Efficient and Online Deep Learning through Model Plasticity and Stability

by

Xiaocong Du

A Dissertation Presented in Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy

Approved October 2020 by the
Graduate Supervisory Committee:

Yu (Kevin) Cao, Chair
Jae-sun Seo
Chaitali Chakrabarti
Deliang Fan

ARIZONA STATE UNIVERSITY

December 2020

ABSTRACT

The rapid advancement of Deep Neural Networks (DNNs), computing, and sensing technology has enabled many new applications, such as the self-driving vehicle, the surveillance drone, and the robotic system. Compared to conventional edge devices (*e.g.* cell phone or smart home devices), these emerging devices are required to deal with much more complicated and dynamic situations in real-time with bounded computation resources. However, there are several challenges, including but not limited to efficiency, real-time adaptation, model stability, and automation of architecture design.

To tackle the challenges mentioned above, model plasticity and stability are leveraged to achieve efficient and online deep learning, especially in the scenario of learning streaming data at the edge:

First, a dynamic training scheme named Continuous Growth and Pruning (CGaP) is proposed to compress the DNNs through growing important parameters and pruning unimportant ones, achieving up to 98.1% reduction in the number of parameters.

Second, this dissertation presents Progressive Segmented Training (PST), which targets catastrophic forgetting problems in continual learning through importance sampling, model segmentation, and memory-assisted balancing. PST achieves state-of-the-art accuracy with 1.5X FLOPs reduction in the complete inference path.

Third, to facilitate online learning in real applications, acquisitive learning (AL) is further proposed to emphasize both knowledge inheritance and acquisition: the majority of the knowledge is first pre-trained in the inherited model and then adapted to acquire new knowledge. The inherited model's stability is monitored by noise injection and the landscape of the loss function, while the acquisition is realized by importance sampling and model segmentation. Compared to a conventional scheme, AL reduces accuracy drop by >10X on CIFAR-100 dataset, with 5X reduction in latency per training image and 150X reduction in training FLOPs.

Finally, this dissertation presents evolutionary neural architecture search in light of model stability (ENAS-S). ENAS-S uses a novel fitness score, which addresses not only the accuracy but also the model stability, to search for an optimal inherited model for the application of continual learning. ENAS-S outperforms hand-designed DNNs when learning from a data stream at the edge.

In summary, in this dissertation, several algorithms exploiting model plasticity and model stability are presented to improve the efficiency and accuracy of deep neural networks, especially for the scenario of continual learning.

ACKNOWLEDGMENTS

First and foremost, I would like to thank my brilliant advisor, Dr. Yu (Kevin) Cao, for his steadfast patience, motivation, and guidance throughout my doctorate studies. Dr. Cao is an extraordinary researcher and taught me the importance of persistence in successful research.

I would also like to take this opportunity to thank my committee members, Dr. Chaitali Chakrabarti, Dr. Deliang Fan, and Dr. Jae-sun Seo, for their strong support and constructive feedback throughout this dissertation. I am also grateful to Dr. Frank Liu (Oak Ridge National Lab) for his guidance and suggestions on my work.

I was fortunate to have an excellent set of friends, colleagues, and collaborators: Ang Chen, Pai-Yu Chen, Xueyao Cao, Sai Gorthy, Zhezhi He, Deepak Kadetotad, Siyu Liu, Yufei Ma, Abinash Mohanty, Devyani Patra, Yuxu Wu, Zihan Xu, Shihui Yin, and many others. Special thanks to Bhargav and Arun for their outstanding mentorship during my internship at Facebook, Inc. I cannot acknowledge all by name, but let that not diminish my gratitude for all the people who supported and helped me during my doctoral study.

I have been blessed with a wonderful family. Special thanks to my amazing parents, Wenhua and Yingwen, and my loving fiance Zheng, for their unconditional love and support. Without their support, encouragement, and countless sacrifices, none of this would have been possible. It is to them I dedicate this dissertation.

The journey towards a doctoral degree is long and arduous: my schedule is not limited to a stereotypical '9-to-5' but becomes 'whenever you find time to think about your research and get your work done'. Fortunately, my efforts paid off. Besides research skills and domain knowledge, I have also gained systematic methods to approach problems and mature mindset to confront a new challenge. I believe that all of these will be precious wealth throughout my life.

TABLE OF CONTENTS

Page

LIST OF TABLES viii

LIST OF FIGURES ix

CHAPTER

1 INTRODUCTION 1

1.1 Motivation 1

1.2 Thesis Contribution 2

1.3 Thesis Outline 5

2 BACKGROUND 6

2.1 Development of Machine Learning 6

2.2 Dataset and Framework 10

3 CGAP: CONTINUOUS GROWTH AND PRUNING FOR EFFICIENT DEEP LEARNING 13

3.1 Introduction 13

3.2 Previous Work 17

3.2.1 Destructive Methods 18

3.2.2 Constructive Methods 18

3.2.3 Orthogonal Methods 19

3.3 Saliency Score 19

3.3.1 Terminology 20

3.3.2 Saliency Score 21

3.4 CGaP Methodology 22

3.4.1 Growth Phase 23

3.4.2 Pruning Phase 27

3.5 Algorithmic Experiments 29

3.5.1 Training Setup 29
3.5.2 Performance Evaluation 31
3.5.3 Visualization of the Dynamic Structures 32
3.5.4 Validating the Saliency-based Growth 35
3.6 Experiments on FPGA Simulator 36
3.6.1 Overview of the FPGA Simulator 36
3.6.2 Results from FPGA Performance Model 37
3.7 Discussion 38
3.8 Conclusion and Future Work 42
4 SINGLE-NET CONTINUAL LEARNING WITH PROGRESSIVE SEGMENTED TRAINING 44
4.1 Introduction 44
4.2 Related Work 48
4.3 Method 50
4.3.1 Overview of PST 50
4.3.2 Memory-assisted Training and Balancing 52
4.3.3 Importance Sampling 53
4.3.4 Model Segmentation and Reinforcement 55
4.4 Accuracy: learning streamed tasks 55
4.4.1 In-depth Analysis 57
4.4.2 Extensive Results 59
4.5 Computation Cost: Learning at the Edge 61
4.5.1 Simulated Results 61
4.6 Ablation Study and Discussion 64

4.6.1 Analysis of Each Component in PST 64
4.6.2 Memory Budget 65
4.7 Conclusion 66
5 ONLINE KNOWLEDGE ACQUISITION WITH SELECTIVE INHERITED MODEL 67
5.1 Introduction 67
5.2 Preliminaries 71
5.2.1 Terminology 71
5.2.2 Conventional Approach of Continual Learning 72
5.2.3 Difference from Transfer Learning 73
5.2.4 Biology Background: Moravec's Paradox 74
5.3 Acquisitive Learning 74
5.3.1 Model Inheritance 74
5.3.2 Knowledge Acquisition 76
5.4 Experimental Results 77
5.4.1 Experiment Setup 77
5.4.2 Amount of Inherited Knowledge 79
5.4.3 Quality of the Inherited Model 80
5.4.4 Learning from a Data Stream with AL 83
5.4.5 Computation Cost and FPGA Prototyping 84
5.4.6 FPGA Demonstration 86
5.5 Conclusion 88
6 EVOLUTIONARY NAS IN LIGHT OF MODEL STABILITY FOR ACCURATE CONTINUAL LEARNING 90

CHAPTER Page

6.1 Introduction 90

6.2 Related work 93

6.3 Methodology 94

6.3.1 Search Space 94

6.3.2 Fitness Evaluation 97

6.3.3 Search Strategy 98

6.3.4 Continual Learning at the Edge 101

6.4 Experimental Results 103

6.4.1 Experiments Setup 103

6.4.2 Effectiveness of ENAS-S 104

6.4.3 Effectiveness of the Proposed Fitness Score 104

6.4.4 Architecture Evolution 108

6.5 Conclusion 109

7 SUMMARY 110

REFERENCES 111

LIST OF TABLES

Table Page

3.1 Evaluation of the Performance on MNIST. 32

3.2 Evaluation of the Performance on CIFAR-100. 32

3.3 Evaluation of the Performance on SVHN. 32

3.4 Evaluation of the Performance on CIFAR-10. 33

3.5 The Impact of Various Structures and and Sizes of the Initial Seed of VGG-19. 40

4.1 With Increasing Data Trained in the Cloud, PST Effectively Mitigates Forgetting. Note That in This Experiment, the Network Size Is Much Smaller than That in Section 4.4.2. 63

4.2 Switching off Different Components of PST Leads to Accuracy Drop to Different Extents. In the Table, Negative Numbers Indicate Accuracy Drop, *e.g.* -0.32 Means 32% Accuracy Drop. Removing Significance Sampling or Model Segmentation Leads to Significant Accuracy Drop, While Removing Memory Leads to Small Accuracy Drop. It Shows That Significance Sampling and Model Segmentation Are Indispensable Steps for PST, and Memory-assisted Balancing Is Supplementary. 64

5.1 Acquisition Capacity for Different Models Shown in Figure 5.6. CIFAR-10 '9+1' Is Used Here. 82

5.2 Required Throughput (TFLOPs/Second) and the Number of Hardware Platforms* Needed to Learn Various Number of Classes with AL. 87

6.1 Block Library in ENAS-S. 96

LIST OF FIGURES

Figure Page

1.1 Emerging Intelligent Systems Are Required to Deal with Dynamic Situations in Real-time. 2

1.2 Road-map Surrounding Efficient and Online Deep Learning through Model Plasticity and Stability. 3

2.1 Linear Classification Is Not Able to Solve the XOR Problem 7

2.2 Multi-layer Perceptron with Loss Gradient Back-propagated. 8

2.3 Architecture of LeNet-5, a Small but Clever CNN. 8

2.4 Architecture of AlexNet 9

2.5 Architecture of VGG-16. 9

2.6 Architecture of GoogLeNet. 9

2.7 Architecture of ResNet and the Residual Unit. 10

2.8 Examples of Commonly Used Datasets. 11

3.1 Energy Breakdown for Modern DNN Structures, Results from Simulation by the FPGA Performance Model. Due to the Redundancy in Parameters, Multiply-accumulator (MAC) and External Memory (DRAM) Access Dominate the Energy Consumption. 14

3.2 The Proposed CGaP Scheme. CGaP Starts the Training from a Seed Network Instead of an Over-parameterized One, Gradually Grows Important Learning Units During the Training and Reaches Peak Capacity at the End of Growth, Then Prunes Secondary Filters and Neurons to Generate an Inference Model with Structured Sparsity and Up-to-date Accuracy. 14

