.

For the verification of the generality of the model, random noises were added to the initial positions of all the cells (including the intelligent cell) and to all the migration paths of the dumb cells during the training process. It turns out that the neural networks can still provide the most proper actions under a large variety of input states after the policy converges, though the optimization process takes longer to converge than that in the scenarios without random noises.

Migration path of the intelligent cell

We found that qualitatively, the intelligent cell Cpaaa adopted a similar migration path to the destination with the directional movement setting, as compared to the observation case (Fig. 6.8(a)), though from the 13th to 19th minute, the observation movement of Cpaaa went towards the anterior faster than the simulation path. The difference between the simulation and observation results indicates that extra regulatory mechanisms (such as cell adhesion, or intermediate sub-mechanisms, see the Discussion section) could be considered to control cell movement during the whole *Cpaaa intercalation* process. On the other hand, without the *Destination* rule, Cpaaa's simulated path is quite far away from the observed path (Fig. 6.8(b)). We used the mean square error (MSE) as a quantitative measurement of the simulated path and the observed path. It turns out that the MSE in Fig. 6.8(a) is much smaller than that in Fig. 6.8(b) (4.05 vs. 237.60). In conclusion, the above results showed that Cpaaa's intercalation is regulated by an active directional movement mechanism, which is strongly influenced by the *Destination* rule (or its alternatives), rather than by a passive movement mechanism. Moreover, another interesting finding is that the standard deviation of the migration path of Cpaaa with the *Destination* rule is controlled in a proper range, whereas that of the path without the *Destination* rule diverges as time goes by. Such a result indicates that the intelligent cell achieves an error correction mechanism in its migration path to the destination.



(a) (b)

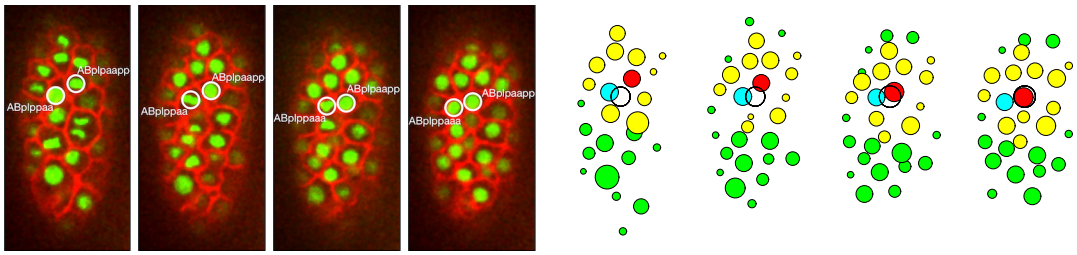
Figure 6.8: (a) Migration paths of Cpaaa with directional movement. (b) Simulation results when training Cpaaa only with the *Boundary* and *Collision* rules, without the *Destination* rule. Results indicate that Cpaaa fell into a suboptimal location. Both simulation paths are the averages over 50 runs, and the shaded regions indicate ranges of one standard deviation greater/less than the average values. The horizontal axis represents the developmental time in minutes. The vertical axis represents the projected position of Cpaaa on the AP-axis to the center of the embryo.

6.4.5 Regulatory mechanisms of group cell migration

In this experiment, we trained the neural network to test the cell movement in group migration via the case of *left-right asymmetry rearrangement*. Rather than explicitly pointing out the destination, we let the intelligent cell (ABplpaapp) follow the leading cell (ABplppaa, or its daughter cells). The reward setting was then modified accordingly: When the distance between the leading cell and the following cell is in a proper range, a positive reward is given. In this case, we did not explicitly point out the leading cell (or its locations) as part of the environment. This is because here we aimed to model cell movement in a specific case and the cells' locations were organized as the input state in a fixed order, so the neural network will gradually learn and know which one is the leading cell (following a leading cell will obtain a big reward). We believe that adding the leading cell as part of the environment will help the model deal with more general and complex cases. The results (Fig. 6.9(b)) show that ABplpaapp always moves following the leading cell, and keeps proper distances from its neighbors. The results are consistent with the observation data (Fig. 6.9(a)), which shows the flexibility of our model by replacing the *Destination* rule with more concrete ones.

6.5 Discussion

In this chapter, we presented a novel approach to model cell movement using a neural network and reinforcement learning within an agent-based modeling framework. Our study showed that neural networks can be adopted to characterize cell movement and that the deep reinforcement learning approach (DQN) can be used to find the optimal migration path of a cell under certain regulatory mechanisms. As comparing to the heuristic rule-based, agent-based models, with which macroscopical behaviors (such as tissue/organ morphogenesis) can be studied [123, 126], this model provides a new point of view in which single cell movements can be defined and optimized over a considerable period of time. In the *Cpaaa intercalation* case, we tested two hypotheses (active directional



(a) (b)

Figure 6.9: The simulation of left-right asymmetry rearrangement. (a) Observation data. The intelligent cell and the leading cell are circled. (b) Simulation results. The cyan circle represents the leading cell, and the others are color coded, as in Fig. 6.7. The white circle here indicates the destination of the intelligent cell only for the purpose of visualization. Both sets of data were collected at the following time steps: 0, 3, 6, and 9 (minutes from the beginning of the simulation).

movement vs. passive movement) that might explain Cpaaa's migration towards the anterior by manipulating the reward settings (use the *Destination* rule or not). Simulation results rejected the passive movement assumption after comparisons between simulated and observed paths of Cpaaa. Such results indicated that target site specification (the *Destination* rule), as a simplified representation of morphogen gradient, is an effective approach for cell migration path learning, especially when regulatory mechanisms lag data collection. The *left-right asymmetry rearrangement* case demonstrated that the framework has the capability to generalize the *Destination* rule to more specific mechanisms (a leader-follower mechanism in this case) to explain certain cell movement behaviors. By comparing simulated cell migration path regulated by the proposed assumptions and the observed path in a reverse engineering perspective, this framework can be used for facilitating new hypotheses during certain developmental processes not only in *C. elegans*, but other in tissues/organisms as well.

This model captures the main aspects of cell movement and provides a new idea that represents cell behaviors with neural networks trained by deep reinforcement learning algorithms. More powerful models can be implemented in the following aspects: (1) Multi-agent reinforcement learning [196, 197] can be used for studying cooperative/competitive cell behaviors by manipulating the rewards in the framework. Such an extension can provide further biological insights. For example, for the *Cpaaa intercalation* case, we may investigate whether the certain group of cells (i.e., Cpaaa and its neighbors) works cooperatively (as a result of the intercalation of Cpaaa) or its neighbors actually act competitively with their own rules (but the regulatory rule of Cpaaa is over-dominant). More specifically, we observed that during the last few minutes of the process, the cell ABarpaapp moves to the posterior to become a neighbor of Cpaaa. It is interesting to study whether ABarpaapp helps Cpaaa to intercalate towards the anterior (cooperative behavior, give both cells rewards when the intercalation of Cpaaa is achieved.), or such a migration of ABarpaapp is just due to its dislocation (competitive behavior, ABarpaapp will not be rewarded when Cpaaa achieves the intercalation.). (2) The hierarchical regulatory mechanism is another area of interest. Although the *Destination* rule provides a

simplified representation of morphogen gradient, it can be generalized with the formation of certain cell neighbor relationships. In the *Cpaaa intercalation* case, the intelligent cell experiences a series of changes of neighbor relationships before reaching the target site. It is worth investigating whether these relationships play as significant sub-goals to serve the ultimate goal. As presented in [108], the deep Q-network performs poorly on hierarchical tasks. Such tasks require more advanced strategies that are obtained by prior knowledge, which can hardly be represented by the input state. Moreover, to investigate the cell-cell interaction mechanism represented with sub-goals, the reward feedback is sparse and delayed, and the search space is extremely large compared to the Destination rule setting. Using a hierarchical model can also boost efficient training. Therefore, future work is immediately needed to implement hierarchical deep reinforcement learning architectures to meet such demands [198]. (3) Other advanced training strategies and reinforcement learning algorithms are also worth investigating to improve the performance of the model, such as learning rate decay [199], continuous control [81], and asynchronous methods [109]. (4) Finally, we hope to incorporate more biological domain knowledge in the model to simulate more complex cell movement behaviors. As one of our previous effort, we have developed a developmental landscape for mutated embryos [153, 154], the mutated cell fate information from this research can be integrated as part of the input state to study a cell's migration path in a mutant. With fate-related adjustments of the regulatory mechanisms and the reward functions behind them, we can verified/rejected the hypotheses of certain cell movement behaviors in a mutant based on the extent of differences between the simulated path and the observed path. Furthermore, by comparing the simulation and observational paths, we can design more biological experiments for follow-up investigations. Other concepts, such as cell-cell adhesion and spatial hindrance, as environmental factors (like the *Collision* and the *Boundary* rule) can also be incorporated to improve the performance of the model.

6.6 Conclusion

In this chapter, we developed a cell movement modeling system by integrating deep reinforcement learning with an ABM framework. Our modeling system can learn a cell's optimal path under certain regulatory mechanisms, and it can also examine hypotheses by comparing the similarities between the simulation cell migration paths and the observation data. These two capabilities, in turn, provide new opportunities to explore large-scale live image data.

Chapter 7

Efficient Cell Migration Modeling with Hierarchical Deep Reinforcement Learning

7.1 Overview

Recent application of deep learning has demonstrated great power in image processing and image analysis in biology and biomedicine in terms of image reconstruction, classification, segmentation, and augmentation [200, 201]. Meanwhile, 3D time-lapse images contain rich information on dynamic cell behaviors such as cell division, cell migration, and collective cell behaviors, which in turn reflect diverse processes of proliferation, differentiation and morphogenesis [202, 203, 57]. In particular, cell-cell interactions produce forces and recognizable features such as movement, shape changes, and spatial configuration of neighboring cells [204]. These features can be explored to infer hidden cell-cell interactions and potentially identify novel biology. The supreme ability of deep learning to capture intricate relationships in features offers a great opportunity to do so. However, it is not yet clear what learning strategies and approaches would be productive.

Deep reinforcement learning (DRL), which formulates the dynamic decision-making problem with a Markov decision process (MDP), has been highly successful in solving dynamic, global optimization problems, such as game-playing [108, 81, 112, 205] and robotic manipulation [114, 206, 207], achieving or surpassing human performance. These problems involve learning to optimize a sequence of actions, where each action is based on the current state of a dynamic environment and the sequence of actions achieves a global optimum [208]. While the time-sequence scheme of DRL is natural to learning dynamic cell behaviors, it typically requires millions of training data and days of computation [209], and its application in bioimage analysis has been limited [200, 201].

Among a variety of DRL approaches, hierarchical deep reinforcement learning (HDRL) [210, 211] emphasizes the use of subgoals, i.e., meaningful intermediate achievements. For example, in a two-level hierarchy, the higher-level module learns to choose the sequence of subgoals to achieve the overall task and the lower-level module learns the sequence of actions to achieve a subgoal [198, 212]. As to the reward, the value functions combine local feedback for the action at each timestep with long-range feedback for achieving subgoals and the ultimate goal. The use of subgoals reduces the search space and the demand for sample size [209] and the optimal solution can be found in an acceptable time.

Here, we exploited HDRL to learn cell behaviors during cell migration from time-lapse images, treating the migrating cell as an agent, the other cells in the images as the environment, and the migration process as a series of intermediate destinations (subgoals). Emergent attributes of the learned migratory behavior such as the choice of subgoals and the associated movement patterns were collected to examine the underlying biology in terms of potential cell-cell interactions and collective cell behaviors. We further hypothesized that after successful learning, the feature extraction component of the policy network in the lower-module of HDRL, which functions to capture salient features of the environment, provides an effective representation of the cell behaviors and cell-cell interactions, and tested this hypothesis with a novel transfer learning experiment whereby the feature extraction component in the lower-level module was transferred into a new

image classifier to recognize additional cases of cell migration driven by the same cellular mechanism.

Using this approach, we examined cell migrations in *C. elegans* embryogenesis [192]. The 3D, time-lapse images contained minimal labeling and annotation of cells where only the position and size of cell nuclei were provided. The reward system consisted of a global feedback (reaching destination in a specified time) and local rules compiled from observational data to guide cell movement actions at each timestep, with the overall goal to arrive at the destination within a specified time but not necessarily copying the observed path. Our method revealed modular organization of cells during cell migration. Subsequent imaging and genetic perturbations confirmed these modules as sequential formation of multicellular rosettes involving the migrating cell and its neighbors, which is a novel mechanism of cell migration that we term sequential rosettes. A new classifier created through transfer learning successfully identified additional cases of cell migration driven by sequential rosettes and distinguished those that are not, suggesting effective representation of cell behaviors in HDRL. Our study demonstrated that HDRL can be used to form informative models of dynamic cell behaviors with simple rules and a small observational dataset, and to discover emergent features of cells and tissues without prior knowledge.

7.2 Design and scheme

7.2.1 Image data and setup for reinforcement learning

The images we studied were 3D, time-lapse images in which cells were labeled with a ubiquitously expressed nuclear marker [187]. The nuclei were segmented and tracked in order to obtain the information of nuclear sizes and positions over time [213, 193, 214].

We used the 3D, time-lapse images to create a reinforcement learning system (see the Methods Section). Specifically, we treated the migrating cell of interest as a learning agent and the other cells in the images as the environment. In order to construct this environment, the positions of the environmental cells over time were cloned from the input image data.

To further enhance the dynamic scenes of the environment, we increased the temporal resolution by 10 fold, interpolating cell positions with small injection of randomness (see the Methods Section). We then annotated elements in the environment that were key for learning, namely a migration destination (a designated cell in the environment), subgoals (cells selected from the environment), and neighbor relationship among cells at every time point (Fig. 7.1(a), see also below).

For the migrating cell, its starting position was copied from the input image data, and the subsequent positions over time were generated by the sequence of movement actions of the agent. For simplicity, the migrating cell moved at a constant speed, which was estimated as the average speed in observational images. The agent learned to choose the direction of movement at each time point.

7.2.2 Reward construction

The rewards in our system consist of long-range and local feedback on cell movement (Fig. 7.1(b)). The long-range feedback includes a global reward for reaching the specified destination, as well as rewards for achieving each subgoal. The local feedback consists of rules that are used to score cell movement at each timestep.

To develop the local rules, we assume that a period of directional movement of the migrating cell is an indication of a directional net force acting on the cell; and that the spatial configuration of cells around the migrating cell reflects the forces. Two models were developed to serve as the local rules.

The first model is termed the Motion Model, which uses a convolutional neural network (CNN) to call directional vs random movement at a given time point. The CNN was trained on observational images that were labeled directional or random after considering the correlation of the velocity vectors in a period of time. In order to focus the learning on features of the local neighborhood, images were cropped to include only the migrating cell and its direct neighbors.

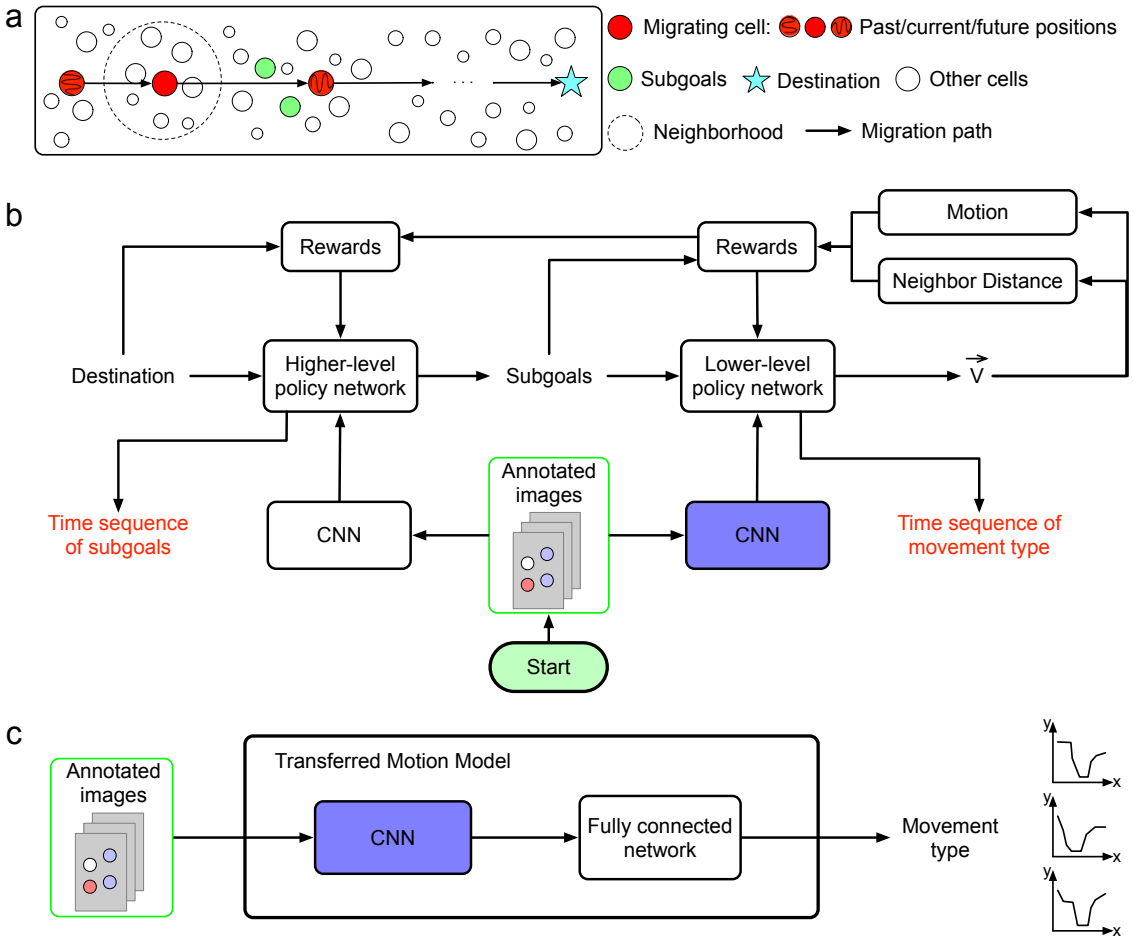


Figure 7.1: Concepts and design to model cell movement with HDRL. (a) A schematic showing actors in cell movement modeling. Small circles represent cell nuclei in a tissue. The migrating cell, subgoals, and other cells are colored red, green, and white, respectively. Dashed circle indicates the current neighborhood of the migrating cell. The migrating cell moves along the arrows towards the destination marked with a cyan star. (b) Architecture and major components of the two-level HDRL used. Arrows indicate data flow between components. Green bounding box indicates the input of the CNNs (feature extraction component). Black bounding boxes indicate model components. Model inputs and outputs are shown as text/symbols without bounding boxes (red indicates output). White and blue CNNs indicate separate networks with their own parameters. (c) Architecture of the TMM. The blue CNN indicates that it is transferred from the CNN of the lower-level module in the HDRL framework in (b).

The second model is termed the Neighbor Distance Model, which considers acceptable minimal distance between neighboring cells. The distribution of such distances was compiled from observational data.

7.2.3 HDRL for model formation

An HDRL with a two-level architecture was used to learn how to guide the migrating cell to reach a given destination in time (Fig. 7.1(b), Table A.1). The lower- and higher-level RL module each contains a policy network, which contains a CNN that serves as the feature extraction component to learn features of the input image series, and a fully-connected network that makes its actions on top of the CNN. Images of the entire embryo were provided as input, allowing the system to learn features beyond the immediate vicinity of the migrating cell. The policy network in the lower-level module uses the image features projected from its CNN to generate suitable movement at each time point (direction of movement with a quantal step size) to reach a given subgoal, while the higher-level policy network uses the image features from its own CNN to select a sequence of subgoals for the lower-level module.

We used the hierarchical Deep Q-Network (h-DQN) [198], which allows for flexible subgoal specification, to integrate value functions based on the local and long-range feedback. Subgoals were chosen from the annotated nuclei that are secondary neighbors of the migrating cell (neighbor of a neighbor). For fast and stable learning, we chose a pair of secondary neighbors at a time as a potential subgoal. A subgoal is considered achieved if both cells become stable neighbors of the migrating cells that last over a certain time period. Similarly, the final destination is considered achieved if the designated destination cell becomes a stable neighbor of the migrating cell (Fig. 7.1(a)). Neighbor relationship among cells is determined by a Neighbor Relationship Model [191] in real-time with a fast random forest classifier that was trained to approximate Voronoi neighbors (see the Methods Section).

To gain more insights on the migration process, we collected all successful time sequences of subgoals (Fig. 7.1(b)), as opposed to the typical single optimal sequence. For each successful sequence, we also recorded the movement types (directional vs random) at each timestep (Fig. 7.1(b)).

7.2.4 Deployment of learned model through transfer learning

We hypothesize that the CNN in the lower-level HDRL module constitutes an effective model that represents the examined cell movement. To test this hypothesis and to exploit the potential model for novel biology, we devised a transfer learning approach to create an image-based classifier of cell movement types. Specifically, the CNN in the lower-level HDRL module was fused to a fully connected network (Fig. 7.1(c)). This classifier, which we termed the Transferred Motion Model (TMM), was trained on labeled observational images of the same cell migration process analyzed by HDRL, with the CNN being fixed to serve the purpose of testing the HDRL-trained CNN. The TMM was then tested on images of other cell migration processes to determine whether or not they use the same underlying mechanism as in the HDRL training case (and therefore share key cellular and tissue features).

7.3 HDRL model formation in C. elegans embryogenesis

We applied our method to examine a cell migration event during *C. elegans* embryogenesis. A cell named Cpaaa is born at the dorsal posterior side of the embryo and migrates anteriorly around 15 minutes after its birth [59, 215]. Cpaaa intercalates in between two rows of cells of the ABarp lineage. The migration ends when Cpaaa becomes the neighbor of the ABarpaapp cell in about 25 minutes (Fig. 7.2(a)). Cpaaa is annotated as the migrating cell, and ABarpaapp the destination.

As the input, embryos were imaged at 1 minute interval, and nuclei were segmented and tracked as described [216]. The Motion Model and the Neighbor Distance Model were

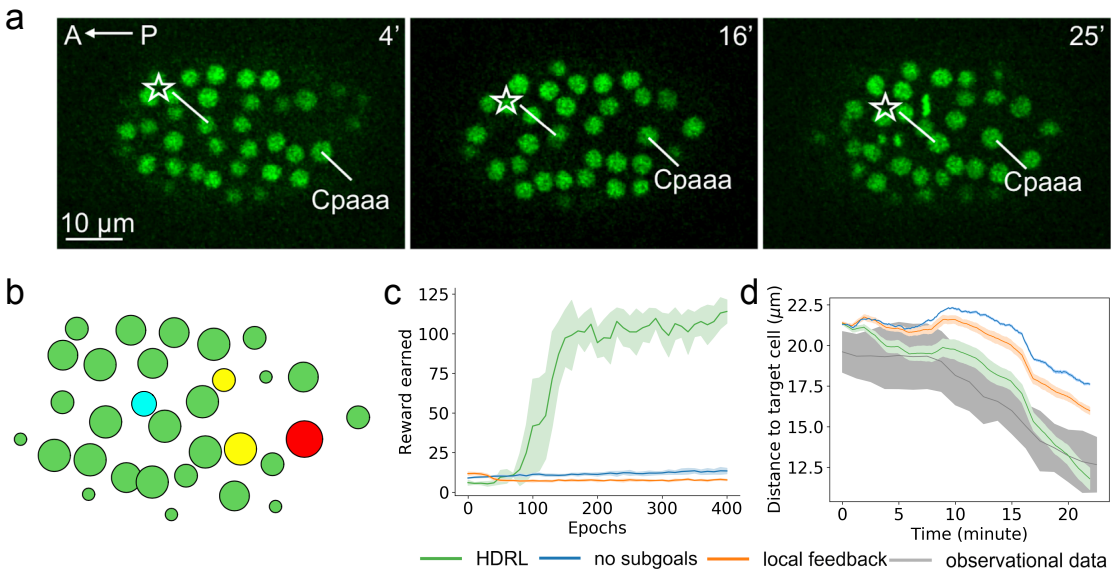


Figure 7.2: Modeling Cpaaa migration in *C. elegans* embryogenesis.(a) Micrographs of *C. elegans* embryo showing the migration of Cpaaa. Dorsal view, anterior to the left (A, anterior; P, posterior). Nuclei in green. Star indicates the ABarpaapp cell. Time 0 is the birth of Cpaaa. (b) The annotated image at a time point during HDRL training. Red, yellow, cyan and green indicate the migrating cell, subgoals, the destination and other cells in the embryo, respectively. (c) The rewards generated over training epochs with different rule settings. "HDRL" indicates HDRL training with the global feedback, subgoals, and local feedback. "No subgoals" indicates DRL training with the global feedback and the local feedback but no subgoals. "Local feedback" indicates training with only the local feedback. The earned rewards were averaged based on five runs and the shaded regions indicate one standard deviation. (d) Position of the migrating cell over time after training with different rule settings. "Observational data" indicates Cpaaa migration in image series. Lines represent the average of each group (20 runs for each rule setting and 10 wild-type embryos for observational data) and shaded regions indicate one standard deviation. Time 0 is 15 minutes after the birth of Cpaaa.

trained on 50 wild-type embryos [170]. For HDRL, the timestep was set at 6 seconds, a 10 fold upsampling from the observational data. The environment was based on the image series of a wild-type embryo. A typical successful scenario lasted around 250-300 timesteps. Through the simulation, HDRL thoroughly examined 120 scenarios (sequences of potential subgoals) and identified a total of 21 successful scenarios.

When both the local and longer-term feedback were used, the migrating cell earned meaningful rewards that guide cell movement through the learning process (Fig. 7.2(b-c)). At the end of learning, the migration path of Cpaaa approximated the path in the observational data (Figure 7.2(d)). In contrast, without using subgoals, the reward did not increase over epochs and at the end of learning Cpaaa failed to move towards the destination, demonstrating the power of using subgoals in HDRL. Similarly, with local feedback alone, the system also failed to learn. While it is not surprising that the simple, common-sense local rules compiled from observational data alone are not sufficient to correctly direct cell movement, combination with the global feedback of destination allow successful learning of cell movement.

7.4 HDRL reveals modular organization of Cpaaa migration

To better understand the successful scenarios of Cpaaa migration, we examined the temporal pattern of directional movement and the set of subgoals.

In terms of the temporal pattern of directional movement, our HDRL reports the Cpaaa movement type (directional as "1" and random as "0") at each timestep. We smoothed the sequence of movement types over a sliding window of 30 timesteps to calculate a value of Movement Index (MI), which better illustrates sustained directional movement (Fig. A.1). The mean MI of the 21 successful scenarios (Fig. 7.3(a)) revealed two phases of directional movement, from minute 0 to 5 and from minute 15 to 22 after Cpaaa starts to migrate.

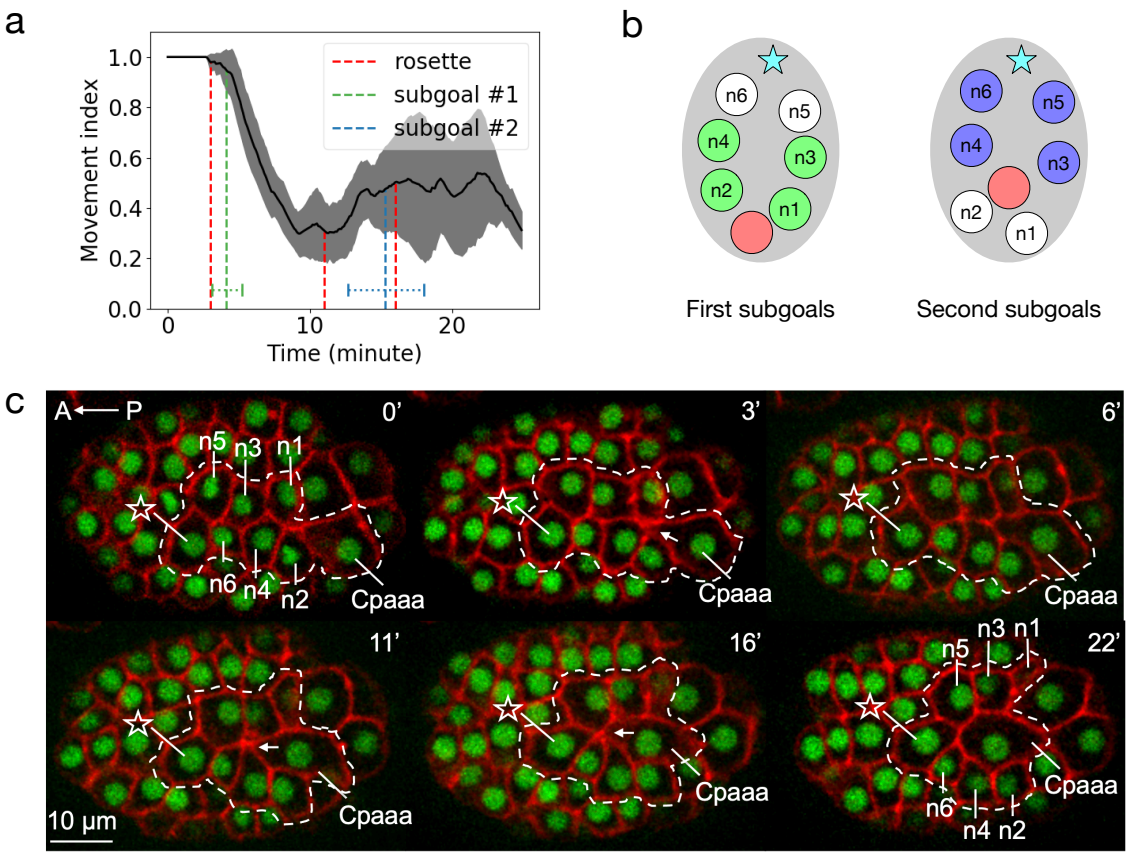


Figure 7.3: Modular organization of Cpaaa movement in HDRL and 3D time-lapse imaging.(a) The MI curve of successful migration scenarios. The average MI (black line) with one standard deviation (shaded region) are shown. Green and blue lines indicate the timing of achieving the first and second subgoal, respectively, where the vertical lines indicate the average time and horizontal line one standard deviation. Red lines indicate the timing of rosette formation identified from imaging experiments shown in (c). Time 0 is 15 minutes after the birth of Cpaaa. (b) The collection of all the subgoals in each of the directional movement phases. The group of subgoals for the first phase includes n1-n4 (green) and the group for the second phase n3-n6 (blue). Red circle, Cpaaa; cyan star, ABarpaapp. (c) Micrographs of *C. elegans* embryo showing sequential rosettes during Cpaaa migration. Dorsal view, anterior to the left (A, anterior; P, posterior). Nuclei in green and cell membranes in red. Time 0 is 15 minute after the birth of Cpaaa. Dashed lines encircle the eight cells involved. Arrows indicate the centers of three rosettes. Star: ABarpaapp; n1: ABarppapp; n2: ABarpppa; n3: ABarppapa; n4: ABarpppap; n5: ABarppaap; n6: ABarpppaa.

The set of subgoals in the 21 successful scenarios also suggests a step-wise organization of movement. Each of the 21 scenarios involved a sequence of two subgoals. All subgoals consisted of the two rows of ABarp cells that Cpaaa migrates through in observational data. For simplicity, these ABarp cells are denoted as n1-n6 (Fig. 7.3(b)). Among the 21 scenarios, the first subgoals involved n1-n4 (green in Fig. 7.3(b)) and the second subgoals n3-n6 (blue in Fig. 7.3(b)). The two subgoals were achieved on average around minute 4.1 and 15.3, respectively, which correspond to the two phases of directional movement (green and blue lines in Fig. 7.3(a)).

7.5 Collective behavior explains organization of cell movement

To better understand the modular organization of the Cpaaa movement, we performed two-color 3D time-lapse imaging of embryogenesis with a broadly expressed cell membrane marker to assess cell shape dynamics on top of the ubiquitously expressed histone marker for cell tracking [216].

The images revealed a striking collective behavior of Cpaaa and its neighboring cells to mediate its migrations (Fig. 7.3(c)). Specifically, Cpaaa forms a sequence of multicellular rosettes with the ABarp cells during its migration (dashed region in Fig. 7.3(c)). Multicellular rosettes are recognized by morphology where 5 or more cells converge to a point [217]. Three rosettes form over time with sequential edge contraction and resolution events (arrowheads in Fig. 7.3(c)). The formation and resolution of each rosette are correlated with Cpaaa movement anteriorly towards the ABarpaapp cell (star in Fig. 7.3(c)), so that when the last rosette resolves Cpaaa forms contact with ABarpaapp and stops migrating. The sequence of rosettes is stereotypical among embryos (n=6 embryos) in terms of the cell composition and timing of each rosette. The first one forms at approximately minute 3 after Cpaaa starts to migrate, which involves Cpaaa and 4 ABarp cells (n1-n4) and resolves around minute 6. The second rosette forms at approximately

minute 11 involving Cpaaa and n3-n6. The third rosette forms at approximately minute 16, involving Cpaaa, n4-n6, and the target cell ABarpaapp, and resolves by minute 22.

While the formation and resolution of multicellular rosettes are known to mediate cell intercalation [217], the sequence of rosettes formed by Cpaaa and the ABarp cells is a distinct form from the known cases. The known cases of rosettes occur in isolation in that a rosette forms and resolves. In contrast, the sequence of rosettes involves overlapping populations of cells and temporally coordinated order of edge contraction and resolution, which indicates an additional level of coordination mechanism. We therefore term this collective behavior as sequential rosettes.