Figure Page

3.3 Illustration of Two-step Growth in Conv-layers. The Growth Phase Follows a Two-step (Growing and Mapping) Procedure. After the Filter $W^j_{l,picked}$ (Green) Is Picked and Split Aside, Giving Birth To $W^j_{l,newborn}$ (Orange), the Projected Input-wise Filter, $W^j_{l+1,projected}$ (Blue) in Layer $l+1$, Is as Well Split aside, Generating $W^j_{l+1,mapped}$ (Black). . 25

3.4 Illustration of Two-step Growth in FC-layers. First, Fan-out Weights $w^j_{L,Newborn}$ (Orange) Is Added, Then Fan-in Weights $\mathbf{W}^j_{l-1,mapped}$ (Black) Form the Connections from the Newborn Neuron to All Neurons in Layer $l-1$. 27

3.5 Number of Parameters During Training, Plotted at the End of Each Epoch. In the Beginning, the Model Size Increases Gradually Due to the Growth. After the Growth Ends and Several Epochs of Training on the Peak Model, One Drop Can Be Observed after the First Pruning. There Are Several Iterations of Pruning at a Frequency of 1. 33

3.6 The VGG-19 Structures Learned by CGaP on CIFAR-100 and SVHN Datasets. The Shared Y-axis for Three Sub-figures Is the Number of Parameters of the Model. 34

3.7 Saliency-based Growth Outperforms Random Growth. The Loss Is Monotonically Decreasing from Epoch 0 to 220 with Small Glitches. Here We Zoomed in from Epoch 120 to 220 to Show the Loss at the End of the Training. 35

3.8 Estimation on FPGA Performance Model. 37

Figure Page

3.9 Visualization of the Filters in conv1_1 in VGG-19 on CIFAR-100 at Four Specific Moments (a-d). Inside Each Figure, the Top Bar Is CGaP Model and the Bottom Bar Is Baseline Model. X-axis Is the Index of Output-wise Weights and Y-axis Is the Index of Input-wise Weights. . . . 39

3.10 A Larger Seed Leads to a Larger Final Model but Fewer Iterations in the Growth Phase. 41

4.1 The Flow Chart of Progressive Segmented Training (PST). (A) We Allow the Current Task T_i and a Memory Set to Update the Free Parameters Θ_{free} (in Light Blue) in the Network While Sharing Fixed Parameters θ_{Fixed} (in Grey) Learned from Previous Tasks. The Fixed-size Memory Set Is Used to Keep the Balance of Training among Various Tasks. (B) We Sort and Select Important Parameters $\Theta_{important}$ (in Dark Blue) for Task T_i, and Reinforce Them by Retraining. These Important Parameters Are Kept Frozen and Will Not Be Updated by Future Tasks. Different from PackNet And Piggyback, the Secondary Parameters (in Light Blue) Are Not Pruned in PST. Instead, New Tasks Will Start from Secondary Parameters and Update the Network, Which Is Essential to Achieve Single-head Classification. For a New Task T_{i+1}, the above Training Routine Repeats in (c) and (d), so on and so Forth. 51

4.2 Comparison of Weight Distribution Between Pruning-based Approaches and Our PST. Pruning-based Approaches Lose Prior Knowledge Due to Pruning, and PST Preserves Prior Knowledge by Segmentation. 57

4.3 The Learning Curve of 2 Tasks on CIFAR-10 with Each Step Annotated. 58

Figure Page

4.4 Single-head Overall Accuracy on CIFAR-100 When Incrementally Learning 20, 10, 5, 2 Tasks in a Sequence. PST Has the Best Accuracy of 2 Tasks and the Second Best Accuracy of 5, 10, 20 Tasks. Though iCaRL.MC Has Better Accuracy than PST, It Requires >24× Computation Cost than PST (See Figure 4.6 for Details) 60

4.5 (a) Single-head Accuracy and (b) Multi-head Accuracy of the First Task T_1 over Time When the Model Is Trained with a Sequence of 20 Tasks on CIFAR-100. 61

4.6 Comparison of the Computation Cost of PST and the Regularization Method. In the Scenario of Edge Learning, More than 24× Reduction in FLOPs for the Weight Update Path (Top), and 1.5× Reduction for Complete Path (Bottom) Are Achieved. 62

4.7 Overall Single-head Accuracy When Incrementally Learning 10 Tasks under Different Memory Budget. 65

5.1 When Learning from a Data Stream of CIFAR-10, Conventional Continual Learning Suffers Catastrophic Forgetting, While the Proposed Acquisitive Learning Successfully Mitigates Such Forgetting by >6X on CIFAR-10. A Well Selective Inherited Model , Knowledge Acquisition and Memory Rehearsal All Contribute the Accurate Learning. Among Them, the Quality of the Inherited Model Is More Vital than the Amount of Memory Used to Replay. Model 1 Refers to ResNet-56 with Better Landscape; Model 2 Refers to ResNet-56-NS with Worse Landscape. 68

Figure Page

5.2 The Flow of Acquisitive Learning Emphasizes the Importance of Both Knowledge Inheritance and Acquisition. 69

5.3 The Main Reason of Catastrophic Forgetting Is the Drift in the Feature Space. Left: Visualizing the Euclidean Distance Between $\varphi(\mathcal{X})$ Of Each Input Image and the Feature Center (*i.e.* the Normalized $\varphi(\mathcal{X})$ Of All the Input Images) after Learning 10 Classes from CIFAR-100 with ResNet-56. Wrongly Classified Samples Are Relatively Further from the Feature Center. Right: After Learning Another 10 Classes from CIFAR-100, the Feature Center Drifts so That the Correlation Between Euclidean Distance and Classification Is Deteriorated. 72

5.4 Accuracy Drop Is Minimized with the an Increasing Amount of Knowledge in the Inherited Model. Top: VGG-16 on CIFAR-10 Dataset. Bottom: ResNet-56 on CIFAR-100 Dataset. 'X+Y' Means That X Classes Are Pre-trained and Y Classes Are Learned as the New Acquisition. There Is No Overlapping of Classes in X and Y. 79

5.5 After injecting noise with the same α, more accuracy drop is observed for less stable models, and vice versa. This ranking is consistent with that from the landscape visualization and roughness measurement shown in Figure 5.6. 80

5.6 Landscape Visualization of the Loss Function for 6 Models. Shallow Models (like VGG-16) Have Smooth Landscapes. Deep Models with Shortcuts Has Smoother Landscapes than the One Without Shortcuts. 81

Figure Page

5.7 Learning Curve for ‘9+1’ Experiment on CIFAR-10 with Two Models. Because of the Smoother Landscape of ResNet-56, Its Acquisition on New Task Is Better than ResNet-56-NS. 82

5.8 The Comparison of Overall Accuracy Between Conventional Continual Learning and the Proposed Acquisitive Learning When (a) Incrementally Learning 1 Class in a Sequence on CIFAR-10 (B) Incrementally Learning 10 Classes in a Sequence on CIFAR-100. In the Figure, ‘I9’ Means That AL Starts Training from a Model That Is Pre-trained on 9 Classes. Similarly, ‘I50’ Means the Inherited Model Is Pre-trained on 50 Classes. 84

5.9 Comparison on Computation Cost Between Conventional Continual Learning and Proposed Acquisitive Learning. 85

5.10 FPGA Demonstration of PST Algorithm. 87

6.1 ENAS-S Leverages Model Stability to Search for an Optimal Architecture Through the Evolutionary Algorithm. Such an Architecture Is Pre-trained, Deployed at the Edge, and Then Exposed to a Few New Observations. The Edge Device Is Required to Acquire New Knowledge as Well as Remembering the Inherited Knowledge Successfully. 92

6.2 Structure of Each Block. 95

6.3 ENAS-S Uses a Module-based, Single-path Backbone Which Makes the Generated Architectures More Efficient to Be Deployed and Trained at the Edge. 96

6.4 Flowchart of the Evolutionary Algorithm. 99

Figure Page

6.5 Visualization of the Evolution Process over Generations. It Is Observed That along with the Evolution, the *Acc* of the Population Is Getting Higher and ΔAcc of the Population Is Getting Lower. Meanwhile, Their Standard Deviation Is Reducing Because the Searching Is Converging over Time. 105

6.6 Loss Landscape of the Model Evolves from Roughness to Smoothness over Generations, Indicating the Improvement in Model Stability. 106

6.7 Comparison among ENAS-S Architectures and Handcrafted Models in the Scenario of *Continual Learning at the Edge*. ENAS-S Architectures Outperform Hand-designed Models in Mitigating Catastrophic Forgetting with Much Small Model Size (as Noted in Figure). 107

6.8 Left: Average Number of Each Component in an Individual; Right: The Normalized Trend over Evolutionary Generations. 108

Chapter 1

INTRODUCTION

1.1 Motivation

Deep neural networks have been used in various applications, such as smartphones, self-driving cars, translation applications, etc. The advantage is that DNNs are accurate and general, but on the other hand, they are complicated, computation, and memory intense. Meanwhile, traditionally, there are two ends from the hardware perspective: edge devices and cloud center, as shown in Figure 1.1. Edge devices, such as cellphones and smartwatches, are smart but usually have limited power budgets. On the other side, the giant cloud data center is equipped with a much larger computing and power budget to handle the training with big data. There is another category of intelligent systems emerging these years, such as self-driving cars, drones, and robots. These emerging systems are usually equipped with several GPUs and a decent capacity of computing. They mainly perform inference, but sometimes they require the capability of online training. These emerging devices are required to deal with much more complicated and dynamic situations than traditional edge devices. In some scenarios, these intelligent systems have to quickly pick up a task, learn it online in a continuous manner, and react immediately, rather than sending the data back to the cloud and waiting for the cloud to handle the training. Thus, the capability of continual learning is a necessary attribute for such an intelligent learning system.

However, there are several challenges to achieve successful continual learning:

- Efficiency. It is vital to achieve smaller but still accurate models so that the

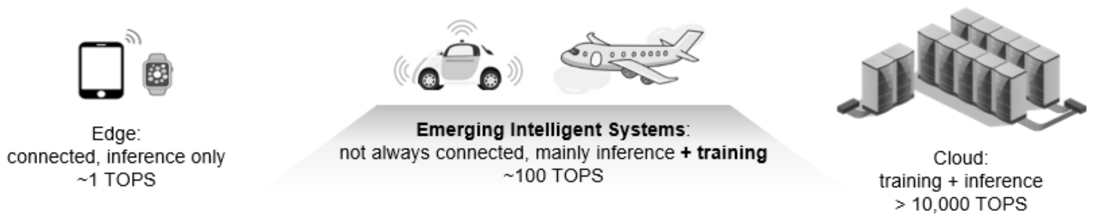

Edge:
connected, inference only
~1 TOPS
Emerging Intelligent Systems:
not always connected, mainly inference + training
~100 TOPS
Cloud:
training + inference
> 10,000 TOPS

Figure 1.1: Emerging Intelligent Systems Are Required to Deal with Dynamic Situations in Real-time.

deployment of DNN on edge devices would be more efficient.