The sequence of three rosettes naturally delineates the Cpaaa movement into three steps that could explain the modular organization revealed by HDRL. The first rosette corresponds to the first phase of directional movement revealed by HDRL in terms of timing (minute 0 to 6 vs. minute 0 to 5) (Fig. 7.3(a)), and the set of subgoals (n1-n4) corresponds to the cell composition of the first rosette (Cpaaa+n1-n4). The second and third rosettes largely overlap in cell composition in terms of the ABarp cells (n3-n6 vs. n4-n6), which correspond to the second subgoals (n3-n6). Timing of the second phase of directional movement in the averaged MI curve (minute 15 to 22) overlaps with the end of the second rosette (minute 11 to 15) and the third rosette (minute 16 to 22). The average MI in the second phase shows increased variance, indicating heterogeneity of paths among individual scenarios. Indeed, the MI curve of individual scenarios (Fig. A.1) shows peaks whose timing corresponds to the second or the third rosette in a subset of scenarios.

7.6 Local cell-cell interactions underlie sequential rosettes

Rosette formation involves a group of neighboring cells. The stereotypical features of the sequential rosettes in terms of cell composition and order of rosettes further indicate a hypothesis where specific cell-cell interactions among appropriate cell types underlie the dynamic behaviors. To test this hypothesis, we examined situations where the fates/types

of the participating cells were perturbed. We analyzed two sets of perturbations where RNAi/knockdown of genes changed the fate in different cells [154].

In the first set, the fate of Cpaaa and/or the ABarp cells were perturbed. Loss of function/RNAi of the pal-1/Caudal gene, which encodes a conserved transcription factor that activates the expression of other genes, abolishes the C fate but does not appear to affect other lineages [154, 218]. In pal-1(RNAi) we found that Cpaaa failed to intercalate into the ABarp cells and stayed where it was born (Fig. 7.4(a)), 6 out of 6 embryos). This phenotype suggests that the C fate is required and rules out the possibility that the ABarp cells are capable of pulling in any neighboring cells. The wwp-1 gene encodes a conserved E3 ubiquitin ligase that marks specific protein targets for degradation. In the early *C. elegans* embryo, wwp-1(RNAi) causes ectopic Notch signaling in the ABarp cell so that the ABarp lineage adopts the fate of the ABalp lineage and produces neurons instead of skin cells, but does not appear to affect the C lineage [154]. In wwp-1(RNAi) Cpaaa also failed to intercalate into the ABarp cells (Fig. 7.4(b), 7 out of 7 embryos). This phenotype suggests that the ABarp fates are required and rules out the possibility that the Cpaaa cell is capable of engaging any neighboring cells for migration. We conclude that the Cpaaa movement requires cell-cell recognition and specific interaction between Cpaaa and the ABarp fates.

In the second set, the Cpaaa and ABarp cells were not affected but an ectopic set of cells with the ABarp fates were produced next to Cpaaa. This situation was achieved by RNAi of the glp-1/Notch gene, which encodes the receptor in the Notch signaling pathway. In glp-1(RNAi) the ABprp cells adopted the ABarp fates (red dashed circle, Fig. 7.4(c), 5 out of 7 embryos) [154]. In 2 out of the 5 embryos where ABprp cells adopted the ABarp fates, Cpaaa migrated normally and intercalated into the ABarp cells. In the remaining 3 embryos, Cpaaa intercalated between the ABarp and ABprp cells. Interestingly, the ABprp cells involved are the corresponding cells as in the wild-type ABarp cells in terms of lineage identity (7.4 legend).

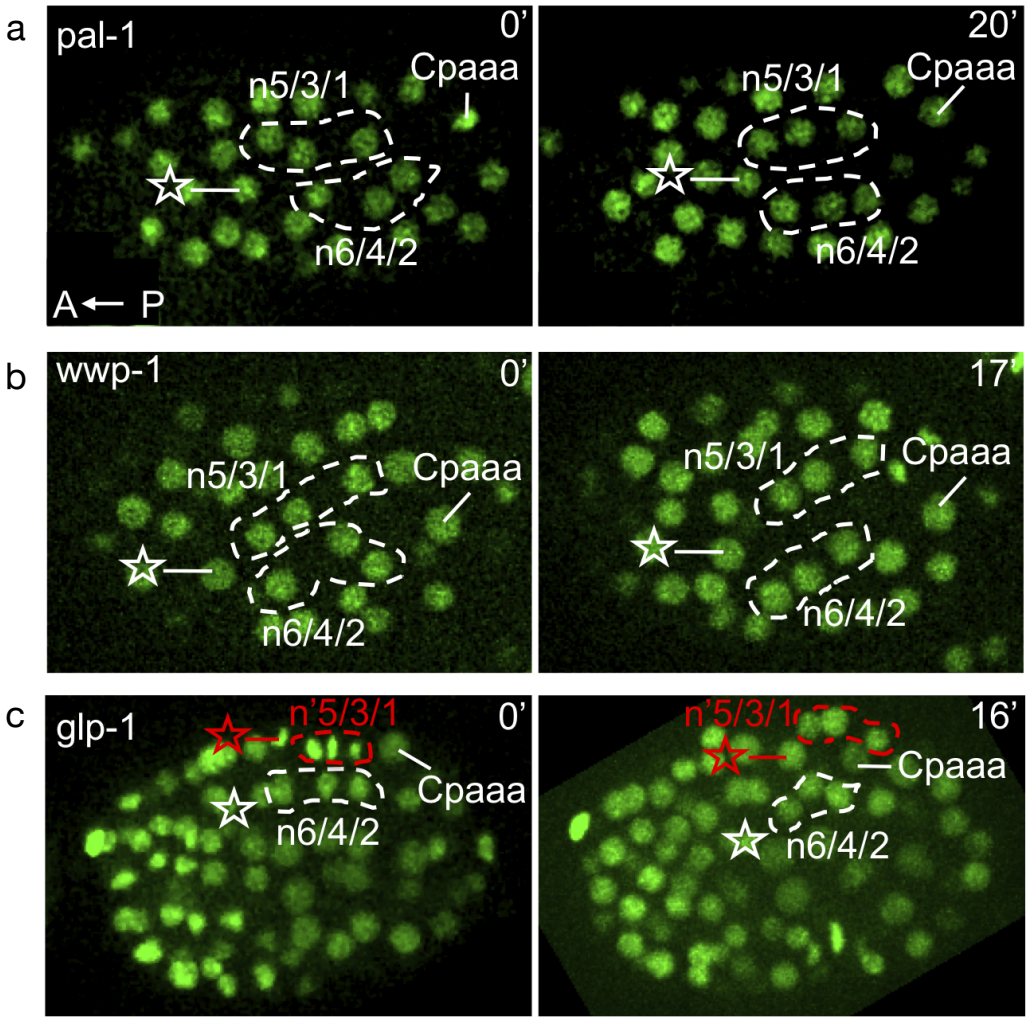


Figure 7.4: Migration of Cpaaa upon genetic perturbations.Micrographs of *C. elegans* embryo. See Fig. 7.2(a) for convention. White star marks the destination in the wild type (ABarpaapp). White dashed circles mark the ABarp cells involved. For n1-6, see Fig. 7.3. Red star, dashed circles and n' mark cells in the ABprp lineage that have adopted the corresponding ABarp fates. Time 0 is 15 minutes after the birth of Cpaaa. (a) A pal-1 (RNAi) embryo. (b) A wwp-1 (RNAi) embryo. (c) A glp-1 (RNAi) embryo. Red star: ABprpaapp; n'1: ABprppapp, n'3: ABprppapa, n'5: ABprppaap,. Note that the wild-type destination (ABarpaapp) is on the dorsal side of the plane shown and the white star indicates the projection of ABarpaapp position on the plane shown. n1/3/5 in the ABarp lineage are also on the dorsal side and not shown.

Taken together, these perturbations support the notion that local interactions among specific cells give rise to the emergent collective cell behaviors of rosette formation and cell migration.

7.7 Transfer learning from HDRL distinguishes movement patterns

The above results established sequential rosettes as a novel mechanism that underlies the migration of Cpaaa. Intriguingly, while HDRL was not aware of this mechanism, the emergent features from HDRL, namely the set of subgoals and phases of directional movement, closely reflects the collective cell behavior of rosette formation. In this section, we further examine to what extent the HDRL-trained CNN provides a model for rosette-based cell migration.

As outlined in Fig. 7.1(c), the CNN of the lower-level HDRL module was connected to a fully-connected neural network in order to create the TMM and classify if a cell movement is based on sequential rosettes or not. The TMM was first trained to classify the movement type of Cpaaa, with the transferred CNN being fixed. The training data was the same set of embryos used to train the Motion Model. The TMM was trained and tested on images of the whole embryo, because the transferred CNN was trained on whole embryo images during HDRL model formation. In contrast, the Motion Model was trained on cropped images that contained just the local neighborhood of the migrating cell. For comparison, it was tested on both cropped and whole images. Directional movement of Cpaaa was labeled as rosette-based movement and random movement of Cpaaa as non rosette-based movement.

The TMM significantly outperformed the Motion Model for classifying Cpaaa movement (Fig. 7.5(a)). With the same training/testing split (see Methods), the average test accuracy of the TMM reached over 90%. The accuracy for the Motion Model, when using cropped images, averaged over 80%. When whole embryo images were used, the accuracy

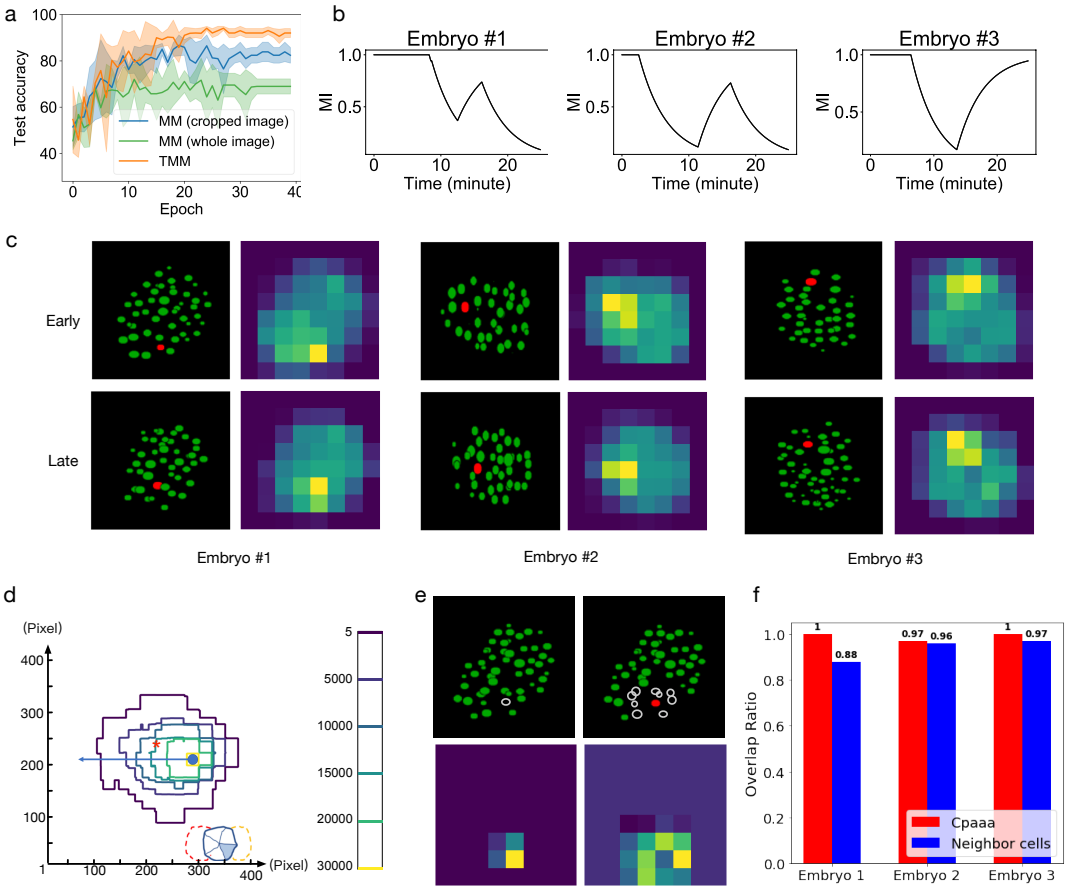


Figure 7.5: Validation and characterization of the TMM.(a) Test accuracy of the TMM and the Motion Model (MM). (b) The MI curve from the TMM in four embryos that were not included in the training set. (c) Example input images and the corresponding summary feature maps of the TMM at an early and a late time point of Cpaaa migration in three embryos. Input images on the left and feature maps on the right. For images, red shows nucleus of Cpaaa and green other cells. For feature maps, colors represent the value at each pixel of the summary map. Warmer color represents higher value. (d) Isocontour representation of the aggregated value of feature maps across embryos and time points. Closed blue circle marks the position of the migrating cell. Red asterisk marks the isocontour. The bottom inset shows a schematic of the migrating cell (shaded blue), current rosette neighbors (white cells in blue circle) as well as the other neighbors of the previous (yellow) and the next (red) rosette. (e) Upper panels: example ablated input image from Embryo 1 after ablating Cpaaa or Cpaaa's neighbors. White open circles marked the positions of the ablated cells. Lower panels: the corresponding summary difference map that shows changes in the summary feature map. (f) The ratio of overlapped area between the effective areas and the ablated cells to the effective area among a total of 74 timesteps in three embryos. See also Fig. A.4 for underlying data.

for the Motion Model dropped to below 70%. These results demonstrate the effectiveness of HDRL-based training of the CNN.

We further tested the TMM on Cpaaa movement using 3 embryos that were not in the training set. Without any further fine-tuning, the TMM successfully revealed the phases of rosette-based movement in the MI plots for each of the 3 embryos (Fig. 7.5(b)).

We then asked how the HDRL-trained CNN may have captured the features of rosette-based cell migration. To this end, we examined the feature maps generated for the 3 embryos used as the test set in Fig. 7.5(b). At each time point, the CNN generated 64 feature maps that were 8x8 (Fig. A.2). We also summed the 64 maps as a summary for the time point (Fig. 7.5(c)). These summary maps as well as many of the individual maps appear to highlight the shape and size of the embryo, the position of the migrating cell, as well as the neighboring cells in the direction of migration regardless of the overall orientation of the embryo. For a more quantitative analysis of these observations, we aligned the summary maps (of representative time points at the early, middle, and late stages of migration in each embryo) by the position of the migrating cell and the migration direction, and up-sampled the 8x8 summary maps to the original resolution of the input microscopic images. We further aggregated the upsampled maps of the 3 embryos and created an isocontour plot to summarize the activation levels relative to the migrating cell (Fig. 7.5(d)). Notably, an isocontour encircles approximately 150×100 pixels around the migrating cell (red asterisk in Fig. 7.5(d)), which is spatially correlated to an area containing the migrating cell and its rosette neighbors, as well as the neighbors participating in the previous and the next rosette, respectively (bottom inset in Fig. 7.5(d)). Thus, the highest level of activation among the feature maps is correlated to the position of the migrating cell and its neighbors.

To further test if the CNN responds to the migrating cell and its neighbors, we conducted two ablation experiments. Specifically, we removed the migrating cell (Cpaaa) or its neighbor cells from the input images and examined the changes in the feature maps. Each embryo at each time point produced 64 feature maps and 64 difference maps between

the corresponding feature maps from the original and the ablated input images (Fig. A.3). The 64 difference maps at each timestep were summed into a summary difference map for further analysis (Fig. 7.5(e)). We focused on the pixels in a summary difference map with significant differences (see the Methods Section), which we refer to as the effective area, and asked how well the effective areas overlap with the ablated cells spatially. Across a total of 74 timesteps in 3 embryos, 97-100% of the pixels in the effective areas overlaps with Cpaaa when Cpaaa was ablated and 88-97% of pixels in the effective area overlaps with the neighbor cells when neighbor cells were ablated (Fig. 7.5(f) and Fig. A.4). These results show that the CNN responds to the migrating cell and its neighbors and that these cells contribute largely locally to the feature maps in their corresponding spatial area.

Finally, to test if the HDRL-trained CNN provides an effective model for rosette-based cell migration, we applied the TMM, which was trained on the Cpaaa cell, to classify other long-range cell migrations in *C. elegans* embryogenesis. Based on the documentation of long-range migrations in the literature [192], we focused on two cells with the largest migration distance, namely mu_int_R and CANL. Both of these migrations occur approximately 3.5 hours later than Cpaaa (500-cell stage vs. 150-cell stage) (Fig 7.6(a-b)), with a smaller average cell size and a higher cell density. The global direction of these migrations is from the anterior to posterior, opposite to that of Cpaaa. As such, these migrations present different embryonic and cellular characteristics than Cpaaa.

The MI curves generated by the TMM showed different modes of movement for these two cells. For mu_int_R (Fig. 7.6(c), n=2 embryos), the MI curve predicted multiple phases of rosette-based movement. In contrast, the movements of CANL were not recognized as rosette-based (Fig. 7.6(d), n=2 embryos), indicating different underlying features/mechanisms than those of Cpaaa.

Subsequent imaging with a cell membrane marker (Fig. 7.6(e)) revealed that mu_int_R migration is indeed mediated by sequential rosettes, while CANL is not. Specifically, for mu_int_R we found a sequence of five rosettes during its migration, occurring at approximately 34, 40, 56, 66, and 74 minutes after the birth of mu_int_R. The timing of these rosettes corresponds to the phases of high MI (red lines in Fig. 7.6(c)). On the other

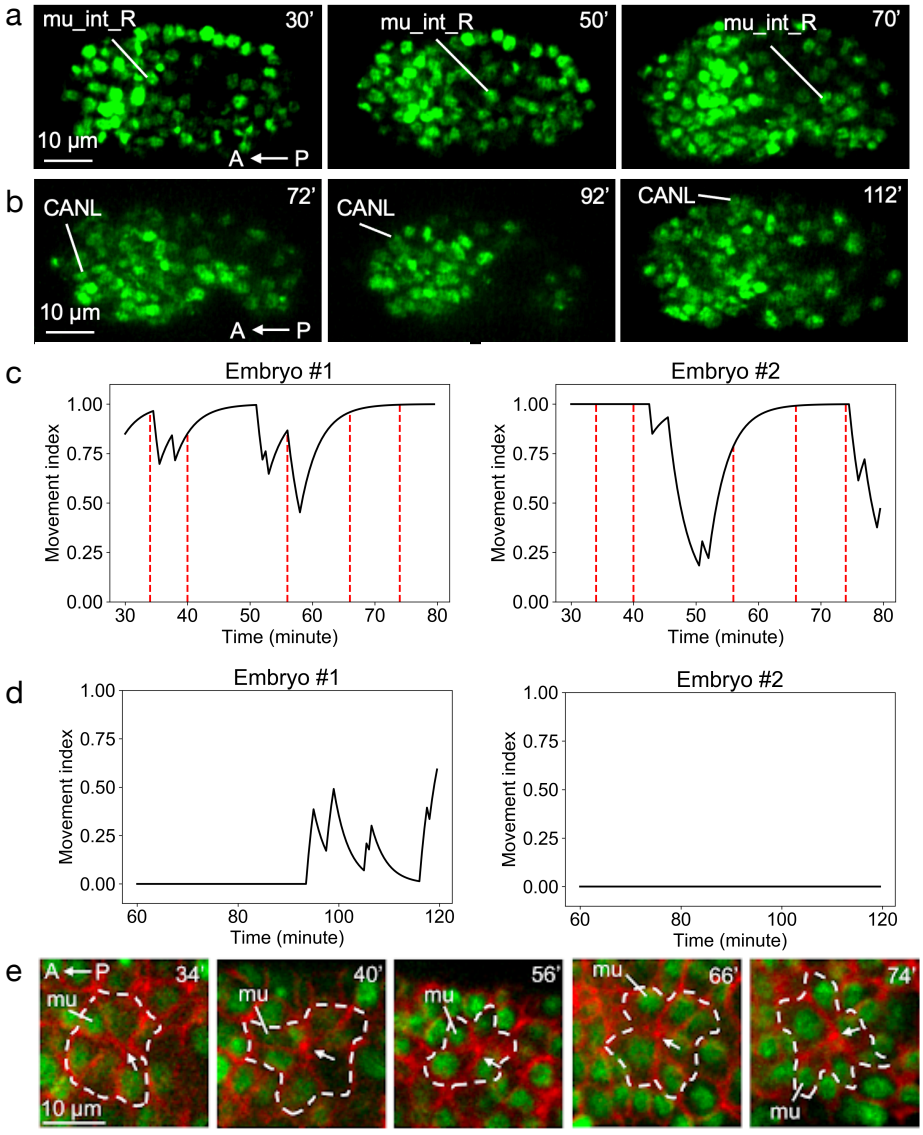


Figure 7.6: TMM classification and 3D time-lapse imaging of mu_int_R and CANL migration. (a,b) Micrographs of *C. elegans* embryo showing the migration of mu_int_R (a) and CANL (b). See Fig. 7.2(a) for convention. Time 0 is the birth of mu_int_R and CANL, respectively. (c,d) The MI curve from the TMM over mu_int_R (c) and CANL (d) migration in two embryos. Time 0 is the birth of mu_int_R and CANL, respectively. Red lines indicate the timing of rosette formation identified from imaging experiments shown in (e). (e) Micrograph of *C. elegans* embryo showing sequential rosettes during mu_int_R (marked by "mu") migration. Dorsal view, anterior to the left. Nuclei in green and cell membranes in red. The dash lines show the contour of rosettes. The arrows indicate rosette centers. Time 0 is the birth of mu_int_R.

hand, we did not find rosettes during CANL migration. Previous studies imaging CANL migration suggested that CANL migrates by generating lamellipodia at the leading edge as in the canonical mechanism of cell migration [219]. These results, in which the TMM is capable of distinguishing migration driven by sequential rosettes from those driven by other mechanisms, suggest that the HDRL-trained CNN provides an effective model for rosette-based cell migration.

7.8 Discussion

Our study presents a data-driven approach to use deep learning to uncover novel biology from images. Essentially, we showed that DRL can be used to form models of unknown cellular behaviors and inspire experimental investigations. Ultimately, our study revealed a previously unknown mechanism of cell migration, which we termed sequential rosettes.

In order for DRL to form a model of a dynamic cell behavior without prior knowledge, we used CNNs as the feature extraction component of the policy networks to examine the images for the environment of the migrating cell. As demonstrated, after learning the CNN in the lower-level module successfully represented the underlying collective cell behavior: in additional cases of cell migration that it had not seen, the model was able to successfully distinguish sequential rosette-based migration from others. While it is technically difficult to explain how the neural network may encode the cell behaviors, high activation in the feature maps appears to be correlated with past, current, and future neighboring cells that form rosettes with the migrating cell. Furthermore, as shown in the test cases, this model seems insensitive to the orientation and scale of rosettes relative to the images despite limited training data, which speaks to the advantages of CNNs for image representation.

By focusing on model formation, our study emphasizes a different perspective to DRL. Conventionally, DRL is used as a generative model to perform new tasks of the same or related nature, be it game playing or robotic manipulation. However, in biological experiments including imaging, the typical challenge is post hoc interpretation of the observations, where the first question is whether an observation can be readily explained by

known mechanisms. In such a setting, classifiers are in demand. In this regard, we showed that after DRL training the feature extraction component in the policy network can be directly transferred to create a classifier. This approach essentially treats DRL as a reward-guided, unsupervised training platform for model formation. To our best knowledge, it is the first time that the feature extraction component of the policy network of DRL is used for transfer learning to perform other tasks.

Our study also demonstrated HDRL as a powerful form of DRL for biology. HDRL showed superior capability of learning and model formation with a small training set, minimal labeling, and simple common sense rules/constraints. In theory, with large training data and practically unlimited computing power for simulation, DRL is capable of learning complex processes without the greedy approach in HDRL to reduce search space. However, such results are often not achievable. For example, in a conceptually similar problem of maze navigation, the model to represent the current position needed to be trained separately with supervised learning in order to achieve a multi-scale representation of space and successful navigation [220]. Alternatively, a greedy reward imitating a molecular gradient of guidance was used to achieve long-range cell migration [59]. In contrast, in this study, HDRL achieved unsupervised model formation without a strong assumption of the underlying biological mechanism, which meets the typical situation of biological data analysis, that is, with partial knowledge and potential for novel biology.

As such HDRL may be used broadly to study different dynamic biological processes and serve different biological questions. For example, with appropriate markers/reporters for imaging, one could examine proliferation patterns, gene expression dynamics, or neuronal activities. However, each problem would require one to carefully define the space/dimensions of the dynamic process (e.g., reporter level on top of (x,y,z,t)) and select biologically meaningful subgoals. More broadly, modeling a dynamic biological process as a sequence of actions in a multi-dimensional space is not limited to image-based data. However, it may not be obvious what the right type of subgoals is for every question. Furthermore, in terms of data, while HDRL reduces the amount of data needed, it may still not be trivial to label the amount of data required. For more complex questions it may still

not be practical to obtain enough data experimentally. Nevertheless, as demonstrated in our study HDRL offers an intriguing approach to exploit deep learning for cell behaviors and dynamic biological processes.

7.9 Methods

7.9.1 Observational dataset and annotation

The observational data in our modeling system are 3D, time-lapse images in which cells are labeled with a ubiquitous nuclear marker. The size and location of each nucleus over time were extracted based on segmentation and tracking of the nuclei [213, 193, 214]. For RL, we used binary images to present the information: each nucleus was represented by a sphere of specified position and size (circular discs on different z planes). The binary images were further annotated with different colors to represent key information: a migrating cell of interest (red), a migration destination (cyan), cells selected as the subgoal (yellow) and all other cells (green). A stack of five planes centered on the migrating cell was used as the input images to the different components in our system.

7.9.2 Reinforcement learning system setup

Time interval in RL was set at 1/10 of that in the observational data. The locations of the environmental cells were derived from observational data with a 10-fold upsampling of temporal resolution and linear interpolation of cell positions. For each interpolated position, a small randomness n_l was ingested based on a normal distribution $n_l \sim \mathcal{N}(0, 0.5)$ with the average value and standard deviation set to 0 and 0.5 pixels. The migrating cell (RL agent) was designed to move at an average speed v (obtained from the observational data) in one of possible directions in a discrete 3D space. To enhance the robustness of RL agent movement decisions, a small randomness n_v was ingested to the speed based on a normal distribution whose standard deviation was set to 10% of the average speed $n_v \sim \mathcal{N}(0, 0.1v)$.

7.9.3 Neighbor relationship model

The Neighbor Relationship Model $f_n = \{0, 1\}$ determines whether two cells are neighbors of each other based on the Voronoi diagram using the center of each nucleus. The Voronoi neighbor relationship is approximated based on a set of criteria as described in the previous work [191]. Denote c_a and c_b the feature vectors of cell a and b, then $f_n(c_a, c_b) = 1$ if a and b are neighbors otherwise 0. A random forest classifier was trained as the Neighbor Relationship Model based on over 940,000 true Voronoi neighbors/non-neighbors during *C. elegans* embryogenesis. This model achieved real-time classification during HDRL, processing a pair of cells in approximately 0.0002 seconds with 99.6% accuracy.

7.9.4 Motion model

The Motion Model was implemented to classify the movement type (directional vs. random) of the migrating cell at a given moment.

Dataset. The dataset to train the Motion Model was established by manually labeling 50 wild-type C. elegans embryos. A time window covering the lifespan of the Cpaaa cell in each embryo, approximately 25 time points per embryo, were manually labelled by authors, using Acetree to observe the movement pattern of the migrating cell. The correlation between the movement direction of the migrating cell in 5 timesteps was assessed. A time point was labeled as 1 for directional movement and 0 for random movement. As the input of the neural network, the labeled images x, centered at the migrating cell, were cropped to include its neighbors (with a size of $x \in \mathbb{R}^{(128\times128)}$). 85% of the labeled data was used as the training set and the rest as the test set.

Neural network architecture. An AlexNet style CNN f_m was used as the classifier, which consists of five convolutional layers followed by a ReLU activation layer after each. A maxpooling layer was implemented for the purpose of downsampling after the first, second, and fifth convolutional layers. The convolutional layers were followed with three fully connected layers and the output of the neural network was a binary call on the two movement types $f_m(x) \in \{0, 1\}$.

Training strategy. The neural network converged (training loss and accuracy) around 40 epochs, with an Adam optimizer, a mini-batch size of 10, and a learning rate of 0.003.

7.9.5 Neighbor distance model

The Neighbor Distance Model was designed to evaluate the likelihood of a spatial distribution among a given group of cells. We used the level of pairwise cell overlap as the basis of the evaluation. Specifically, the level of overlap C between a pair of cells was calculated as the ratio between the distance of the two cells and the sum of their radii. The radii of cells were estimated based on the embryo volume, the total number of cells in the embryo, and each cell's lineage identity as previously described [191].

Ground truth of cell spatial distribution. The ground truth of the cell overlapping levels was collected from 50 wild-type embryos. We computed a probability density function of overlapping levels (Fig. A.5) and found that all the overlapping values were between $\alpha = 0.3$ and $\beta = 0.8$.

Spatial distribution evaluation. At a given simulation step, we evaluated the spatial distribution among the migrating cell and its neighbors by evaluating the level of overlapping between the migrating cell and each of its neighbors. A pairwise overlapping level less than α was considered "completely acceptable" and given a reward of 0. A pairwise overlap level greater than β was considered "completely unacceptable" and given a reward of $-\infty$. For an overlapping level between α and β, the reward r_n^i was set as a negative value of the cumulative distribution function of the overlapping value between cell i and the migrating cell $F_C(c) = P(C \leq c)$ from the ground truth (Fig. A.5), leading to a reward ranging from $[0, -1)$:

$$r_n = \begin{cases} 0 & c \leq \alpha \\ -F_C(c) & \alpha \leq c \leq \beta \\ -\infty & \beta \leq c \end{cases}$$

7.9.6 HDRL model architecture and parameters

We developed a two-level HDRL model using h-DQN [198].

Higher-level module

Network architecture. The higher-level module contains a policy network f_p^h and a CNN f_f^h. The policy network is a fully connected layer, which takes the feature vectors from the CNN to select the current subgoal $f_p^h(v_a^h) = i \in \{0, 1, \cdots, N_s\}$, where N_s is the total number of subgoal candidates. The CNN contains two convolutional layers (see Table A.1), which extracts feature vectors v_a^h from the annotated images x_a^h of the current time point $f_f^h(x_a^h) = v_a^h$.

Input and output. The higher-level module took the annotated image stacks as the input. The size of the entire embryo is around 250-300 pixels. We resized the input images to 128x128. The neural network produced a pair of cells as the potential subgoal, which was sent to the lower-level module. The subgoals were selected from a candidate pool S that contains the secondary neighbors of the migrating cell (neighbor of a neighbor), which were determined by the Neighbor Relationship Model f_n. $S = \{(x, y) : f_n(c_m, c_i) = 1, f_n(c_i, c_z) = 1, f_n(c_m, c_x) \times f_n(c_m, c_y) = 0,$ for all i and $z = (x$ or $y)\}$, where c_m is the feature vector of the migrating cell (m) and $m \neq i \neq x \neq y$. c_i is the feature vectors of the first neighbor cells (i). $c_{z=(x \text{ or } y)}$ is the feature vectors of the cells (z) that is the neighbor of the first neighbor cells (i). The number of subgoals was not predefined in our study, however, in our implementation, the total number of subgoals in a scenario cannot go beyond 3 because of the secondary neighbor rule during the training.

Rewards. The rewards for achieving a subgoal and the final destination were set to 10 and 100, respectively.

Hyperparameters. The network was trained with an Adam optimizer, a batch size of 128, and a learning rate of 0.0001. A replay buffer with a size of 128 was used and training samples were stored and randomly sampled from the buffer to train the network. Similar to the previous study [108], a target network was used to stabilize the training process and

its parameters were copied from the network of the higher-level module every 20 epochs. The reward discount factor γ was set to 0.8 and the ϵ-greedy factor was set to 0.95. The network was trained for 300 epochs.

Lower-level module

Network architecture. The CNN f_f^l in the lower-level module has the same architecture as in the higher-level module. The policy network f_p^l is a fully connected layer, which takes the feature vectors v_a^l from the CNN to select the current atomic movement action $f_f^l(x_a^l) = v_a^l, f_p^l(v_a^l) = i \in \{0, 1, \cdots, N_a\}$, where x_a^l is the input images stacks and N_a is the total number of atomic movement actions.

Input and output. The lower-level module took the annotated image stacks, as described in the higher-level module, with the subgoal from the higher-level module marked with yellow as the input. The output of the module was the atomic movement action from one of the eight directions of action in the XY-plane with 45 degrees between each of them. The quantal step size of the movement was fixed to the average value of step size of the migrating cell and its neighbors during the migration process with the observational data.

Rewards. The lower-level module took rewards from both the local and the long-range feedback. The local feedback consisted of two parts: (1) a reward R_N according to the distribution of the normalized distance between the migrating cell and its neighbors outputted by the Neighbor Distance Model (see Neighbor Distance Model), and (2) a reward R_M, which equals to 1 if the current situation is classified as a directional movement by the Motion Model otherwise 0. For the long-range feedback, when the subgoal is achieved, i.e., the migrating cell becomes a stable neighbor of the subgoal cells for 15 timesteps, a reward R_L, which equals to 10 was given. The total reward R is represented as $R = R_n + R_M + R_L$.

Hyperparameters. The network was trained with an Adam optimizer, a batch size of 32, and a learning rate of 0.00002. The replay buffer contained 4000 samples and the target

network was updated every 1000 iterations. The reward discount factor γ was set to 0.98 and the ϵ-greedy factor was set to 0.9.

7.9.7 Transferred motion model

The Transferred Motion Model $f_t = \{0, 1\}$ was designed to classify the movement type (rosette-based vs. non-rosette-based movements) discovered in the Cpaaa migration. It contained a fully-connected neural network and the CNN, transferred from the trained lower-level HDRL module f_f^l.

Dataset. The image stacks used to train the TMM, the labeling and the training/test data split strategies are the same as those in the Motion Model. The input of the TMM is the whole embryonic image stack (see Higher-level module's input) rather than the cropped images (see Motion Model's input).

Neural network architecture. The TMM consists of two parts: (1) two convolutional layers transferred from the lower-level HDRL module with the trained parameters, and (2) three fully connected layers following the convolutional layers with randomly initialized weights.