- Real-time adaptation. The learning system should update its knowledge (*i.e.* network parameters) according to the new coming data stream in real-time.
- Model stability. A trained DNN model should be stable to preserve the previously acquired knowledge when learning new knowledge online. In other words, the network parameters should not be overwritten or interfered by learning new data.
- Automation of DNN design. The existing DNN architectures are designed by experts, but in practice, most end users have limited expertise in architecture design. So it is critical to design DNN architectures according to various applications and scenarios automatically.

1.2 Thesis Contribution

This dissertation focuses on the aforementioned challenges through two attributes of DNNs: model plasticity and model stability. Model plasticity refers to the model's architecture adaptation, including constructive and destructive ways. Model stability represents how well the model can preserve previously encoded knowledge over time. In this dissertation, random noise is injected into the pre-trained model as the per-

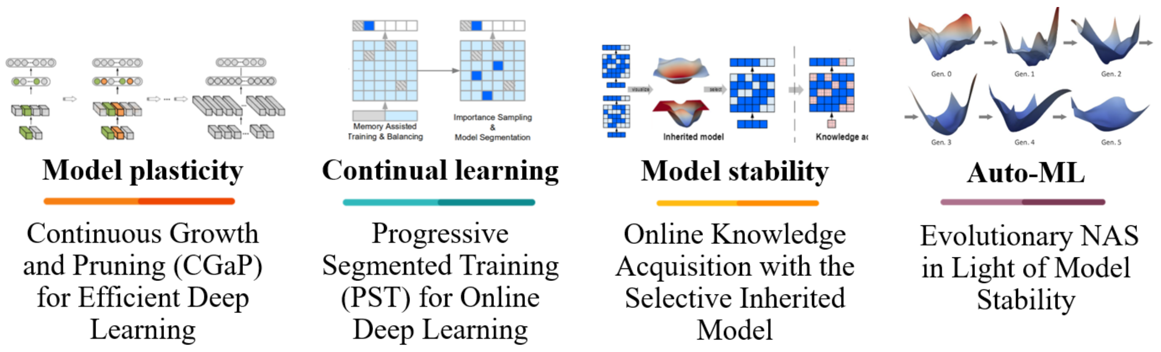

Memory Assisted Training & Balancing
Importance Sampling & Model Segmentation
Inherited model
Knowledge ac
Gen. 0
Gen. 1
Gen. 2
Gen. 3
Gen. 4
Gen. 5
Model plasticity
Continual learning
Model stability
Auto-ML
Continuous Growth and Pruning (CGaP) for Efficient Deep Learning
Progressive Segmented Training (PST) for Online Deep Learning
Online Knowledge Acquisition with the Selective Inherited Model
Evolutionary NAS in Light of Model Stability

Figure 1.2: Road-map Surrounding Efficient and Online Deep Learning through Model Plasticity and Stability.

turbation to infer the model stability and then loss landscape, and its corresponding roughness score are obtained to validate the stability. A model with better stability has less accuracy drop after perturbation, a smoother and flatter landscape and a lower roughness score. All these techniques center around exploiting model plasticity and model stability, shown in Figure 1.2. The contributions of this thesis are:

- A training scheme, called Continuous Growth and Pruning (CGaP) that leverages model plasticity, is proposed to achieve small yet accurate DNN models so that the deployment of DNNs on hardware would be more efficient. CGaP starts from training a small network seed, then literally executes continuous growth by adding important learning units and finally prunes secondary ones for efficient inference. The inference model generated from CGaP is sparse in the structure, largely decreasing the inference power and latency when deployed on hardware platforms. With popular DNN structures on representative datasets, the efficacy of CGaP is benchmarked by both algorithmic simulation and architectural modeling on Field-programmable Gate Arrays (FPGA).

- A training algorithm named progressive segmented training (PST) is proposed to mitigate catastrophic forgetting problem in continual learning. Leveraging the redundant capacity of a single network, model parameters for each task are separated into two groups: one important group which is frozen to preserve current knowledge, and secondary group to be saved (not pruned) for a future learning. Without additional regularization, the simple yet effective approach successfully incorporates multiple tasks and achieves state-of-the-art accuracy in the single-head evaluation on CIFAR-10 and CIFAR-100 datasets. Moreover, the segmented training significantly improves computation efficiency in continual learning at the edge.

- Considering a real-time situation, continual learning may not be used to learn everything from scratch. Thus, a novel paradigm named acquisitive learning (AL) is proposed to enable online learning at the edge based on a prepared inherited model. Unlike previous approaches that focus only on model adaptation, AL emphasizes the importance of both knowledge inheritance and acquisition: the knowledge is first pre-trained and selected in the cloud (the inherited model and selection) and then adapted to new knowledge (the acquisition). The inherited model's stability is monitored by the landscape of the loss function, while the acquisition is realized segmented training. The combination of knowledge inheritance and acquisition reduces accuracy drop by >10X on the CIFAR-100 dataset. Furthermore, AL benefits edge computing with 5X reduction in latency per training image on FPGA prototype and 150X reduction in training FLOPs.

- An evolutionary neural architecture search in light of model stability (ENAS-S) is proposed to search for an optimal, hardware-friendly inherited architecture to achieve accurate continual learning at the edge. On CIFAR-10 and CIFAR-100, experiments present that ENAS-S achieves competitive architectures with lower catastrophic forgetting and smaller model size when learning from a data stream than handcrafted DNNs.

1.3 Thesis Outline

The outline of this thesis is as below:

Chapter 2 introduces the background of machine learning, deep learning, deep neural networks, and datasets.

Chapter 3 describes the training scheme, CGaP, which grows the network during training and achieves smaller DNN models with structured sparsity. The content of this chapter is based primarily on [1].

Chapter 4 presents the training algorithm, PST, which uses model segmentation to prevent catastrophic forgetting. The content of this chapter is based primarily on [2].

Chapter 5 describe the novel brain-inspired paradigm named acquisitive learning (AL), which is proposed to improve accuracy and efficiency for continual learning at the edge via model stability. Furthermore, the detailed definition and visualization of model stability is presented in this chapter. The content of this chapter is based primarily on [3–5].

Chapter 6 develops an evolutionary neural architecture search (ENAS) algorithm that emphasizes the *Stability* of the inherited model, namely ENAS-S. ENAS-S aims to find optimal architectures for accurate continual learning at the edge.

Chapter 7 concludes the dissertation and provides some thoughts on future directions.

Chapter 2

BACKGROUND

2.1 Development of Machine Learning

Back in 1949, the first step to machine learning was proposed by D.O.Hebb *et al.* and named *Hebbian Learning* [6]. It was based on a neuro-psychological learning formulation. By building a connection between neurons (nodes), it memorizes the stimulus. Cells that fire together, wire together.

In 1952, Arthur Samuel from IBM invented a program that could play Checkers. It could obtain experience during the process of playing, just like human learning. Arthur Samuel then defined *machine learning* (ML) as a field of study that gives a computer the ability without being explicitly programmed [7].

In 1957, an exciting discovery came out, which was called *Perceptron.* It was proposed by F. Rosenblatt. Perception is still in use nowadays as it is more practical than Hebbian Learning rules. The perceptron is designed to illustrate some of the fundamental properties of intelligent systems in general, without becoming too deeply enmeshed in the special and usually unknown conditions which hold for particular biological organisms [8].

In 1960, B.Widrow *et al.* [9] brought out something called *Delta Learning* that is then used as a practical procedure for perceptron training. It is also known as the Least Square problem. A combination of those two ideas creates a good linear classifier.

However,in 1969 M. Minsky [10] doubt that the perceptron learning cannot classify XOR problem (Figure 2.1 [11]) with linear classification. The machine learning stopped development until 1980.

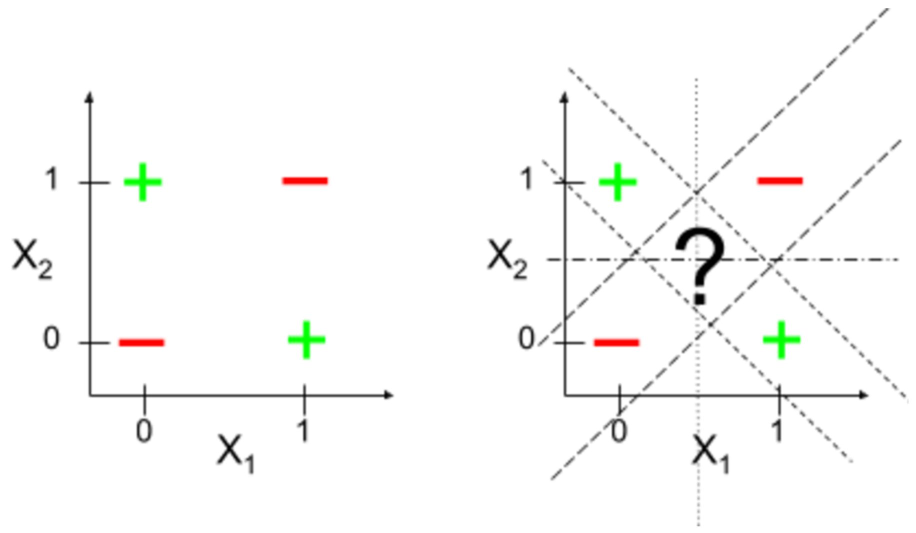
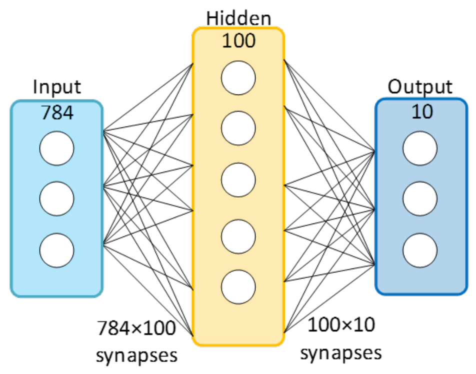


Figure 2.1: Linear Classification Is Not Able to Solve the XOR Problem

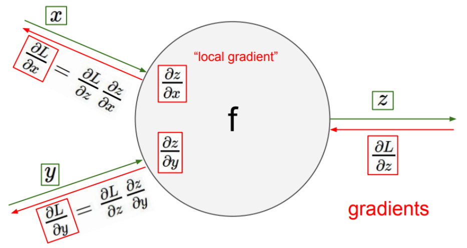
In 1981, P.J. Werbos *et al.* brought out *multi-layer perceptron* (MLP) with *back-propagation* (BP) [12]. Then in 1986, with the publishing of a paper by Rumelhart, Hinton, and Williams [13] *et al.*, the importance of the back-propagation was appreciated by the machine learning community at large. Till now, back-propagation is still the main ingredient in machine learning algorithms. Figure 2.2(a) and Figure 2.2(b) provide examples of MLP and BP.