Training strategy. During the training of TMM, the weights in the convolutional layers were frozen and only the weights of the fully connected layers were allowed to update. The training loss and accuracy of the TMM converged around 40 epochs, with an Adam optimizer, a mini-batch size of 10, and a learning rate of 0.0001.

7.9.8 Analysis of feature maps

Ablation experiments were performed as follows.

Ablated input image. An ablated input image $x_a \in \mathbb{R}^{(128\times128)}$ was generated by removing the cell(s) of interest (Cpaaa or its neighbor cells) during the input image generating process. All the ablated Cpaaa's neighbor cells were identified by the neighbor relationship model.

Summary feature/difference maps. All the feature maps $F_i \in \mathbb{R}^{(8\times8)}$, $i = 1, 2, \cdots, N_i$ (where N_i is the total number of feature maps) for ablation experiments were generated from the CNN in TMM (see Main text) on observational timesteps. At each timestep, we summed all 64 individual feature/difference maps to a summary feature/difference map $F = \sum_{i=1}^{64} F_i$ (Fig. 7.5(e)).

Effective area. At a given timestep of an embryo, all pixels of the summary difference maps were classified into two categories according to the intensity using K-means unsupervised clustering with $K = 2$. Pixels in the cluster with the higher intensity are referred as the effective area. The number of pixels in the effective area of the summary difference map at each timestep in three embryos were also recorded (Fig. A.4).

Ablated cell area. Ablated cell area was defined as the area in feature maps that spatially corresponds to the ablated cell(s) at a given timestep of an embryo. It was calculated from ablated cell's location and its radii in the input image (see Neighbor Distance Model). Specifically, four bounding box locations (x, y) of each ablated cell in the input image were first downscaled by multiplying $(8/128, 8/128)$ and then rounded it down to get the mapped pixel(s) in the feature maps. Ablated cell area was obtained by aggregating all the mapped pixels of ablated cells at a given timestep of an embryo.

7.9.9 Microscopy, cell tracking and visualization

C. elegans culture, microscopy, and cell lineage tracing were performed as previously described [216]. *C. elegans* strains used in this study: BV24 (ltIs44 [pie-1p::mCherry::PH (PLC1delta1) + unc-119(+)] zuIs178 [his-72p (1kb)::his-72::SRPVAT::GFP::his-72 3UTR + unc-119(+)] V) (for Cpaaa); and DCR4318 (olaex2540 [Punc-33_PHD_GFP_unc54, Punc-122_RFP]; ujIs113) (for mu_int_R and CANL). The movies of movement of Cpaaa, mu_int_R, and CANL were generated by the WormGUIDES software [166] from cell tracking of a real embryo, which facilitates the visualization of spatiotemporal dynamics of selected cells in *C. elegans* embryos using 3D rendering.

7.9.10 Characterization of Cpaaa migration paths

The Cpaaa migration paths were represented as a timeplot of the distance between Cpaaa and the target cell (ABarpaapp). The migration paths among a given group of embryos were represented as a timeplot of the average migration path with one standard deviation. For paths in the observational data, a temporal alignment was applied in order to minimize the temporal variation of cell birth and migration events among embryos. Specifically, the middle point between the maximal and minimal distance in each migration path was used to identify a reference time point. All migration paths were aligned by the reference time point before calculating the average path and standard deviation.

7.10 Conclusion

In this chapter, we used hierarchical deep reinforcement learning (HDRL), known for multiscale learning and data efficiency, to examine cell migrations based on images with ubiquitous nuclear label and simple rules formulated from empirical statistics of the images. We applied the proposed framework to *C. elegans* embryogenesis and HDRL reveals a multi-phase, modular organization of cell movement. Imaging with additional cellular markers confirms the modular organization as a novel migration mechanism, which we term sequential rosettes. Furthermore, HDRL forms a transferable model that successfully differentiates sequential rosettes-based migration from others. Our study demonstrates a powerful and efficient approach to infer the underlying biology from time-lapse imaging without prior knowledge.

Chapter 8

Discussion, Conclusion, and Future Work

In this dissertation, we introduce (1) a new network pruning approach from the perspective of structural redundancy reduction to improve the performance of compact neural networks, (2) a knowledge distillation approach when the training samples are not accessible, and the black-box pre-trained teacher only returns hard labels, and (3) how to use hierarchical deep reinforcement learning for efficient modeling of cell migration. We believe that these work provides inspirations on how to design efficient models on various kinds of tasks.

There are some future work that can be investigated to further improve the proposed approaches.

Network pruning. In our network pruning approach, the graph establishment is an important part. Now the the proposed approach relies on an manual selection of the threshold to determine the edge connections in the graph, which is not flexible. An automatic approach can be investigated to search the proper threshold that can build proper graphs via some search algorithms. Further more, it is also worth evaluating the proposed approach on more complicated tasks, such as object detection and image synthesis.

Knowledge distillation. We introduced KD from a decision-based black-box teacher for the first time. Our study motivated a new line of research on KD, in which the black-box teacher only returns top-1 classes. It is a much more challenging scenario because the class probabilities of the training samples need to be constructed by iteratively querying from the DB3 teacher. With the training set accessible, our DB3KD achieved competitive performance on FLOWERS102, in which samples largely overlap with ImageNet. We believe that DB3KD can work effectively on large-scale datasets. With the training samples not available, like most of the existing works, a large amount of computing resource is required for pseudo sample generation, making zero-shot KD hard to accomplish with large-scale datasets. With a DB3 teacher, even more iterations are needed compared to learning from a white-box model. Although we proposed the first principled solution, we hope it helps to raise attention in this area and promote efficient approaches.

Cell migration modeling with deep reinforcement learning. Our study has shown that deep reinforcement learning as a powerful tool to model cell migration procedures. In our study, a single cell is trained to learning its optimal migration path under pre-defined regulatory mechanisms. Since the developmental process contains many cells in the system, it is worth investigating training multiple cells at the same time to discover there collaborative behavior patterns with multi-agent reinforcement learning.

Bibliography

[1] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012. 1, 59, 74

[2] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016. 1, 32, 59

[3] Jie Hu, Li Shen, and Gang Sun. Squeeze-and-excitation networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7132–7141, 2018. 1

[4] Zi Wang, Chengcheng Li, Huiru Shao, and Jiande Sun. Eye recognition with mixed convolutional and residual network (micore-net). *IEEE Access*, 6:17905–17912, 2018. 1

[5] Ross Girshick. Fast r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 1440–1448, 2015. 1

[6] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems*, pages 91–99, 2015. 1

[7] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 779–788, 2016. 1

[8] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014. 1, 8

[9] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434*, 2015. 1

[10] Chengcheng Li, Zi Wang, and Hairong Qi. Fast-converging conditional generative adversarial networks for image synthesis. In *2018 25th IEEE International Conference on Image Processing (ICIP)*, pages 2132–2136. IEEE, 2018. 1

[11] Zi Wang. Learning fast converging, effective conditional generative adversarial networks with a mirrored auxiliary classifier. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pages 2566–2575, 2021. 1

[12] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3431–3440, 2015. 1

[13] Jifeng Dai, Kaiming He, and Jian Sun. Instance-aware semantic segmentation via multi-task network cascades. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3150–3158, 2016. 1

[14] Guosheng Lin, Anton Milan, Chunhua Shen, and Ian Reid. Refinenet: Multi-path refinement networks for high-resolution semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1925–1934, 2017. 1

[15] Howard Lee and Yi-Ping Phoebe Chen. Image based computer aided diagnosis system for cancer detection. *Expert Systems with Applications*, 42(12):5356–5365, 2015. 1

[16] Pranav Rajpurkar, Jeremy Irvin, Kaylie Zhu, Brandon Yang, Hershel Mehta, Tony Duan, Daisy Ding, Aarti Bagul, Curtis Langlotz, Katie Shpanskaya, et al. Chexnet: Radiologist-level pneumonia detection on chest x-rays with deep learning. *arXiv preprint arXiv:1711.05225*, 2017. 1

[17] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014. 1

[18] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015. 1

[19] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4700–4708, 2017. 1

[20] Ningning Ma, Xiangyu Zhang, Hai-Tao Zheng, and Jian Sun. Shufflenet v2: Practical guidelines for efficient cnn architecture design. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 116–131, 2018. 1, 12

[21] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. *arXiv preprint arXiv:1801.04381*, 2018. 1, 12

[22] Roman Novak, Yasaman Bahri, Daniel A Abolafia, Jeffrey Pennington, and Jascha Sohl-Dickstein. Sensitivity and generalization in neural networks: an empirical study. *arXiv preprint arXiv:1802.08760*, 2018. 1

[23] Mikhail Belkin, Daniel Hsu, Siyuan Ma, and Soumik Mandal. Reconciling modern machine-learning practice and the classical bias–variance trade-off. *Proceedings of the National Academy of Sciences*, 116(32):15849–15854, 2019. 1

[24] Preetum Nakkiran, Gal Kaplun, Yamini Bansal, Tristan Yang, Boaz Barak, and Ilya Sutskever. Deep double descent: Where bigger models and more data hurt. *arXiv preprint arXiv:1912.02292*, 2019. 1

[25] Madhu S Advani, Andrew M Saxe, and Haim Sompolinsky. High-dimensional dynamics of generalization error in neural networks. *Neural Networks*, 132:428–446, 2020. 1

[26] Yehuda Dar, Vidya Muthukumar, and Richard G Baraniuk. A farewell to the bias-variance tradeoff? an overview of the theory of overparameterized machine learning. *arXiv preprint arXiv:2109.02355*, 2021. 1

[27] Ting Chen, Simon Kornblith, Kevin Swersky, Mohammad Norouzi, and Geoffrey Hinton. Big self-supervised models are strong semi-supervised learners. *arXiv preprint arXiv:2006.10029*, 2020. 1

[28] Yong-Deok Kim, Eunhyeok Park, Sungjoo Yoo, Taelim Choi, Lu Yang, and Dongjun Shin. Compression of deep convolutional neural networks for fast and low power mobile applications. *arXiv preprint arXiv:1511.06530*, 2015. 1

[29] Xiyu Yu, Tongliang Liu, Xinchao Wang, and Dacheng Tao. On compressing deep models by low rank and sparse decomposition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 7370–7379, 2017. 1, 12

[30] Karen Ullrich, Edward Meeds, and Max Welling. Soft weight-sharing for neural network compression. *arXiv preprint arXiv:1702.04008*, 2017. 1

[31] Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*, 2015. 1, 5, 12

[32] Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. Xnor-net: Imagenet classification using binary convolutional neural networks. In *European conference on computer vision*, pages 525–542. Springer, 2016. 1

[33] Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew Howard, Hartwig Adam, and Dmitry Kalenichenko. Quantization and training of neural networks for efficient integer-arithmetic-only inference. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2704–2713, 2018. 1

[34] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015. 1, 2, 5, 7, 8, 12, 48, 51, 59, 60

[35] Suraj Srinivas and R Venkatesh Babu. Data-free parameter pruning for deep neural networks. *arXiv preprint arXiv:1507.06149*, 2015. 1

[36] Guobin Chen, Wongun Choi, Xiang Yu, Tony Han, and Manmohan Chandraker. Learning efficient object detection models with knowledge distillation. In *Advances in Neural Information Processing Systems*, pages 742–751, 2017. 1

[37] Zi Wang. Data-free knowledge distillation with soft targeted transfer set synthesis. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 10245–10253, 2021. 1, 5, 8, 49, 67

[38] Chengcheng Li, Zi Wang, and Hairong Qi. Online knowledge distillation by temporal-spatial boosting. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pages 197–206, 2022. 1

[39] Song Han, Jeff Pool, John Tran, and William Dally. Learning both weights and connections for efficient neural network. In *Advances in neural information processing systems*, pages 1135–1143, 2015. 1, 2, 5, 12

[40] Adam Polyak and Lior Wolf. Channel-level acceleration of deep face representations. *IEEE Access*, 3:2163–2175, 2015. 1, 6, 20, 32, 33

[41] Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. Pruning filters for efficient convnets. *arXiv preprint arXiv:1608.08710*, 2016. 1, 2, 3, 5, 6, 12, 20, 32, 33

[42] Pavlo Molchanov, Stephen Tyree, Tero Karras, Timo Aila, and Jan Kautz. Pruning convolutional neural networks for resource efficient inference. *arXiv preprint arXiv:1611.06440*, 2016. 1, 2, 3, 6, 12, 18, 20, 32, 33

[43] Yihui He, Xiangyu Zhang, and Jian Sun. Channel pruning for accelerating very deep neural networks. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1389–1397, 2017. 1

[44] Jian-Hao Luo, Jianxin Wu, and Weiyao Lin. Thinet: A filter level pruning method for deep neural network compression. *arXiv preprint arXiv:1707.06342*, 2017. 1, 6

[45] Ruichi Yu, Ang Li, Chun-Fu Chen, Jui-Hsin Lai, Vlad I Morariu, Xintong Han, Mingfei Gao, Ching-Yung Lin, and Larry S Davis. Nisp: Pruning networks using neuron importance score propagation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 9194–9203, 2018. 1, 6, 33

[46] Yang He, Ping Liu, Ziwei Wang, Zhilan Hu, and Yi Yang. Filter pruning via geometric median for deep convolutional neural networks acceleration. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4340–4349, 2019. 1, 5, 6, 12, 32, 33

[47] Chengcheng Li, Zi Wang, and Hairong Qi. An efficient pipeline for pruning convolutional neural networks. In *2020 19th IEEE International Conference on Machine Learning and Applications (ICMLA)*, pages 907–912. IEEE, 2020. 1

[48] Zi Wang, Chengcheng Li, Xiangyang Wang, and Dali Wang. Towards efficient convolutional neural networks through low-error filter saliency estimation. In *Pacific Rim International Conference on Artificial Intelligence*, pages 255–267. Springer, 2019. 1, 12

[49] Tianyun Zhang, Shaokai Ye, Kaiqi Zhang, Jian Tang, Wujie Wen, Makan Fardad, and Yanzhi Wang. A systematic dnn weight pruning framework using alternating direction method of multipliers. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 184–199, 2018. 2, 5

[50] Song Han, Xingyu Liu, Huizi Mao, Jing Pu, Ardavan Pedram, Mark A Horowitz, and William J Dally. Eie: efficient inference engine on compressed deep neural network. In *Computer Architecture (ISCA), 2016 ACM/IEEE 43rd Annual International Symposium on*, pages 243–254. IEEE, 2016. 2, 6

[51] Adriana Romero, Nicolas Ballas, Samira Ebrahimi Kahou, Antoine Chassang, Carlo Gatta, and Yoshua Bengio. Fitnets: Hints for thin deep nets. *arXiv preprint arXiv:1412.6550*, 2014. 2, 7

[52] Sergey Zagoruyko and Nikos Komodakis. Paying more attention to attention: Improving the performance of convolutional neural networks via attention transfer. *arXiv preprint arXiv:1612.03928*, 2016. 2, 7

[53] Byeongho Heo, Minsik Lee, Sangdoo Yun, and Jin Young Choi. Knowledge distillation with adversarial samples supporting decision boundary. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 3771–3778, 2019. 2, 7

[54] Yang He, Guoliang Kang, Xuanyi Dong, Yanwei Fu, and Yi Yang. Soft filter pruning for accelerating deep convolutional neural networks. *arXiv preprint arXiv:1808.06866*, 2018. 2, 32, 33

[55] Hengyuan Hu, Rui Peng, Yu-Wing Tai, and Chi-Keung Tang. Network trimming: A data-driven neuron pruning approach towards efficient deep architectures. *arXiv preprint arXiv:1607.03250*, 2016. 2, 6, 20, 33

[56] Pavlo Molchanov, Arun Mallya, Stephen Tyree, Iuri Frosio, and Jan Kautz. Importance estimation for neural network pruning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11264–11272, 2019. 3

[57] Philipp J Keller. Imaging morphogenesis: technological advances and biological insights. *Science*, 340(6137), 2013. 3, 126

[58] Yunhe Feng, Zongyao Chen, Dali Wang, Jian Chen, and Zhili Feng. Deepwelding: A deep learning enhanced approach to gtaw using multisource sensing images. *IEEE Transactions on Industrial Informatics*, 16(1):465–474, 2019. 3

[59] Zi Wang, Dali Wang, Chengcheng Li, Yichi Xu, Husheng Li, and Zhirong Bao. Deep reinforcement learning of cell movement in the early stage of c. elegans embryogenesis. *Bioinformatics*, 34(18):3169–3177, 2018. 3, 132, 146

[60] Zi Wang, Yichi Xu, Dali Wang, Jiawei Yang, and Zhirong Bao. Hierarchical deep reinforcement learning reveals a modular mechanism of cell movement. *Nature Machine Intelligence*, pages 1–11, 2022. 3

[61] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017. 5

[62] Viacheslav Moskalenko, Alona Moskalenko, Artem Korobov, Olha Boiko, Serhii Martynenko, and Oleksandr Borovenskyi. Model and training methods of autonomous navigation system for compact drones. In *2018 IEEE Second International Conference on Data Stream Mining & Processing (DSMP)*, pages 503–508. IEEE, 2018. 5