Another important discovery is called *Neocognitron* [14], a hierarchical and multi-layered artificial neural network proposed by Kunihiko Fukushima *et al.* in the 1980s. It has been used for handwritten character recognition and other pattern recognition tasks and served as the inspiration for convolutional neural networks (CNNs). The inspiration for convolution and pooling came from this.

In 1998, researchers lead by Yann LeCun *et al.* proposed a seven-layer convolution-based network: LeNet-5 [15]. LeNet-5 is a convolutional network designed for handwritten and machine-printed character recognition. LeNet-5 improved the accuracy



(a) An example of multi-layer perceptron structure.



x
$\frac{\partial L}{\partial x} = \frac{\partial L}{\partial z}\frac{\partial z}{\partial x}$
"local gradient"
$\frac{\partial z}{\partial x}$
f
$\frac{\partial z}{\partial y}$
z
$\frac{\partial L}{\partial z}$
y
$\frac{\partial L}{\partial y} = \frac{\partial L}{\partial z}\frac{\partial z}{\partial y}$
gradients

(b) An example of error/loss/gradient back-propagation.

Figure 2.2: Multi-layer Perceptron with Loss Gradient Back-propagated.

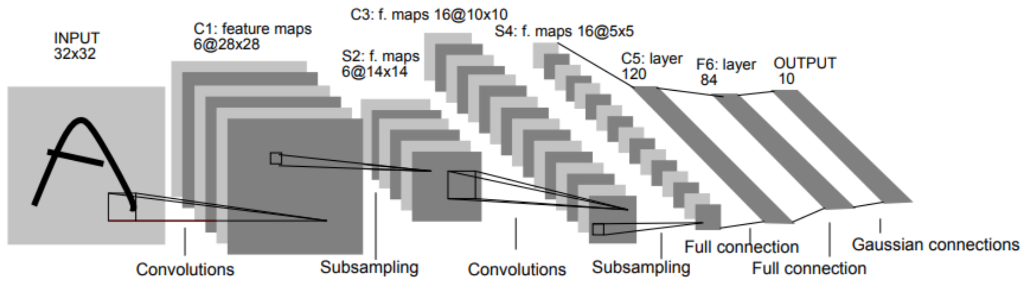


Figure 2.3: Architecture of LeNet-5, a Small but Clever CNN.

of handwritten digits to above 99%. Fig 2.3 [15] shows the structure of LeNet-5.

However, convolutional networks were not brought to the forefront because of the proposal of *Support Vector Machines* (SVM) [16] by Vapnik and Cortes in 1995 with very strong theoretical standing and empirical results. This was the time separating the ML community into two crowds as neural network (NN) or SVM advocates.

After decades of development (several useful research such as decision tree [17], random forests [18], Adaboost *et al.* [19], and ways to avoid gradient vanishing), AlexNet [20] from the Hinton group won the prize with huge advantage on ImageNet dataset [21] in 2012, raising an upsurge of deep learning. AlexNet (Figure 2.4 [20]) is

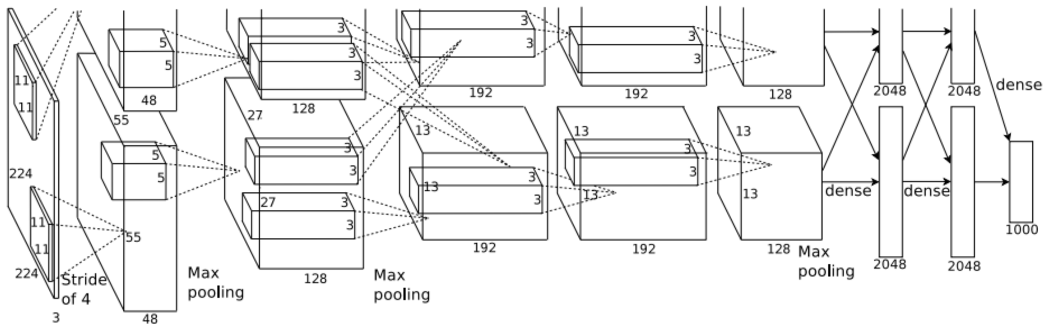


Figure 2.4: Architecture of AlexNet

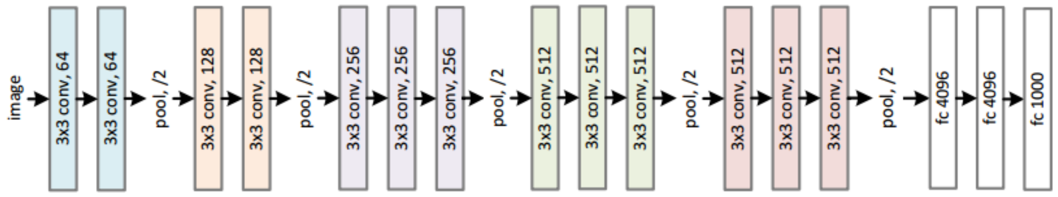


Figure 2.5: Architecture of VGG-16.

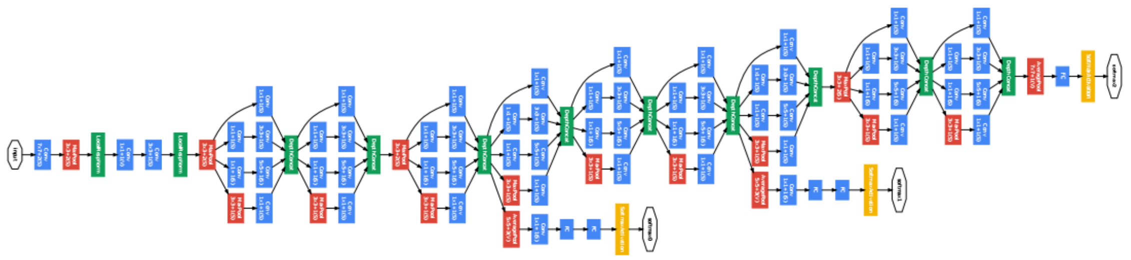
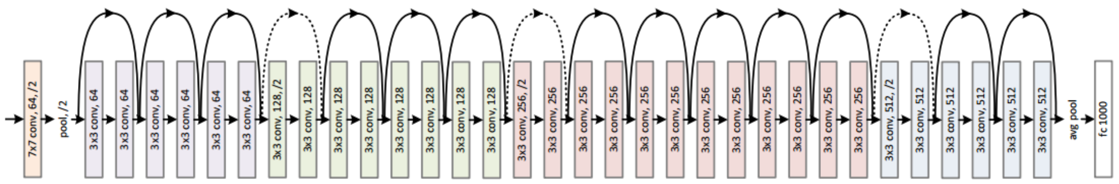


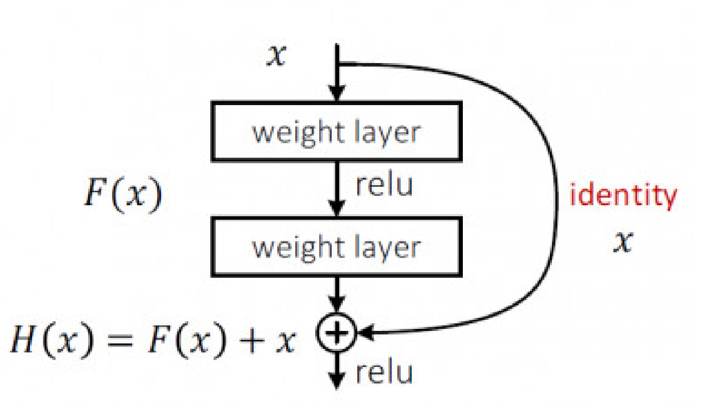
Figure 2.6: Architecture of GoogLeNet.

a well-designed CNN, with ReLU, dropout and other techniques, in a larger size and deeper. The idea of *deep neural networks* (DNNs) has developed since then. Several deep and accurate network such as VGG (Figure 2.5) [22], GoogLeNet (Figure 2.6) [23] and so on, appeared.

In 2016, Kaiming He *et al.* proposed ResNets (Figure 2.7) [24], which feed the output of two successive convolutional layers and also bypass the input to the next



(a) Architecture of ResNet



(b) Residual unit

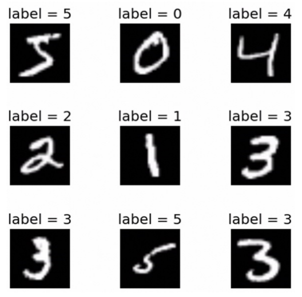
Figure 2.7: Architecture of ResNet and the Residual Unit.

layers via skip-connections. ResNets make deep learning way deeper and successfully prevent vanishing gradient. In Chapter 6, this dissertation also validates that ResNet skip-connections also improve model stability and smoothness of loss landscape.

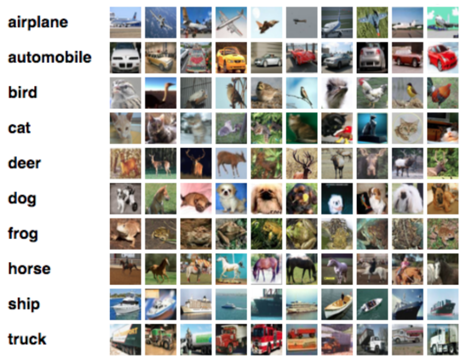
Some other algorithms that usually apply to unsupervised learning and reinforcement learning are neglected here as they are not tightly related to this dissertation.

2.2 Dataset and Framework

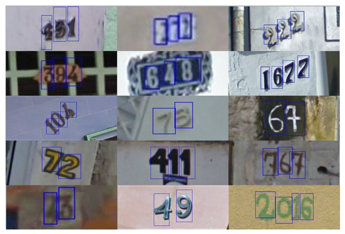
Different datasets for a variety of machine learning tasks are collected and labeled to test the model's performance. The commonly used datasets include MNIST, CIFAR-10, CIFAR-100, SVHN, and ImageNet for image classification, as shown in Figure 2.8.



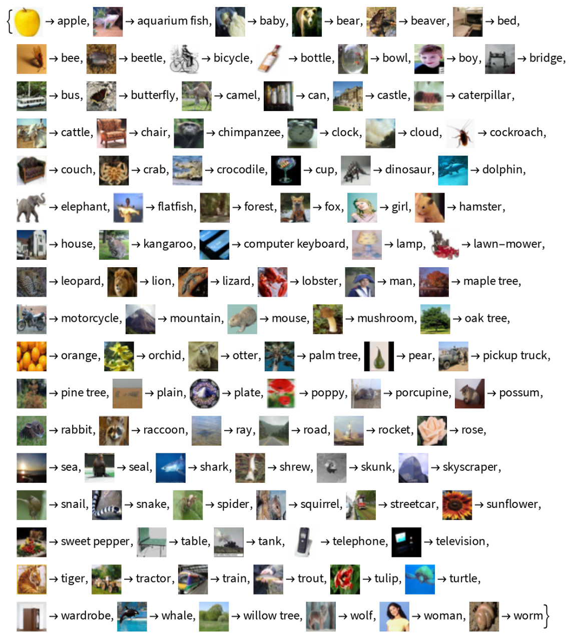
(a) MNIST.



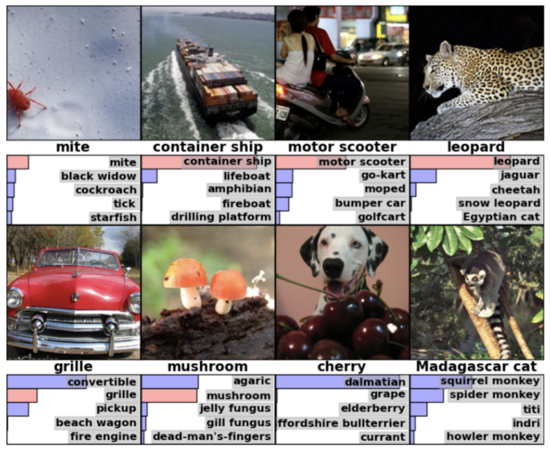
(b) CIFAR-10



(c) SVHN


→ apple, → aquarium fish, → baby, → bear, → beaver, → bed,
→ bee, → beetle, → bicycle, → bottle, → bowl, → boy, → bridge,
→ bus, → butterfly, → camel, → can, → castle, → caterpillar,
→ cattle, → chair, → chimpanzee, → clock, → cloud, → cockroach,
→ couch, → crab, → crocodile, → cup, → dinosaur, → dolphin,
→ elephant, → flatfish, → forest, → fox, → girl, → hamster,
→ house, → kangaroo, → computer keyboard, → lamp, → lawn−mower,
→ leopard, → lion, → lizard, → lobster, → man, → maple tree,
→ motorcycle, → mountain, → mouse, → mushroom, → oak tree,
→ orange, → orchid, → otter, → palm tree, → pear, → pickup truck,
→ pine tree, → plain, → plate, → poppy, → porcupine, → possum,
→ rabbit, → raccoon, → ray, → road, → rocket, → rose,
→ sea, → seal, → shark, → shrew, → skunk, → skyscraper,
→ snail, → snake, → spider, → squirrel, → streetcar, → sunflower,
→ sweet pepper, → table, → tank, → telephone, → television,
→ tiger, → tractor, → train, → trout, → tulip, → turtle,
→ wardrobe, → whale, → willow tree, → wolf, → woman, → worm

(d) CIFAR-100



(e) ImageNet.

Figure 2.8: Examples of Commonly Used Datasets.

MNIST [15] is a dataset for handwritten digits with 60,000 training images and 10,000 test images. There are ten classes with ten digits. The image is in greyscale (thus, one color channel) and the size of 28×28.

CIFAR-10 and CIFAR-100 [25] is a dataset of 3-channel image including 50,000 training images and 10,000 test images including 10-class and 100-class, respectively.

Each image is with size of $32 \times 32 \times 3$.

The Street View House Number (SVHN) [26] is a real-world color image dataset resized to a fixed resolution of 32×32 pixels. It contains 73,257 training images and 26,032 testing images.

ImageNet [21] is a large-scale dataset including 1000 categories and 1.2 million images in the training set, and 50,000 images, 50 per class in the testing set. Top-1 and Top-5 error rate or score is the metrics used to measure classification performance. The top-1 score is to check if the predicted top class is the same as the given label. The top-5 score is to check if one of the predicted top 5 classes is the same as the provided label.

Deep learning frameworks offer building blocks for designing, training, and validating deep neural networks through a high-level programming interface. Widely used deep learning frameworks such as PyTorch, TensorFlow, Caffe2, Cognitive toolkit, MXNet, and others rely on GPU-accelerated libraries such as cuDNN and NCCL to deliver high-performance multi-GPU accelerated training. Frameworks used in each work are described in each chapter.

Chapter 3

CGAP: CONTINUOUS GROWTH AND PRUNING FOR EFFICIENT DEEP LEARNING

3.1 Introduction

Deep Neural Networks have various applications including image classification [20], object detection [27], speech recognition [28] and natural language processing [29]. However, the accuracy of DNNs heavily relies on massive amounts of parameters and deep structures, making it hard to deploy DNNs on resource-limited embedded systems. When training or inferring the DNN models on hardware, the model must be stored in the external memory such as dynamic random-access memory (DRAM) and fetched multiple times. These operations are expensive in computation, memory access, and energy consumption. For example, Figure 3.1 shows the energy consumption of one inference pass in several modern DNN structures, simulated by the FPGA performance model [30] under the setting of 300 MHz operating frequency and 19.2 GB/s DRAM bandwidth. The input image size is 32×32. A typical DNN model is too large to fit in on-chip memory. For instance, VGG-19 [22] has 20.4M parameters. Running such a model requires frequent external memory access, exacerbating the power consumption of a typical embedded system.

Previous researches have designed customized hardware for DNN acceleration [31, 32]. Most of them are limited to relatively small neural networks, such as LeNet-5 [15]. For larger networks such as AlexNet [20] and VGG-16 [22], additional efforts are usually required to improve the hardware efficiency [33,34]. For example, [33] saves the energy through data gating and zero skipping. Some other works focus on data reuse

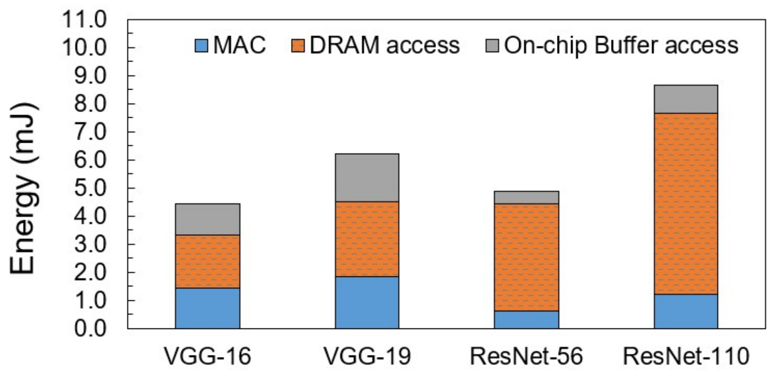


Figure 3.1: Energy Breakdown for Modern DNN Structures, Results from Simulation by the FPGA Performance Model. Due to the Redundancy in Parameters, Multiply-accumulator (MAC) and External Memory (DRAM) Access Dominate the Energy Consumption.

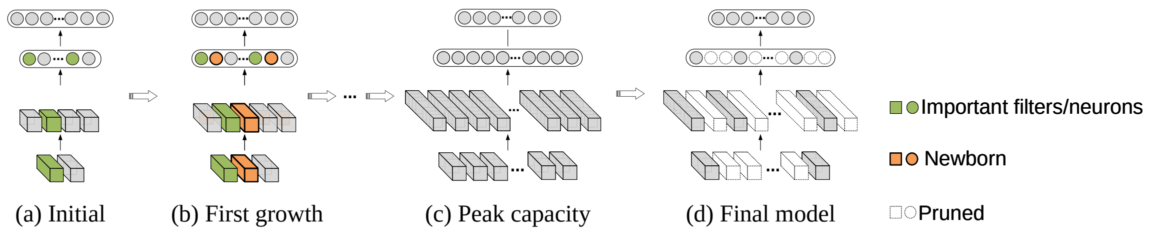


Figure 3.2: The Proposed CGaP Scheme. CGaP Starts the Training from a Seed Network Instead of an Over-parameterized One, Gradually Grows Important Learning Units During the Training and Reaches Peak Capacity at the End of Growth, Then Prunes Secondary Filters and Neurons to Generate an Inference Model with Structured Sparsity and Up-to-date Accuracy.

of convolutional layers and demonstrate the results on specific hardware [31, 35–37]. However, their improvements are limited on those networks where fully-connected layer is widely used, such as RNNs and LSTMs.

To support more general models, network pruning is a popular approach by removing secondary weights and neurons. Network pruning executes a three-step procedure, which 1) trains a pre-designed network from scratch, 2) removes less important connections or filters/neurons according to a saliency score (a metrics to measure the importance of weights and learning units) [38–42], or by adding a regularization term into the loss function [43, 44], and 3) fine-tunes to recover the accuracy.

However, the above pruning techniques suffer from two limitations: (1) training a large and fixed network from scratch could be sub-optimal as it introduces redundancy; (2) in the process of training, pruning only discards less important weights at the end of training but does not strengthen important weights and nodes. These limitations of network pruning confine the learning performance as well as the model pruning efficiency (i.e., how many parameters can be removed and how structured the sparsity is).

In contrast to the static DNN model, the biological nervous system exhibits active growth and pruning through the lifetime. [45–47] have observed that the rapid growth of neurons and synapses takes place in an infant's brain and is vital to the maturity of an adult's brain. In brains, some neurons and synapses are used more frequently and are consequently strengthened. Those neurons and synapses that are not used consistently are weakened and removed. The structural plasticity of brain is central to the study of developmental biology.

Inspired by this observation from biology, we propose a training scheme named Continuous Growth and Pruning (CGaP), which leverages structural plasticity to tackle the aforementioned limitations of pruning techniques. Instead of training an over-parameterized network from scratch, CGaP starts the training from a small network seed (Figure 3.2(a)), whose size is as low as 0.1%-3% of the full-size reference model. In each iteration of the growth, CGaP locally sorts neurons and filters (also known as output channels in some literature) according to our saliency score (Section 3.3.2). Based on the saliency score, important learning units are selected and the corresponding new units are added (see Figure 3.2(b)). The selection and addition of important units help reinforce the learning and increase model capacity. Then a filter-wise and neuron-wise pruning will be executed on the post-growth model (Figure 3.2(c)) based on pruning metrics. Finally, CGaP generates a significantly sparse and structured inference model (Figure 3.2(d)) with accuracy improved. In the generated inference model, large amounts of filters and neurons have been removed, achieving structured pruning. Compared to non-structured pruning [38], CGaP benefits hardware implementation as it reduces the computation volume and memory access without any additional hardware architecture change.