[63] Max Jaderberg, Andrea Vedaldi, and Andrew Zisserman. Speeding up convolutional neural networks with low rank expansions. *arXiv preprint arXiv:1405.3866*, 2014. 5

[64] Mary Phuong and Christoph Lampert. Towards understanding knowledge distillation. In *International Conference on Machine Learning*, pages 5142–5151. PMLR, 2019. 5

[65] Xin Jin, Cuiling Lan, Wenjun Zeng, and Zhibo Chen. Uncertainty-aware multi-shot knowledge distillation for image-based object re-identification. *arXiv preprint arXiv:2001.05197*, 2020. 5

[66] Sukmin Yun, Jongjin Park, Kimin Lee, and Jinwoo Shin. Regularizing class-wise predictions via self-knowledge distillation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 13876–13885, 2020. 5

[67] Nikolaos Passalis, Maria Tzelepi, and Anastasios Tefas. Heterogeneous knowledge distillation using information flow modeling. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2339–2348, 2020. 5

[68] Stephen José Hanson and Lorien Y Pratt. Comparing biases for minimal network construction with back-propagation. In *Advances in neural information processing systems*, pages 177–185, 1989. 5

[69] Yann LeCun, John S Denker, and Sara A Solla. Optimal brain damage. In *Advances in neural information processing systems*, pages 598–605, 1990. 5

[70] Wenlin Chen, James Wilson, Stephen Tyree, Kilian Weinberger, and Yixin Chen. Compressing neural networks with the hashing trick. In *International Conference on Machine Learning*, pages 2285–2294, 2015. 5

[71] Xingyu Liu, Jeff Pool, Song Han, and William J Dally. Efficient sparse-winograd convolutional neural networks. *arXiv preprint arXiv:1802.06367*, 2018. 5

[72] Hanyu Peng, Jiaxiang Wu, Shifeng Chen, and Junzhou Huang. Collaborative channel pruning for deep networks. In *International Conference on Machine Learning*, pages 5113–5122, 2019. 6

[73] Wenyuan Zeng and Raquel Urtasun. Mlprune: Multi-layer pruning for automated neural network compression. 2018. 6

[74] Mingbao Lin, Rongrong Ji, Yan Wang, Yichen Zhang, Baochang Zhang, Yonghong Tian, and Ling Shao. Hrank: Filter pruning using high-rank feature map. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 1529–1538, 2020. 6, 33

[75] Zhengguang Zhou, Wengang Zhou, Houqiang Li, and Richang Hong. Online filter clustering and pruning for efficient convnets. In *2018 25th IEEE International Conference on Image Processing (ICIP)*, pages 11–15. IEEE, 2018. 6

[76] Rahul Duggal, Cao Xiao, Richard Vuduc, and Jimeng Sun. Cup: Cluster pruning for compressing deep neural networks. *arXiv preprint arXiv:1911.08630*, 2019. 6, 33

[77] Anubhav Ashok, Nicholas Rhinehart, Fares Beainy, and Kris M Kitani. N2n learning: Network to network compression via policy gradient reinforcement learning. In *International Conference on Learning Representations*, 2018. 6

[78] Zuxuan Wu, Tushar Nagarajan, Abhishek Kumar, Steven Rennie, Larry S Davis, Kristen Grauman, and Rogerio Feris. Blockdrop: Dynamic inference paths in residual networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 8817–8826, 2018. 6

[79] Xuanyang Zhang, Zhanxing Zhu, Zenglin Xu, et al. Learning to search efficient densenet with layer-wise pruning. 2018. 6

[80] Yihui He, Ji Lin, Zhijian Liu, Hanrui Wang, Li-Jia Li, and Song Han. Amc: Automl for model compression and acceleration on mobile devices. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 784–800, 2018. 6

[81] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015. 6, 9, 124, 127

[82] Qiangui Huang, Kevin Zhou, Suya You, and Ulrich Neumann. Learning to prune filters in convolutional neural networks. In *2018 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pages 709–718. IEEE, 2018. 6

[83] Jian-Hao Luo and Jianxin Wu. Autopruner: An end-to-end trainable filter pruning method for efficient deep model inference. *arXiv preprint arXiv:1805.08941*, 2018. 6, 33

[84] Yawei Li, Shuhang Gu, Christoph Mayer, Luc Van Gool, and Radu Timofte. Group sparsity: The hinge between filter pruning and decomposition for network compression. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 8018–8027, 2020. 6

[85] Maxim Berman, Leonid Pishchulin, Ning Xu, Matthew B Blaschko, and Gérard Medioni. Aows: Adaptive and optimal network width search with latency constraints. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11217–11226, 2020. 6

[86] Eugene Lee and Chen-Yi Lee. Neuralscale: Efficient scaling of neurons for resource-constrained deep neural networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 1478–1487, 2020. 6

[87] Yawei Li, Shuhang Gu, Kai Zhang, Luc Van Gool, and Radu Timofte. Dhp: Differentiable meta pruning via hypernetworks. In *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part VIII 16*, pages 608–624. Springer, 2020. 6

[88] Bowen Baker, Otkrist Gupta, Nikhil Naik, and Ramesh Raskar. Designing neural network architectures using reinforcement learning. *arXiv preprint arXiv:1611.02167*, 2016. 7

[89] Barret Zoph and Quoc V Le. Neural architecture search with reinforcement learning. *arXiv preprint arXiv:1611.01578*, 2016. 7

[90] Xuanyi Dong and Yi Yang. Network pruning via transformable architecture search. In *Advances in Neural Information Processing Systems*, pages 759–770, 2019. 7, 33

[91] Zhuang Liu, Mingjie Sun, Tinghui Zhou, Gao Huang, and Trevor Darrell. Rethinking the value of network pruning. In *International Conference on Learning Representations*, 2018. 7

[92] Deepak Mittal, Shweta Bhardwaj, Mitesh M Khapra, and Balaraman Ravindran. Recovering from random pruning: On the plasticity of deep convolutional neural networks. *arXiv preprint arXiv:1801.10447*, 2018. 7

[93] Cristian Buciluǎ, Rich Caruana, and Alexandru Niculescu-Mizil. Model compression. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 535–541, 2006. 7

[94] Jimmy Ba and Rich Caruana. Do deep nets really need to be deep? In *Advances in neural information processing systems*, pages 2654–2662, 2014. 7

[95] Junho Yim, Donggyu Joo, Jihoon Bae, and Junmo Kim. A gift from knowledge distillation: Fast optimization, network minimization and transfer learning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4133–4141, 2017. 7

[96] Jangho Kim, SeongUk Park, and Nojun Kwak. Paraphrasing complex network: Network compression via factor transfer. In *Advances in neural information processing systems*, pages 2760–2769, 2018. 7

[97] Akisato Kimura, Zoubin Ghahramani, Koh Takeuchi, Tomoharu Iwata, and Naonori Ueda. Few-shot learning of neural networks from scratch by pseudo example optimization. *arXiv preprint arXiv:1802.03039*, 2018. 8, 67

[98] Dongdong Wang, Yandong Li, Liqiang Wang, and Boqing Gong. Neural networks are more productive teachers than human raters: Active mixup for data-efficient knowledge distillation from a blackbox model. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 1498–1507, 2020. 8, 67

[99] Tianhong Li, Jianguo Li, Zhuang Liu, and Changshui Zhang. Few sample knowledge distillation for efficient network compression. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 14639–14647, 2020. 8

[100] Raphael Gontijo Lopes, Stefano Fenu, and Thad Starner. Data-free knowledge distillation for deep neural networks. *arXiv preprint arXiv:1710.07535*, 2017. 8, 59, 67

[101] Gaurav Kumar Nayak, Konda Reddy Mopuri, Vaisakh Shaj, R Venkatesh Babu, and Anirban Chakraborty. Zero-shot knowledge distillation in deep networks. *arXiv preprint arXiv:1905.08114*, 2019. 8, 48, 67

[102] Hanting Chen, Yunhe Wang, Chang Xu, Zhaohui Yang, Chuanjian Liu, Boxin Shi, Chunjing Xu, Chao Xu, and Qi Tian. Data-free learning of student networks. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 3514–3522, 2019. 8, 49, 59, 67

[103] Paul Micaelli and Amos J Storkey. Zero-shot knowledge transfer via adversarial belief matching. In *Advances in Neural Information Processing Systems*, pages 9551–9561, 2019. 8

[104] Hongxu Yin, Pavlo Molchanov, Jose M Alvarez, Zhizhong Li, Arun Mallya, Derek Hoiem, Niraj K Jha, and Jan Kautz. Dreaming to distill: Data-free knowledge transfer via deepinversion. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8715–8724, 2020. 8, 49

[105] Tribhuvanesh Orekondy, Bernt Schiele, and Mario Fritz. Knockoff nets: Stealing functionality of black-box models. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4954–4963, 2019. 8

[106] Linfeng Zhang, Jiebo Song, Anni Gao, Jingwei Chen, Chenglong Bao, and Kaisheng Ma. Be your own teacher: Improve the performance of convolutional neural networks via self distillation. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 3713–3722, 2019. 9

[107] Tommaso Furlanello, Zachary Lipton, Michael Tschannen, Laurent Itti, and Anima Anandkumar. Born again neural networks. In *International Conference on Machine Learning*, pages 1607–1616. PMLR, 2018. 9

[108] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015. 9, 104, 107, 124, 127, 150

[109] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International Conference on Machine Learning*, pages 1928–1937, 2016. 9, 124

[110] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017. 9

[111] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013. 9, 99, 104, 107, 114

[112] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484, 2016. 9, 74, 127

[113] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the game of go without human knowledge. *Nature*, 550(7676):354–359, 2017. 9

[114] Shixiang Gu, Ethan Holly, Timothy Lillicrap, and Sergey Levine. Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates. In *2017 IEEE international conference on robotics and automation (ICRA)*, pages 3389–3396. IEEE, 2017. 9, 127

[115] Li Li, Yisheng Lv, and Fei-Yue Wang. Traffic signal timing via deep reinforcement learning. *IEEE/CAA Journal of Automatica Sinica*, 3(3):247–254, 2016. 9

[116] Charles M Macal and Michael J North. Tutorial on agent-based modeling and simulation. In *Proceedings of the Winter Simulation Conference, 2005.*, pages 14–pp. IEEE, 2005. 9

[117] Minsung Kim and Minki Kim. Group-wise herding behavior in financial markets: an agent-based modeling approach. *PloS one*, 9(4):e93661, 2014. 9

[118] Vygintas Gontis and Aleksejus Kononovicius. Consentaneous agent-based and stochastic model of the financial markets. *PLoS One*, 9(7):e102201, 2014. 9

[119] Thomas E Gorochowski, Antoni Matyjaszkiewicz, Thomas Todd, Neeraj Oak, Kira Kowalska, Stephen Reid, Krasimira T Tsaneva-Atanasova, Nigel J Savery, Claire S Grierson, and Mario Di Bernardo. Bsim: an agent-based tool for modeling bacterial populations in systems and synthetic biology. 2012. 9

[120] Takuto Sakamoto. Computational research on mobile pastoralism using agent-based modeling and satellite imagery. *PloS one*, 11(3):e0151157, 2016. 9

[121] Anja Voss-Böhme. Multi-scale modeling in morphogenesis: a critical analysis of the cellular potts model. 2012. 10

[122] Nan Chen, James A Glazier, Jesús A Izaguirre, and Mark S Alber. A parallel implementation of the cellular potts model for simulation of cell-based morphogenesis. *Computer physics communications*, 176(11-12):670–681, 2007. 10

[123] Yaki Setty. Multi-scale computational modeling of developmental biology. *Bioinformatics*, 28(15):2022–2028, 2012. 10, 98, 121

[124] Yaki Setty, Diana Dalfó, Dorota Z Korta, E Jane Albert Hubbard, and Hillel Kugler. A model of stem cell population dynamics: in silico analysis and in vivo validation. *Development*, 139(1):47–56, 2012. 10

[125] Kathryn Atwell, Zhao Qin, David Gavaghan, Hillel Kugler, E Jane Albert Hubbard, and James M Osborne. Mechano-logical model of c. elegans germ line suggests feedback on the cell cycle. *Development*, 142(22):3902–3911, 2015. 10

[126] Yaki Setty, Irun R Cohen, Yuval Dor, and David Harel. Four-dimensional realistic modeling of pancreatic organogenesis. *Proceedings of the National Academy of Sciences*, 105(51):20374–20379, 2008. 10, 121

[127] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553):436, 2015. 12

[128] Jiahui Yu, Linjie Yang, Ning Xu, Jianchao Yang, and Thomas Huang. Slimmable neural networks. *arXiv preprint arXiv:1812.08928*, 2018. 12, 33

[129] Chengcheng Li, Zi Wang, Xiangyang Wang, and Hairong Qi. Single-shot channel pruning based on alternating direction method of multipliers. *arXiv preprint arXiv:1902.06382*, 2019. 12

[130] Zi Wang and Chengcheng Li. Channel pruning via lookahead search guided reinforcement learning. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pages 2029–2040, 2022. 12

[131] Xiaohan Ding, Guiguang Ding, Yuchen Guo, Jungong Han, and Chenggang Yan. Approximated oracle filter pruning for destructive cnn width optimization. *arXiv preprint arXiv:1905.04748*, 2019. 12

[132] Zi Wang, Chengcheng Li, and Xiangyang Wang. Convolutional neural network pruning with structural redundancy reduction. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 14913–14922, 2021. 13

[133] Matthew Richard Gibson. Clusters and covers: geometric set cover algorithms. 2010. 29

[134] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. Technical report, Citeseer, 2009. 32, 59

[135] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009. 32, 48

[136] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems*, pages 8024–8035, 2019. 33

[137] Zhuangwei Zhuang, Mingkui Tan, Bohan Zhuang, Jing Liu, Yong Guo, Qingyao Wu, Junzhou Huang, and Jinhui Zhu. Discrimination-aware channel pruning for deep neural networks. In *Advances in Neural Information Processing Systems*, pages 875–886, 2018. 33

[138] Shaohui Lin, Rongrong Ji, Chenqian Yan, Baochang Zhang, Liujuan Cao, Qixiang Ye, Feiyue Huang, and David Doermann. Towards optimal structured cnn pruning via generative adversarial learning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2790–2799, 2019. 33

[139] Mingbao Lin, Rongrong Ji, Yuxin Zhang, Baochang Zhang, Yongjian Wu, and Yonghong Tian. Channel pruning via automatic structure search. *arXiv preprint arXiv:2001.08565*, 2020. 33

[140] Yuhui Xu, Yuxi Li, Shuai Zhang, Wei Wen, Botao Wang, Yingyong Qi, Yiran Chen, Weiyao Lin, and Hongkai Xiong. Trp: Trained rank pruning for efficient deep neural networks. *arXiv preprint arXiv:2004.14566*, 2020. 33

[141] Minsoo Kang and Bohyung Han. Operation-aware soft channel pruning using differentiable masks. *arXiv preprint arXiv:2007.03938*, 2020. 33

[142] Paul Covington, Jay Adams, and Emre Sargin. Deep neural networks for youtube recommendations. In *Proceedings of the 10th ACM conference on recommender systems*, pages 191–198, 2016. 49

[143] Gustavo López, Luis Quesada, and Luis A Guerrero. Alexa vs. siri vs. cortana vs. google assistant: a comparison of speech-based natural user interfaces. In *International Conference on Applied Human Factors and Ergonomics*, pages 241–250. Springer, 2017. 49

[144] Inc Clarifai. Clarifai: Computer vision and ai enterprise platform. 2020. 49

[145] Wieland Brendel, Jonas Rauber, and Matthias Bethge. Decision-based adversarial attacks: Reliable attacks against black-box machine learning models. *arXiv preprint arXiv:1712.04248*, 2017. 55

[146] Minhao Cheng, Thong Le, Pin-Yu Chen, Jinfeng Yi, Huan Zhang, and Cho-Jui Hsieh. Query-efficient hard-label black-box attack: An optimization-based approach. *arXiv preprint arXiv:1807.04457*, 2018. 55

[147] Yujia Liu, Seyed-Mohsen Moosavi-Dezfooli, and Pascal Frossard. A geometry-inspired decision-based attack. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 4890–4898, 2019. 55

[148] Minhao Cheng, Simranjit Singh, Patrick Chen, Pin-Yu Chen, Sijia Liu, and Cho-Jui Hsieh. Sign-opt: A query-efficient hard-label adversarial attack. *arXiv preprint arXiv:1909.10773*, 2019. 55

[149] Yining Wang, Simon Du, Sivaraman Balakrishnan, and Aarti Singh. Stochastic zeroth-order optimization in high dimensions. In *International Conference on Artificial Intelligence and Statistics*, pages 1356–1365, 2018. 55

[150] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998. 59

[151] Maria-Elena Nilsback and Andrew Zisserman. Automated flower classification over a large number of classes. In *2008 Sixth Indian Conference on Computer Vision, Graphics & Image Processing*, pages 722–729. IEEE, 2008. 59

[152] Kenji Sugioka, Kota Mizumoto, and Hitoshi Sawa. Wnt regulates spindle asymmetry to generate asymmetric nuclear β-catenin in c. elegans. *Cell*, 146(6):942–954, 2011. 73