Algorithmic experiments and hardware simulations validate that CGaP significantly decreases the number of external and on-chip memory accesses, accelerating the inference by bypassing the removed filters and neurons. On the algorithm side, we demonstrate the performance in accuracy and model pruning on several networks and datasets. For instance, CGaP reduces 78.9% parameters of VGG-19 with +0.37% accuracy improvement on CIFAR-100 [25], 85.8% parameters with +0.23% accuracy improvement on SVHN [26]. For ResNet-110 [24], CGaP reduces 64.0% parameters with +0.09% accuracy improvement on CIFAR-10 [25]. These results exceed the state-of-the-art pruning methods [38–41, 48, 49]. Furthermore, we validate the efficiency of the inference model generated from CGaP using FPGA simulator [30]. For one inference pass of VGG-19 on CIFAR-100, previous non-structured pruning approach [38] requires energy consumption of 2.7×10^9 pJ in accessing DRAM and 5.6 ms inference latency, while CGaP requires only 2.2×10^9 pJ and 4.4 ms latency.

The contribution of this work is as follows:

- A brain-inspired training flow (CGaP) with a dynamic structure is proposed.

CGaP grows the network from a small seed and effectively reduces over-parameterization without sacrificing accuracy.

- The advantage of structured sparsity of the inference model generated from CGaP is validated using a high-level FPGA performance model including on-chip buffer access energy, external memory access energy and inference latency.
- The discussion and understanding of the reason that the growth improves the learning efficiency are provided.

The rest of the paper is organized as follows. Section 3.2 introduces the background of model pruning. Section 3.3 demonstrates the saliency score used to select the learning units. Section 3.4 describes the proposed Continuous Growth and Pruning scheme. Section 3.5 presents the experimental results from algorithmic simulations. Section 3.6 demonstrates the simulation results from FPGA performance modeling. Section 3.7 discusses the understanding of network plasticity as well as ablation study. Section 3.8 concludes this work and discusses the insight into future work.

3.2 Previous Work

There have been broad interests in reducing the redundancy of DNNs in order to deploy them on a resource-limited hardware platform. The structural surgery is a widely used approach and can be categorized into destructive direction and constructive direction. We will discuss these two directions, as well as orthogonal approaches to our methods in this section.

3.2.1 Destructive Methods

The destructive methods zero out specific connections or remove filters or neurons in convolutional or fully-connected layers, generating a sparse model. Weight magnitude pruning [38] pruned weights by setting the selected weights to zeros. The selection is based on L1-norm, i.e., the absolute value of the weight. Weight magnitude pruning generates a sparse weight matrix, but not in a structured way. In this case, specific hardware design [50] is needed to take advantage of the optimized inference model, otherwise the non-structured sparsity does not benefit hardware acceleration due to the overhead in model management. The kernel-wise pruning [39] pruned kernels layer by layer based on the saliency metrics of each filter and achieved structured sparsity in the inference model. Compared to [39], CGaP prunes filters, leading to a more structured inference model. Besides the saliency-based pruning, the penalty-based approach has been explored by [44, 51] and structured sparsity was achieved. Our method is different from all the above pruning schemes from two perspectives: (1) We start training from a small seed other than an over-parameterized network; (2) Besides removing secondary filters/neurons, we also reinforce important ones to further improve learning accuracy and model compactness.

3.2.2 Constructive Methods

The constructive approaches include techniques that add new connections or filters to enlarge the model capacity. [52, 53] increased network size by adding random neurons with fresh initialization (i.e., weights are randomly initialized, without pre-trained information). They evaluated their approach on basic XOR problems. Different from their approach, CGaP selectively adds neurons and filters that are initialized with the information learned from the previous training. Meanwhile, CGaP is validated on modern DNNs and datasets under more realistic scenarios. [54] grew the smallest Neural Tree Networks (NTN) to minimize the number of classification errors on Boolean function learning tasks, and used pruning to enhance the generalization of NTN. [55] improved the accuracy of radial basis function (RBF) networks on function approximation tasks by adding and removing hidden neurons. To enhance the accuracy of spike-based classifiers, [56] progressively added dendrites to the network, and then optimized the topology of the dendritic tree. Different from them, CGaP aims at improving the efficiency of the inference model of modern Deep Neural Networks on image classification tasks. [57] constructed the DNN by activating connections and choosing a set of convolutional filters among a bunch of randomly generated filters according to their influence on the training performance. However, this approach highly depends on trial and error to find the optimal set of filters that could reduce loss the most. This approach is sensitive to power and timing budgets, limiting its extension on large datasets. Unlike their work, CGaP directly grows the network from a seed, minimizing the effort on trail and error.

3.2.3 Orthogonal Methods

The orthogonal methods, such as low-precision quantization and low-rank decomposition, compress the DNN models by quantizing the parameters to fewer bits [58,59], or by finding a low-rank approximation [60, 61]. Note that our CGaP approach can be combined with these orthogonal methods to further improve inference efficiency.

3.3 Saliency Score

In this section, we describe the detailed methodology of CGaP, starting from the saliency score, which is used to sample the importance of a learning unit. Section 3.3.1 defines the terminology we use in this paper. Section 3.3.2 provides the mathematical proof of the saliency score we adopt.

3.3.1 Terminology

A DNN can be treated as a feedforward multi-layer architecture that maps the input images to certain output vectors. Each layer is a certain function, such as convolution, ReLU, pooling and inner product, whose input is $\mathcal{X}$, output is $\mathcal{Y}$ and parameter is $\mathcal{W}$ in case of convolutional and fully-connected layers. Hereby the convolutional layer (conv-layer) is formulated as: $\mathcal{Y}_l = \mathcal{X}_l * \mathcal{W}_l$, wherein $\mathcal{X}_l \in \mathbb{R}^{I_l \times Wi_l \times Hi_l}$, $\mathcal{Y}_l \in \mathbb{R}^{O_l \times Wo_l \times Ho_l} \Leftrightarrow \mathcal{X}_{l+1} \in \mathbb{R}^{I_{l+1} \times Wi_{l+1} \times Hi_{l+1}}$, $\mathcal{W}_l \in \mathbb{R}^{O_l \times I_l \times K \times K}$, where subscript $_l$ denotes the index of the layer. And the fully-connected layer is represented by: $\mathcal{Y}_l = \mathcal{X}_l \cdot \mathcal{W}_l$, where the input $\mathcal{X}_l \in \mathbb{R}^{I_l}$, the output $\mathcal{Y}_l \in \mathbb{R}^{O_l} \Leftrightarrow \mathcal{X}_{l+1} \in \mathbb{R}^{I_{l+1}}$, and the parameter matrix is $\mathcal{W}_l \in \mathbb{R}^{O_l \times I_l}$.

Convolutional layer (conv-layer) l the 4 dimensions of its weight matrix are: the number of output channels O_l, the number of input channels I_l, and the kernel width and height K, respectively. We denote the o-th **3D filter**, which generates the o-th output channel in the feature map, as $W_l^o \in \mathbb{R}^{I_l \times K \times K}$. The i-th **2D kernel** in the o-th filter is denoted as $W_l^{o,i} \in \mathbb{R}^{K \times K}$. On the other hand, a **4D weight tensor** $\mathbf{W}_l^i \in \mathbb{R}^{O_l \times 1 \times K \times K}$, which operates on the i-th input feature map, is a package of O_l kernels across all output channels. For example, in Figure 3.3, $W_{l,picked}^j$ is a 3D filter consisting of I_l kernels, and $\mathbf{W}_{l+1,projected}^j$ as well as $\mathbf{W}_{l+1,mapped}^j$ are both 4D tensors with dimension of $O_l \times 1 \times K \times K$, which include all the output channels but have only one input channel located at j. The $W_l^{o,i,m,n} \in \mathbb{R}^{1 \times 1}$ refers to one weight at the m-th row and the n-th column in the o-th filter of the i-th input channel.

Fully-connected layer (fc-layer) l input $\mathcal{X}_l$ propagate from one hidden activation i to the next layer. We refer the whole set of $W^i_{l,fan-out}$ as a neuron N^i_l. This neuron receives information from previous layer $l-1$ through its **fan-in** weights $W^i_{l,fan-in} \in \mathbb{R}^{1\times I_{l-1}}$ (as shown in Figure 3.4) and propagates to the next layer through **fan-out** weights $W^i_{l,fan-out} \in \mathbb{R}^{O_l\times 1}$. Also note that the output dimension of layer $l-1$ equals to the input dimension of layer l, i.e., $O_{l-1} = I_l$. The weight pixel in layer l at the cross-point of row o and column i is denoted as $W^{o,i}_{l,fan-out} \in \mathbb{R}^{1\times 1}$. Moreover, the 'depth' of a DNN model indicates the number of layers, and the 'width' of a DNN model refers to the number of filters or neurons of each layer.

Learning units Growing or pruning a **filter** W^o_l indicates adding or removing $W^o_l \in \mathbb{R}^{I_l\times K\times K}$ and its corresponding output feature map. Growing or pruning a **neuron** N^i_l means adding or removing both $W^i_{l,fan-out} \in \mathbb{R}^{O_l\times 1}$ and $W^i_{l,fan-in} \in \mathbb{R}^{1\times I_{l-1}}$.

3.3.2 Saliency Score

We adopt a saliency score to measure the effect of a single filter/neuron on the loss function, i.e., the importance of each learning unit. The saliency score is developed from Taylor Expansion of the loss function. Previously, [62] applied it on pruning. In this paper, we adopt this saliency score and apply it on the growth and pruning scheme. In this section, we provide a mathematical formulation of the saliency score.

The saliency score represents the difference between the loss with and without each unit. In other words, if the removal of a filter/neuron leads to relatively small accuracy degradation, this unit is recognized as an unimportant unit, and vice versa. Thus, the objective function to get the filter with the highest saliency score is formulated as:

$$\underset{W_l^o}{argmin}|\Delta\mathcal{L}(W_l^o)| \Leftrightarrow \underset{W_l^o}{argmin}|\mathcal{L}(\mathcal{Y};\mathcal{X},\mathcal{W}) - \mathcal{L}(\mathcal{Y};\mathcal{X},W_l^o = \mathbf{0})|. \tag{3.1}$$

Using the first-order of the Taylor Expansion:

$$|\mathcal{L}(\mathcal{Y};\mathcal{X},\mathcal{W}) - \mathcal{L}(\mathcal{Y};\mathcal{X},W_l^o = \mathbf{0})| \ at \ W_l^o = \mathbf{0}. \tag{3.2}$$

we get:

$$\begin{aligned} |\Delta\mathcal{L}(W_l^o)| &\simeq |\frac{\partial\mathcal{L}(\mathcal{Y};\mathcal{X},\mathcal{W})}{\partial W_l^o} W_l^o| \\ &= \sum_{i=0}^{I_l}\sum_{m=0}^{K}\sum_{n=0}^{K} |\frac{\partial\mathcal{L}(\mathcal{Y};\mathcal{X},\mathcal{W})}{\partial W_l^{o,i,m,n}} W_l^{o,i,m,n}|. \end{aligned} \tag{3.3}$$

Similarly, the saliency score of a neuron is derived as:

$$|\Delta\mathcal{L}(N_l^i)| \simeq |\frac{\partial\mathcal{L}(\mathcal{Y};\mathcal{X},\mathcal{W})}{\partial W_{l,fan-out}^i} W_{l,fan-out}^i| = \sum_{o=0}^{O_l} |\frac{\partial\mathcal{L}(\mathcal{Y};\mathcal{X},\mathcal{W})}{\partial W_{l,fan-out}{}^{o,i}} W_{l,fan-out}^{o,i}|. \tag{3.4}$$

3.4 CGaP Methodology

With the saliency score as the foundation, we develop the entire CGaP flow atop. This section explains the overall flow and the detailed implementation of each step in CGaP.

The CGaP scheme is described in Algorithm 1. Starting from a small network seed, the growth takes place periodically at a frequency of f_{growth} (see Algorithm 1 line 4, where '%' denotes the operation to obtain the remainder of division). During each growth, important learning units are chosen and grown at growth ratio β layer by layer from the bottom (input) to top (output), based on the local ranking of the saliency score. The growth phase stops when reaching a capacity threshold $\tau_{capa.}$, followed by several epochs of training on the peak model M_{peak}. When the training accuracy reaches a threshold $\tau_{accu.}$, the pruning phase starts. Pruning is performed layer by layer, from the bottom layer to the top layer, at the frequency of $f_{pruning}$. The details in the growth phase and the pruning phase is demonstrated as follows.

Algorithm 1 Entire flow

Input: Model seed $M_{initial}$

```
1: Initialize a small network model $M_{current} \leftarrow M_{initial}$.
2: for epoch = 1 to E do
3:     Train current model $M_{current}$ and fetch $Accuracy$.
4:     if $epoch\%\frac{1}{f_{growth}} = 0$ and $M_{current} < \tau_{capa.}$ then
5:         Grow the network according to Algorithm 2
6:         $M_{current} \leftarrow M_{grown}$.
7:     end if
8:     $M_{peak} \leftarrow M_{current}$.
9:     if epoch$\%\frac{1}{f_{pruning}} = 0$ and $Accuracy > \tau_{accu.}$ then
10:        Prune the network following Algorithm 3
11:        $M_{current} \leftarrow M_{pruned}$.
12:    end if
13: end for
14: $M_{final} \leftarrow M_{current}$ and test $M_{final}$.
```

Output: Final compact model M_{final}

3.4.1 Growth Phase

Algorithm 2 presents the methodology in the growth phase. Each iteration of growth in a layer consists of two steps: growth in layer l and mapping in the adjacent layer. There are two conditions need to be discussed separately: convolutional layers (Figure 3.3) and fully-connected layers (Figure 3.4). Due to the difference between these two kinds of operation as discussed previously, after the growth of layer l, the mapping in conv-layer takes place at the adjacent layer $l+1$. In fc-layers, the mapping is in layer $l-1$.

Algorithm 2 Growth phase

Input: Current network $M_{current}$

```
1: for each layer $l = 1$ to L do
2:     for each filter $W_l^o$ in conv-layer $l$, or each neuron $N_l^i$ in fc-layer $l$ do
3:         Calculate growth score $GS_{W_l^o}$ according to Eq. 3.3 and $GS_{N_l^i}$ according to Eq. 3.4.
4:     end for
5:     Sort all units and select $\beta O_l$ filters or $\beta I_l$ neurons with the highest $GS_{W_l^o}$ or $GS_{N_l^i}$.
6:     for each filter j = 1 to $\beta O_l$ (for fc-layer, $\beta I_l$) do
7:         Add one filter/neuron on the side of the each picked filter/neuron in layer $l$.
8:         Initialize picked and new-born filters (neurons) according to Eq. 3.5 and Eq. 3.6.
9:         Map corresponding input-wise weight in layer $l + 1$ (fan-in weights in layer $l - 1$).
10:        Initialize projected and mapped filters according to Eq. 3.7 and Eq. 3.8 (neurons according to Eq. 3.9 and Eq. 3.10).
11:    end for
12: end for
```

Output: M_{grown}

Growth in conv-layer l According to the local ranking of the saliency score (Eq. 3.3), we sort all the 3D filters in this layer. With a growth ratio β, $\beta O_{l,t}$ filters are selected in the l-th layer at the t-th growth. On the side of each selected filter $\mathbf{W}^j_{l,picked} \in \mathbb{R}^{I_l \times K \times K}$, as shown in Figure 3.3, we create a new filter that has the same

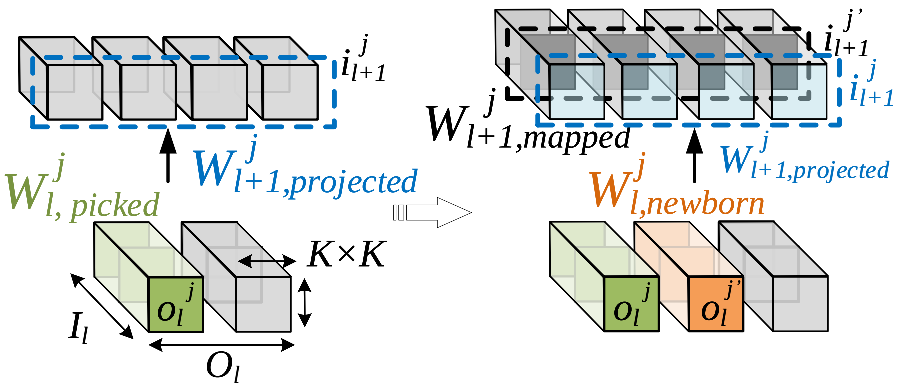


Figure 3.3: Illustration of Two-step Growth in Conv-layers. The Growth Phase Follows a Two-step (Growing and Mapping) Procedure. After the Filter $W^j_{l,picked}$ (Green) Is Picked and Split Aside, Giving Birth To $W^j_{l,newborn}$ (Orange), the Projected Input-wise Filter, $W^j_{l+1,projected}$ (Blue) in Layer $l+1$, Is as Well Split aside, Generating $W^j_{l+1,mapped}$ (Black).

size, named $\mathbf{W}^j_{l,newborn} \in \mathbb{R}^{I_l\times K\times K}$.

In the ideal case, the new filter $\mathbf{W}^j_{l,newborn}$ and existing filter $\mathbf{W}^j_{l,picked}$ are expected to collaborate with each other and optimize the learning. The existing filter $\mathbf{W}^j_{l,picked}$ has already learned on the current task. To keep the same learning pace between the existing filter and the new filter, we initialize $\mathbf{W}^j_{l,newborn}$ as follows:

$$\mathbf{W}^j_{l,newborn} = \sigma\mathbf{W}^j_{l,picked} + X \sim U([-\mu,\mu]), \tag{3.5}$$

$$\mathbf{W}^j_{l,picked} = \sigma\mathbf{W}^j_{l,picked} + X \sim U([-\mu,\mu]), \tag{3.6}$$

where $\sigma \in (0,1]$ is a scaling factor and X is a constant following uniform distribution in $[-\mu,\mu]$, where $\mu \in (0,1]$. Instead of random initialization, the above initialization helps reconcile the learning status of the newborn filters with the old filters. Meanwhile, the scaling factor prevents output from an exponential explosion caused by the feedforward propagation $\mathcal{Y}_l = \mathcal{X}_l * \mathcal{W}_l$. The noise X prevents the learning from sticking at a local minimum that leads to sub-optimal solutions. No matter which distribution the noise X follows, X in a reasonable range is able to provide similar performance. However, other distributions usually introduce more hyper-parameters and thus, require more efforts in parameter tuning. For example, Gaussian noise introduces more hyper-parameter, e.g., the standard deviation, than uniform noise. For simplicity, we use uniformly distributed noise.

Mapping in conv-layer $l+1$ After the number of filters in layer l grows from $O_{l,t}$ to $(1+\beta)O_{l,t}$, the number of output feature maps also increases from $O_{l,t}$ to $(1+\beta)O_{l,t}$. Therefore, the input-wise dimension of layer $l+1$ should increase correspondingly in order to be consistent in data propagation. To match the dimension, we first locate the 4D tensor $\mathbf{W}^j_{l+1,projected}$ in layer $l+1$, which processes the feature maps generated by $W^j_{l,picked}$. Then we add a new 4D tensor $\mathbf{W}^j_{l+1,mapped}$ adjacent to $\mathbf{W}^j_{l+1,projected}$. The $\mathbf{W}^j_{l+1,mapped}$ and $\mathbf{W}^j_{l+1,projected}$ are initialized as follows:

$$\mathbf{W}^j_{l+1,mapped} = \sigma \mathbf{W}^j_{l+1,projected} + X \sim U([-\mu, \mu]), \tag{3.7}$$

$$\mathbf{W}^j_{l+1,projected} = \sigma \mathbf{W}^j_{l+1,projected} + X \sim U([-\mu, \mu]). \tag{3.8}$$

To summarize, as illustrated in Figure 3.3, the filter $W^j_{l,picked}$ (green) is selected according to the saliency score and a new tensor $W^j_{l,newborn}$ (orange) is added. Then the input-wise tensor $\mathbf{W}^j_{l+1,projected}$ (in blue dashed rectangular) in layer $l+1$ is projected, and $\mathbf{W}^j_{l+1,mapped}$ (in black dashed rectangular) is generated.

After layer l grows and layer $l+1$ is mapped, layer $l+1$ grows and layer $l+2$ is mapped, so on and so forth till the last convolutional layer. It is worth mentioning that for the 'projection shortcuts' [24] with 1×1 convolutions in ResNet [24], the dimension mapping is between the two layers that the shortcut connects, not necessarily to be the adjacent layers.

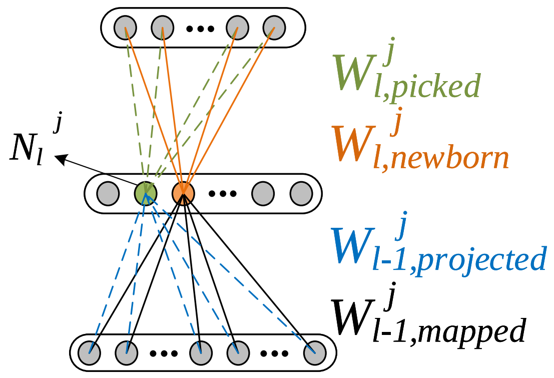


Figure 3.4: Illustration of Two-step Growth in FC-layers. First, Fan-out Weights $w^j_{L,Newborn}$ (Orange) Is Added, Then Fan-in Weights $\mathbf{W}^j_{l-1,mapped}$ (Black) Form the Connections from the Newborn Neuron to All Neurons in Layer $l-1$.