[153] Zhuo Du, Anthony Santella, Fei He, Michael Tiongson, and Zhirong Bao. De novo inference of systems-level mechanistic models of development from live-imaging-based phenotype analysis. *Cell*, 156(1):359–372, 2014. 74, 98, 124

[154] Zhuo Du, Anthony Santella, Fei He, Pavak K Shah, Yuko Kamikawa, and Zhirong Bao. The regulatory landscape of lineage differentiation in a metazoan embryo. *Developmental cell*, 34(5):592–607, 2015. 74, 80, 98, 100, 124, 138

[155] Anthony Santella, Ismar Kovacevic, Laura A Herndon, David H Hall, Zhuo Du, and Zhirong Bao. Digital development: a database of cell lineage differentiation in c. elegans with lineage phenotypes, cell-specific gene functions and a multiscale model. *Nucleic acids research*, page gkv1119, 2015. 74

[156] Jonathan W Brandt and V Ralph Algazi. Continuous skeleton computation by voronoi diagram. *CVGIP: Image understanding*, 55(3):329–338, 1992. 74

[157] Jürgen Hench, Johan Henriksson, Martin Lüppert, and Thomas R Bürglin. Spatio-temporal reference model of caenorhabditis elegans embryogenesis with cell contact maps. *Developmental biology*, 333(1):1–13, 2009. 74, 98

[158] Miguel A Luengo-Oroz, Louise Duloquin, C Castro, Thierry Savy, Emmanuel Faure, Benoit Lombardot, Paul Bourgine, Nadine Peyriéras, and Andrés Santos. Can voronoi diagram model cell geometries in early sea-urchin embryogenesis? In *Biomedical Imaging: From Nano to Macro, 2008. ISBI 2008. 5th IEEE International Symposium on*, pages 504–507. IEEE, 2008. 74

[159] Nicolas Olivier, Miguel A Luengo-Oroz, Louise Duloquin, Emmanuel Faure, Thierry Savy, Israël Veilleux, Xavier Solinas, Delphine Débarre, Paul Bourgine, Andrés Santos, et al. Cell lineage reconstruction of early zebrafish embryos using label-free nonlinear microscopy. *Science*, 329(5994):967–971, 2010. 74

[160] Franz Aurenhammer and Herbert Edelsbrunner. An optimal algorithm for constructing the weighted voronoi diagram in the plane. *Pattern Recognition*, 17(2):251–257, 1984. 74

[161] Tsung-Jung Hsieh, Hsiao-Fen Hsiao, and Wei-Chang Yeh. Forecasting stock markets using wavelet transforms and recurrent neural networks: An integrated system based on artificial bee colony algorithm. *Applied soft computing*, 11(2):2510–2525, 2011. 74

[162] Meenakshi Rawat, Karun Rawat, and Fadhel M Ghannouchi. Adaptive digital predistortion of wireless power amplifiers/transmitters using dynamic real-valued focused time-delay line neural networks. *IEEE Transactions on Microwave Theory and Techniques*, 58(1):95–104, 2010. 74

[163] A Okebe, B Boots, and K Sugihara. Concepts and applications of voronoi diagrams, 2000. 75

[164] Begoña Abellanas, Manuel Abellanas, Arne Pommerening, Dolores Lodares, and Simón Cuadros. A forest simulation approach using weighted voronoi diagrams. an application to mediterranean fir abies pinsapo boiss stands. *Forest Systems*, 25(2):062, 2016. 75

[165] Jun Guo and Hamid Jafarkhani. Sensor deployment with limited communication range in homogeneous and heterogeneous wireless sensor networks. *IEEE Transactions on Wireless Communications*, 15(10):6771–6784, 2016. 75

[166] Anthony Santella, Raúl Catena, Ismar Kovacevic, Pavak Shah, Zidong Yu, Javier Marquina-Solis, Abhishek Kumar, Yicong Wu, James Schaff, Daniel Colón-Ramos,

et al. Wormguides: an interactive single cell developmental atlas and tool for collaborative multidimensional data exploration. *BMC bioinformatics*, 16(1):189, 2015. 75, 87, 153

[167] Franz Aurenhammer. Voronoi diagrams-a survey of a fundamental geometric data structure. *ACM Computing Surveys (CSUR)*, 23(3):345–405, 1991. 75

[168] Kenneth E Hoff III, John Keyser, Ming Lin, Dinesh Manocha, and Tim Culver. Fast computation of generalized voronoi diagrams using graphics hardware. In *Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, pages 277–286. ACM Press/Addison-Wesley Publishing Co., 1999. 77

[169] Christopher M.. Bishop. *Pattern recognition and machine learning*. Springer, 2006. 79

[170] Julia L Moore, Zhuo Du, and Zhirong Bao. Systematic quantification of developmental phenotypes at single-cell resolution during embryogenesis. *Development*, 140(15):3266–3274, 2013. 80, 109, 113, 134

[171] UTK. Utk eecs hydra lab. 2017. 80

[172] James R Priess. Notch signaling in the c. elegans embryo. *WormBook*, 1:1–16, 2005. 84

[173] Thomas J Boyle, Zhirong Bao, John I Murray, Carlos L Araya, and Robert H Waterston. Acetree: a tool for visual analysis of caenorhabditis elegans embryogenesis. *BMC bioinformatics*, 7(1):275, 2006. 94, 112

[174] John Isaac Murray, Thomas J Boyle, Elicia Preston, Dionne Vafeados, Barbara Mericle, Peter Weisdepp, Zhongying Zhao, Zhirong Bao, Max Boeck, and Robert H Waterston. Multidimensional regulation of gene expression in the c. elegans embryo. *Genome research*, 22(7):1282–1294, 2012. 98

[175] Dali Wang, Zheng Lu, Yichi Xu, Zi Wang, Anthony Santella, and Zhirong Bao. Cellular structure image classification with small targeted training samples. *IEEE Access*, 7:148967–148974, 2019. 98

[176] Ralf Schnabel, Harald Hutter, Don Moerman, and Heinke Schnabel. Assessing normal embryogenesis incaenorhabditis elegansusing a 4d microscope: Variability of development and regional specification. *Developmental biology*, 184(2):234–265, 1997. 98

[177] Claudiu A Giurumescu, Sukryool Kang, Thomas A Planchon, Eric Betzig, Joshua Bloomekatz, Deborah Yelon, Pamela Cosman, and Andrew D Chisholm. Quantitative semi-automated analysis of morphogenesis with single-cell resolution in complex embryos. *Development*, 139(22):4271–4279, 2012. 98

[178] Koji Kyoda, Eru Adachi, Eriko Masuda, Yoko Nagai, Yoko Suzuki, Taeko Oguro, Mitsuru Urai, Ryoko Arai, Mari Furukawa, Kumiko Shimada, et al. Wddd: worm developmental dynamics database. *Nucleic acids research*, 41(D1):D732–D737, 2012. 98

[179] Andy L Olivares, Miguel A González Ballester, and Jérôme Noailly. Virtual exploration of early stage atherosclerosis. *Bioinformatics*, 32(24):3798–3806, 2016. 98

[180] Stefan Hoehme and Dirk Drasdo. A cell-based simulation software for multi-cellular systems. *Bioinformatics*, 26(20):2641–2642, 2010. 98

[181] Zi Wang, Benjamin J Ramsey, Dali Wang, Kwai Wong, Husheng Li, Eric Wang, and Zhirong Bao. An observation-driven agent-based modeling and analysis framework for c. elegans embryogenesis. *PloS one*, 11(11):e0166551, 2016. 99, 100

[182] Eric Wang, Anthony Santella, Zi Wang, Dali Wang, and Zhirong Bao. Visualization of 3-dimensional vectors in a dynamic embryonic system—wormguides. *Journal of Computer and Communications*, 5(12):70, 2017. 99

[183] Dali Wang, Zi Wang, Xiaopeng Zhao, Yichi Xu, Zhirong Bao, et al. An observation data driven simulation and analysis framework for early stage c. elegans embryogenesis. *Journal of Biomedical Science and Engineering*, 11(08):225, 2018. 99

[184] Jen-Yi Lee and Bob Goldstein. Mechanisms of cell positioning during c. elegans gastrulation. *Development*, 130(2):307–320, 2003. 99

[185] David Shook and Ray Keller. Mechanisms, mechanics and function of epithelial–mesenchymal transitions in early development. *Mechanisms of development*, 120(11):1351–1383, 2003. 99

[186] Chun-Min Lo, Hong-Bei Wang, Micah Dembo, and Yu-li Wang. Cell movement is guided by the rigidity of the substrate. *Biophysical journal*, 79(1):144–152, 2000. 99

[187] Zhirong Bao, John I Murray, Thomas Boyle, Siew Loon Ooi, Matthew J Sandel, and Robert H Waterston. Automated cell lineage tracing in caenorhabditis elegans. *Proceedings of the National Academy of Sciences of the United States of America*, 103(8):2707–2712, 2006. 101, 113, 128

[188] Carl-Philipp Heisenberg and Yohanns Bellaïche. Forces in tissue morphogenesis and patterning. *Cell*, 153(5):948–962, 2013. 101

[189] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*, volume 1. MIT press Cambridge, 1998. 102, 107

[190] Maxim Egorov. Multi-agent deep reinforcement learning. 2016. 107

[191] Zi Wang, Dali Wang, Husheng Li, and Zhirong Bao. Cell neighbor determination in the metazoan embryo system. In *Proceedings of the 8th ACM International Conference on Bioinformatics, Computational Biology, and Health Informatics*, pages 305–312. ACM, 2017. 109, 113, 131, 148, 149

[192] John E Sulston, E Schierenberg, John G White, and JN Thomson. The embryonic cell lineage of the nematode caenorhabditis elegans. *Developmental biology*, 100(1):64–119, 1983. 112, 128, 143

[193] Anthony Santella, Zhuo Du, and Zhirong Bao. A semi-local neighborhood-based framework for probabilistic cell lineage tracing. *BMC bioinformatics*, 15(1):217, 2014. 112, 128, 147

[194] Christian Pohl and Zhirong Bao. Chiral forces organize left-right patterning in c. elegans by uncoupling midline and anteroposterior axis. *Developmental cell*, 19(3):402–412, 2010. 113

[195] Zhirong Bao and John I Murray. Mounting caenorhabditis elegans embryos for live imaging of embryogenesis. *Cold Spring Harbor Protocols*, 2011(9):pdb–prot065599, 2011. 113

[196] Lucian Busoniu, Robert Babuska, and Bart De Schutter. A comprehensive survey of multiagent reinforcement learning. *IEEE Transactions on Systems, Man, And Cybernetics-Part C: Applications and Reviews, 38 (2), 2008*, 2008. 123

[197] Ardi Tampuu, Tambet Matiisen, Dorian Kodelja, Ilya Kuzovkin, Kristjan Korjus, Juhan Aru, Jaan Aru, and Raul Vicente. Multiagent cooperation and competition with deep reinforcement learning. *PloS one*, 12(4):e0172395, 2017. 123

[198] Tejas D Kulkarni, Karthik Narasimhan, Ardavan Saeedi, and Josh Tenenbaum. Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation. In *Advances in Neural Information Processing Systems*, pages 3675–3683, 2016. 124, 127, 131, 150

[199] Matthew D Zeiler. Adadelta: an adaptive learning rate method. *arXiv preprint arXiv:1212.5701*, 2012. 124

[200] Chinmay Belthangady and Loic A Royer. Applications, promises, and pitfalls of deep learning for fluorescence image reconstruction. *Nature methods*, 16(12):1215–1225, 2019. 126, 127

[201] Erick Moen, Dylan Bannon, Takamasa Kudo, William Graf, Markus Covert, and David Van Valen. Deep learning for cellular image analysis. *Nature methods*, 16(12):1233–1246, 2019. 126, 127

[202] Kristopher M Barnes, Li Fan, Mark W Moyle, Christopher A Brittin, Yichi Xu, Daniel A Colón-Ramos, Anthony Santella, and Zhirong Bao. Cadherin preserves cohesion across involuting tissues during c. elegans neurulation. *Elife*, 9:e58626, 2020. 126

[203] Felix Buggenthin, Florian Buettner, Philipp S Hoppe, Max Endele, Manuel Kroiss, Michael Strasser, Michael Schwarzfischer, Dirk Loeffler, Konstantinos D Kokkaliaris, Oliver Hilsenbeck, et al. Prospective identification of hematopoietic lineage choice by deep learning. *Nature methods*, 14(4):403–406, 2017. 126

[204] Benoit Ladoux and René-Marc Mège. Mechanobiology of collective cell behaviours. *Nature reviews Molecular cell biology*, 18(12):743–757, 2017. 126

[205] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, et al. A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science*, 362(6419):1140–1144, 2018. 127

[206] Hai Nguyen and Hung La. Review of deep reinforcement learning for robot manipulation. In *2019 Third IEEE International Conference on Robotic Computing (IRC)*, pages 590–595. IEEE, 2019. 127

[207] Dmitry Kalashnikov, Alex Irpan, Peter Pastor, Julian Ibarz, Alexander Herzog, Eric Jang, Deirdre Quillen, Ethan Holly, Mrinal Kalakrishnan, Vincent Vanhoucke, et al.

Qt-opt: Scalable deep reinforcement learning for vision-based robotic manipulation. *arXiv preprint arXiv:1806.10293*, 2018. 127

[208] Kai Arulkumaran, Marc Peter Deisenroth, Miles Brundage, and Anil Anthony Bharath. Deep reinforcement learning: A brief survey. *IEEE Signal Processing Magazine*, 34(6):26–38, 2017. 127

[209] Emre O Neftci and Bruno B Averbeck. Reinforcement learning in artificial and biological systems. *Nature Machine Intelligence*, 1(3):133–143, 2019. 127

[210] Richard S Sutton, Doina Precup, and Satinder Singh. Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artificial intelligence*, 112(1-2):181–211, 1999. 127

[211] Alexander Sasha Vezhnevets, Simon Osindero, Tom Schaul, Nicolas Heess, Max Jaderberg, David Silver, and Koray Kavukcuoglu. Feudal networks for hierarchical reinforcement learning. In *International Conference on Machine Learning*, pages 3540–3549. PMLR, 2017. 127

[212] Chen Tessler, Shahar Givony, Tom Zahavy, Daniel Mankowitz, and Shie Mannor. A deep hierarchical approach to lifelong learning in minecraft. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 31, 2017. 127

[213] Anthony Santella, Zhuo Du, Sonja Nowotschin, Anna-Katerina Hadjantonakis, and Zhirong Bao. A hybrid blob-slice model for accurate and efficient detection of fluorescence labeled nuclei in 3d. *BMC bioinformatics*, 11(1):1–13, 2010. 128, 147

[214] Braden Katzman, Doris Tang, Anthony Santella, and Zhirong Bao. Acetree: a major update and case study in the long term maintenance of open-source scientific software. *BMC bioinformatics*, 19(1):1–7, 2018. 128, 147

[215] Zi Wang, Dali Wang, Chengcheng Li, Yichi Xu, Husheng Li, and Zhirong Bao. Modeling cell migration with convolutional neural network and deep reinforcement

learning. Technical report, Oak Ridge National Lab.(ORNL), Oak Ridge, TN (United States), 2019. 132

[216] Pavak K Shah, Matthew R Tanner, Ismar Kovacevic, Aysha Rankin, Teagan E Marshall, Nathaniel Noblett, Nhan Nguyen Tran, Tony Roenspies, Jeffrey Hung, Zheqian Chen, et al. Pcp and sax-3/robo pathways cooperate to regulate convergent extension-based nerve cord assembly in c. elegans. *Developmental cell*, 41(2):195–203, 2017. 132, 136, 153

[217] Adam C Paré, Athea Vichas, Christopher T Fincher, Zachary Mirman, Dene L Farrell, Avantika Mainieri, and Jennifer A Zallen. A positional toll receptor code directs convergent extension in drosophila. *Nature*, 515(7528):523–527, 2014. 136, 137

[218] Craig P Hunter and Cynthia Kenyon. Spatial and temporal controls target pal-1 blastomere-specification activity to a single blastomere lineage in c. elegans embryos. *Cell*, 87(2):217–226, 1996. 138

[219] Yicong Wu, Alireza Ghitani, Ryan Christensen, Anthony Santella, Zhuo Du, Gary Rondeau, Zhirong Bao, Daniel Colón-Ramos, and Hari Shroff. Inverted selective plane illumination microscopy (ispim) enables coupled cell identity lineaging and neurodevelopmental imaging in caenorhabditis elegans. *Proceedings of the National Academy of Sciences*, 108(43):17708–17713, 2011. 145

[220] Andrea Banino, Caswell Barry, Benigno Uria, Charles Blundell, Timothy Lillicrap, Piotr Mirowski, Alexander Pritzel, Martin J Chadwick, Thomas Degris, Joseph Modayil, et al. Vector-based navigation using grid-like representations in artificial agents. *Nature*, 557(7705):429–433, 2018. 146

Appendix

Appendix A

Supplement for Chapter 7

A.1 Architecture of the CNN and the policy network in HDRL

The network architecture used as our models are shown in Table A.1.

A.2 MI curves of the successful migration scenarios using the Motion Model

The MI curve of individual scenarios (21 in total) are presented in Fig. A.1.

A.3 Feature maps of Cpaaa migration at one of the early migration timesteps

The feature maps extracted from the policy network at one of the early migration timesteps are presented in Fig. A.2.

Table A.1: Architecture of the CNN and the policy network in HDRL.

	Layer index	Layer type	Number of filters	Kernel size	Stride	Padding	Activation
CNN	1	Convolution	32	5x5	4	2	ReLu
	2	Maxpoling	-	2x2	2	0	-
	3	Convolution	64	3x3	1	1	Relu
	4	Maxpooling	-	2x2	2	0	-
policy network	5	Fully connected	Number of subgoals	-	-	-	-

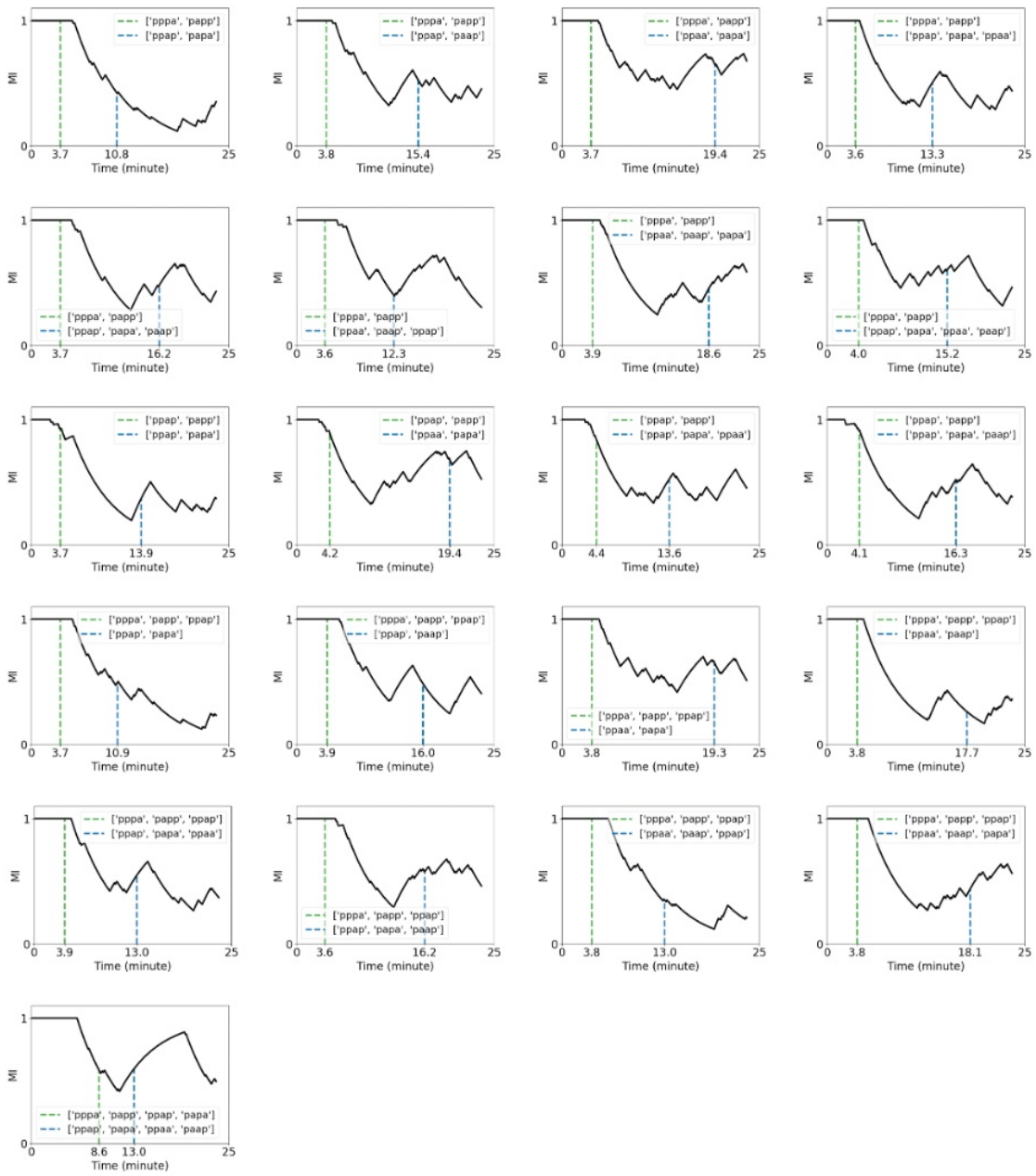


Figure A.1: MI curves of the successful migration scenarios using the Motion Model. Green and blue dashed lines indicate the time when the first and second potential subgoals (ABarp cells) are reached.

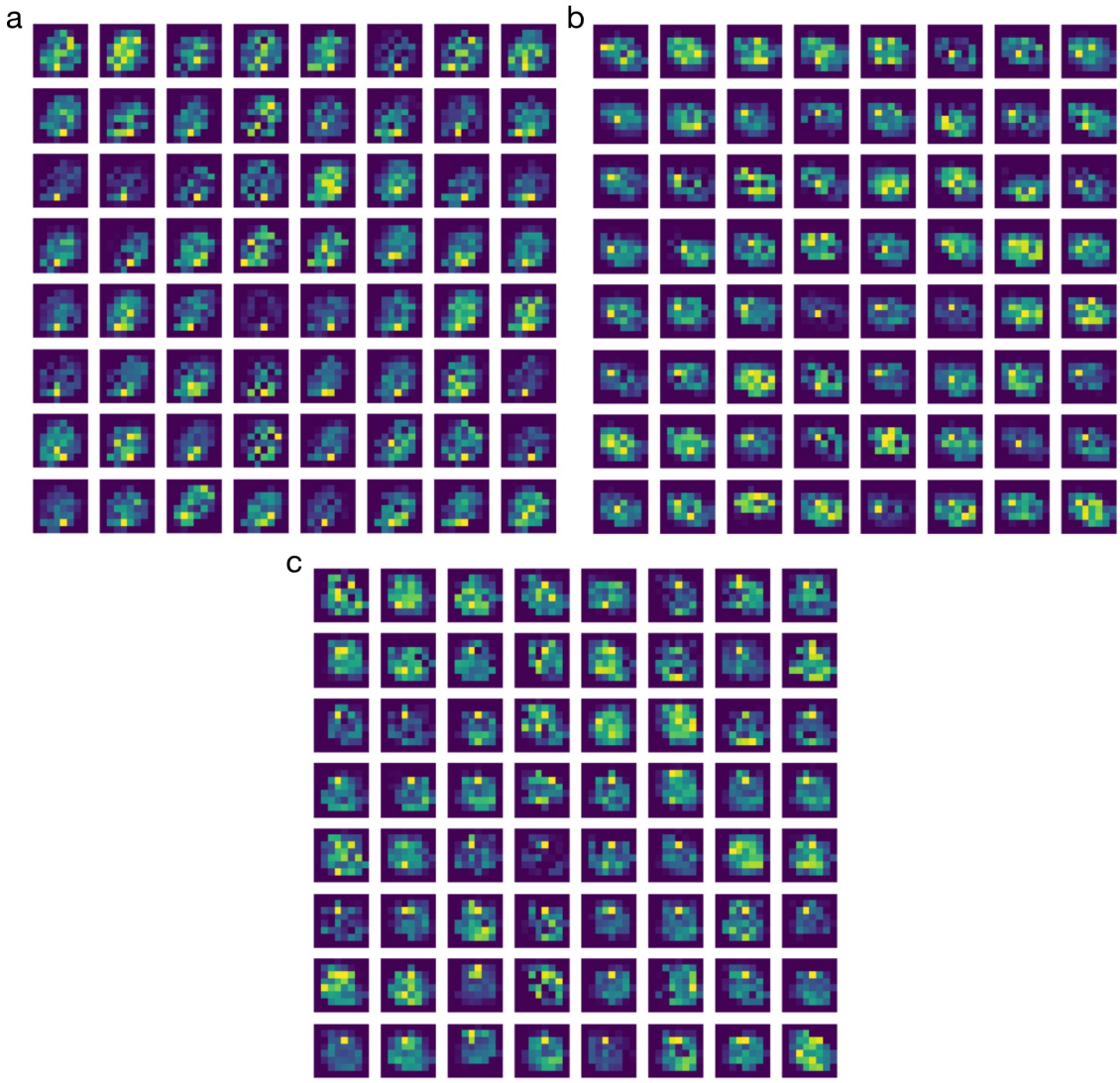


Figure A.2: Feature maps of Cpaaa migration at one of the early migration timesteps in Embryos 1 (a), 2 (b), and 3 (c). Yellow represents highest pixel value and black lowest. All the maps reveal the size and orientation of the corresponding embryo. The yellow spots in half of the feature maps (32-39 out of 64) indicate the location of the migrating cell.

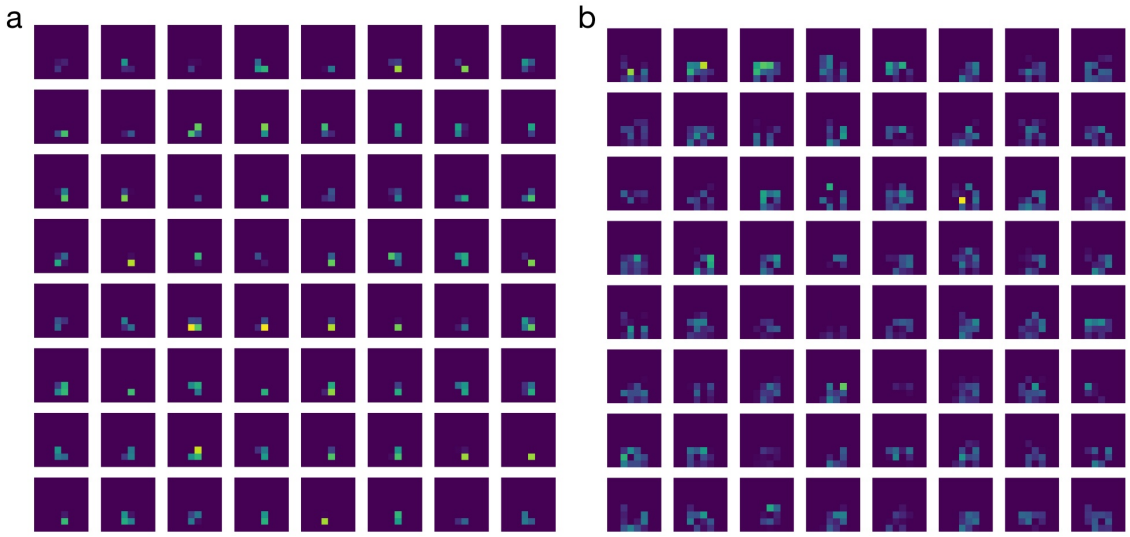


Figure A.3: Examples of individual difference maps at a single timestep. Each of the 64 difference maps is shown for an ablation experiment after ablating Cpaaa (a) and Cpaaa's neighbor cells (b).

A.4 Examples of individual difference maps at a single timestep

Examples of individual difference maps (the difference between the original feature maps and those after ablating certain cells) are presented in Fig. A.3.

A.5 The number of pixels in the effective area of the summary difference map at each timestep in three embryos

The number of pixels in the effective area of the summary difference map at each timestep are presented in Fig. A.4.

A.6 Distribution for normalized minimal distance between neighbor cells

Distribution for normalized minimal distance between neighbor cells are presented in Fig. A.5.

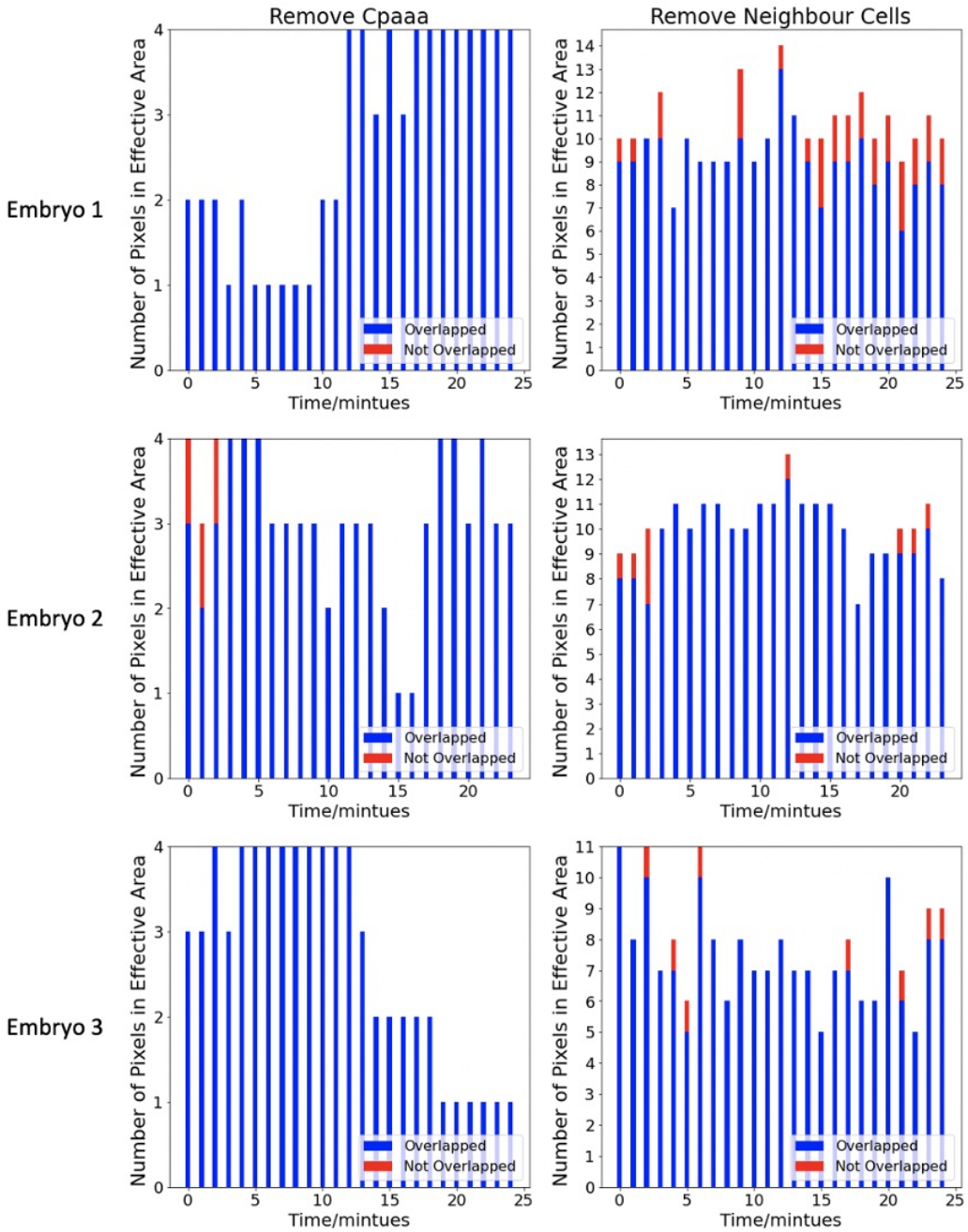


Figure A.4: The number of pixels in the effective area of the summary difference map at each timestep in three embryos. Bars colored with blue/red represent the number of pixels where the ablated cells overlapped/not overlapped with the effective area.

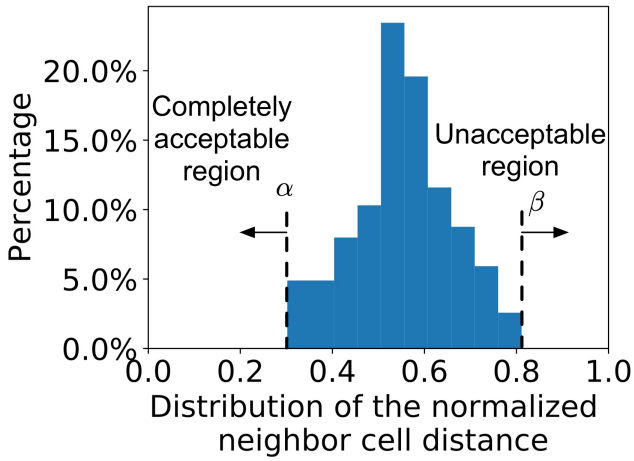


Figure A.5: Distribution for normalized minimal distance between neighbor cells. Statistics compiled from a collection of 50 wild-type embryos. X-axis indicates the level of cell overlapping. $\alpha = 0.3$ and $\beta = 0.8$ indicate the completely acceptable/unacceptable overlapping values, respectively.

Vita

Zi Wang received his bachelor's and master's degrees in Electrical Engineering from Shandong University, China in 2012 and 2015, respectively. In August 2015, he entered the doctoral program in computer engineering at the Min H. Kao Department of Electrical Engineering and Computer Science (EECS) at the University of Tennessee, Knoxville.

Zi's research interests include machine learning and developmental system modeling. Specifically, he is interested in neural network compression technologies such as network pruning and knowledge distillation, and modeling cell behaviors in the developmental system with deep reinforcement learning.