Growth and mapping in fc-layers As illustrated in Figure 3.4, the neuron growth in fc-layers l occurs at fan-out weights, and its initialization follows Eq. 3.5 and 3.6.

The mapping in fc-layers take place in the fan-in weights as follows:

$$\mathbf{W}^j_{l-1,mapped} = \sigma \mathbf{W}^j_{l-1,projected} + X \sim U([-\mu,\mu]), \tag{3.9}$$

$$\mathbf{W}^j_{l-1,projected} = \sigma \mathbf{W}^j_{l-1,projected} + X \sim U([-\mu,\mu]). \tag{3.10}$$

After growing the last conv-layer, We flatten the output feature map of this conv-layer, treat it as the input from layer $l-1$ and map in the same manner.

3.4.2 Pruning Phase

Pruning in each layer consists of two steps: weight pruning and unit pruning. First, we sort weight pixels locally in each conv-layer according to Eq.3.11:

$$PS_{W_l^{o,i,m,n}} = \left|\frac{\partial \mathcal{L}(\mathcal{Y};\mathcal{X},\mathcal{W})}{\partial W_l^{o,i,m,n}} W_l^{o,i,m,n}\right| \tag{3.11}$$

```
Algorithm 3 Pruning phase
Input: Current network $M_{current}$
1: for each weight $W_l^{o,i,m,n} \in \mathbb{R}^{1\times1}$ in conv-layer $l$ or each $W_l^{o,i} \in \mathbb{R}^{1\times1}$ in fc-layer $l$ do
2:     Calculate weight pruning score $PS_W$ according to Eq. 3.11 for conv-layers and Eq. 3.12 for fc-layers.
3: end for
4: Sort weights by $PS_W$.
5: Zero-out the lowest $\gamma_W \prod(O_l, I_l, K, K)$ weights in conv-layer and $\gamma_W \prod(I_l, O_l)$ weights in fc-layer.
6: for each filter $W_l^o$ (neuron $N_l^i$) in all layers do
7:     Zero-out entire filter $W_l^o$ (neuron $N_l^i$) if the weight sparsity is larger than pruning rate $\gamma_F$ ($\gamma_N$).
8: end for
Output: $M_{pruned}$
```

and in each fc-layer according to Eq.3.12:

$$PS_{W_l^{o,i}} = \left|\frac{\partial\mathcal{L}(\mathcal{Y};\mathcal{X},\mathcal{W})}{\partial W_{l,fan-out}{}^{o,i}} W_{l,fan-out}^{o,i}\right| \tag{3.12}$$

In each layer, $100\gamma_W\%$ weight pixels with the lowest PS_W are set as zero, where $\gamma_W \in (0,1)$ is the weight pruning rate. Then the entire filter/neuron whose sparsity is larger than the filter/neuron pruning rate γ_F or $\gamma_N \in (0,1)$ is set to zero. In this way, a large amount of entire filters/neurons are pruned, leading to a compact inference model.

3.5 Algorithmic Experiments

To evaluate the proposed approach, we present experimental results in this section. We perform experiments on several modern DNN structures (LeNet [15], VGG-Net [22], ResNet [24]) and representative datasets (MNIST [15], CIFAR-10, CIFAR-100 [25], SVHN [26]).

3.5.1 Training Setup

Network structures The LeNet-5 architecture consists of two sets of convolutional, ReLU [63] and max pooling layers, followed by two fully-connected layers and finally a softmax classifier. The VGG-16 and VGG-19 structures we use have the same convolutional structure as [22] but are redesigned with only two fully-connected to be fairly compared with the pruning-only method [39]. Therefore, the VGG-16 (VGG-19) has 13 (16) convolutional layers, each is followed by a batch normalization layer [64] and a ReLU activation. The structures of ResNet-56 and ResNet-110 follow [39]. Each convolutional layer is followed by a batch normalization layer and ReLU activation. During the training, the depth of the networks remains constant since CGaP does not touch the depth of the network, but the width of each layer changes.

Note that in the following text, we denote the full-size models trained from scratch without sparsity regularization as 'baseline' models. The three-step pruning schemes that remove weights or filters but do not execute network growth are denoted as 'pruning-only' models.

Datasets MNIST is a handwritten digit dataset in grey-scale (i.e., one color channel) with 10 classes from digit 0 to digit 9. It consists of 60,000 training images and 10,000 testing images. The CIFAR-10 dataset consists of 60,000 32×32 color images in 10 classes, with 5000 training images and 1000 testing images per class. The CIFAR-100 dataset has 100 classes, including 500 training images and 100 testing images per class. The Street View House Number (SVHN) is a real-world color image dataset that is resized to a fixed resolution of 32×32 pixels. It contains 73,257 training images and 26,032 testing images.

Hyper-parameters We set the learning rate to be 0.1 and divide by 10 for every 30% of the training epochs. We train our model using Stochastic Gradient Descent (SGD) with a batch size of 128 examples, a momentum of 0.9, and a weight decay of 0.0005. The loss function is the cross-entropy loss with softmax function. We train 60, 200, 220 and 100 epochs on MNIST, CIFAR-10, CIFAR-100 and SVHN datasets, respectively. In the growth phase, we have hyper-parameters set as follows: the growth stopping condition $\tau_{capa.} = O_{1,baseline}$, i.e., the growth stops at the t-th growth if the number of filters in the $(t+1)$-th growth is larger than the baseline model. The growth ratio β is set as 0.6. The growth frequency f_{growth} is set as 1/3. The scaling factor σ in Eq. 3.5 to Eq. 3.10 is set to 0.5 and μ is 0.1. The pruning frequency $f_{pruning}$ is set to be 1. The setting of the weight pruning rate γ_W follows [38], [39] and [41] for LeNet-5, VGG-Net and ResNet, respectively. γ_F and γ_N is set to be same as γ_W.

Framework and platform The experiments are performed with PyTorch [65] framework on one NVIDIA GeForce GTX 1080 Ti platform. It is worth mentioning that experiments performed with different frameworks may have variation in accuracy and performance. Thus, to have a fair comparison among CGaP, baseline and pruning-only methods, all the results in Table 3.1, 3.2, 3.3 and 3.4 are obtained from experiments with PyTorch framework.

3.5.2 Performance Evaluation

With training setup as aforementioned, we perform experiments on several datasets with modern DNN architectures. In Table 3.1, Table 3.2, Table 3.3 and Table 3.4, we summarize the performance attained by CGaP on MNIST, CIFAR-100, SVHN, and CIFAR-10 datasets, respectively. To be specific, the second column 'Accuracy' denotes the inference accuracy in percentage achieved by the baseline model, the up-to-date pruning-only approaches and CGaP approach, respectively.

The column 'FLOPs' represent the calculated number of FLOPs of a single inference pass. The calculation of FLOPs follows the method described in [62]. Fewer FLOPs means lower computation cost in one inference pass. The neighboring column, 'Pruned', represents the reduction of FLOPs in the compressed model as compared to the baseline model. The column 'Param.' stands for the number of parameters of the inference model. Fewer parameters promise a smaller model size. The last column, 'Pruned', denotes the percentage pruned in parameters compared to the baseline. Larger pruned percentage implies fewer computation operations and more compact model. The best result of each column is highlighted in bold.

The results shown in Table 3.1 to 3.4 prove that CGaP outperforms the previous pruning-only approaches in accuracy and model size. For instance, as displayed in Table 3.4, on ResNet-56, our CGaP approach achieves 93.20% accuracy with 32.5% reduction in FLOPs and 37.6% reduction in parameters, while the up-to-date pruning-only method [48] that deals with static structure only reaches 92.56% accuracy with 32.1% reduction in FLOPs and 14.1% reduction in parameters. On ResNet-110, though [49] achieves 0.09% higher accuracy than CGaP, CGaP overwhelms it by trimming 22.5% more FLOPs.

Table 3.1: Evaluation of the Performance on MNIST.

Method	Accuracy	FLOPs	Pruned	Param.	Pruned
LeNet5-Baseline	99.29	4.59M	–	431K	–
Pruning [40]	99.26	0.85M	81.5%	112K	74.0%
Pruning [38]	99.23	0.73M	84.0%	36K	92.0%
CGaP	**99.36**	**0.44M**	**90.4%**	**8K**	**98.1%**

Table 3.2: Evaluation of the Performance on CIFAR-100.

Method	Accuracy	FLOPs	Pruned	Param.	Pruned
VGG19 - Baseline	72.63	797M	–	20.4M	–
Pruning [48]	71.85	NA	–	10.1M	50.5%
Pruning [41]	72.85	501M	37.1%	5.0M	75.5%
CGaP	**73.00**	**373M**	**53.2%**	**4.3M**	**78.9%**

Table 3.3: Evaluation of the Performance on SVHN.

Method	Accuracy	FLOPs	Pruned	Param.	Pruned
VGG19 - Baseline	96.02	797M	–	20.4M	–
Pruning [41]	96.13	398M	50.1%	3.1M	84.8%
CGaP	**96.25**	**206M**	**74.2%**	**2.9M**	**85.8%**

3.5.3 Visualization of the Dynamic Structures

Figure 3.5 presents the dynamic model size during CGaP training. During the growth phase, the model size continuously increases and reaches a peak capacity. When the pruning phase starts, the model size drops.

Furthermore, the sparsity achieved by CGaP is structured. In other words, large amounts of filters and neurons are entirely pruned. For instance, the baseline LeNet-

Table 3.4: Evaluation of the Performance on CIFAR-10.

Method	Accuracy	FLOPs	Pruned	Param.	Pruned
VGG16 - Baseline	93.25	630M	–	15.3M	–
Pruning [39]	93.40	410M	34.9%	5.4M	64.7%
CGaP	**93.59**	**280M**	**56.2%**	**4.5M**	**70.6%**
ResNet-56 - Baseline	93.03	268M	–	0.85M	–
Pruning [48]	92.56	182M	32.1%	0.73M	14.1%
Pruning [66]	90.20	**134M**	**50.0%**	NA	-
CGaP	**93.20**	181M	32.5%	**0.53M**	**37.6%**
ResNet-110 - Baseline	93.34	523M	–	1.72M	–
Pruning [39]	93.11	310M	40.7%	1.16M	32.6%
Pruning [49]	**93.52**	300M	40.8%	NA	–
CGaP	93.43	**192M**	**63.3%**	**0.62M**	**64.0%**

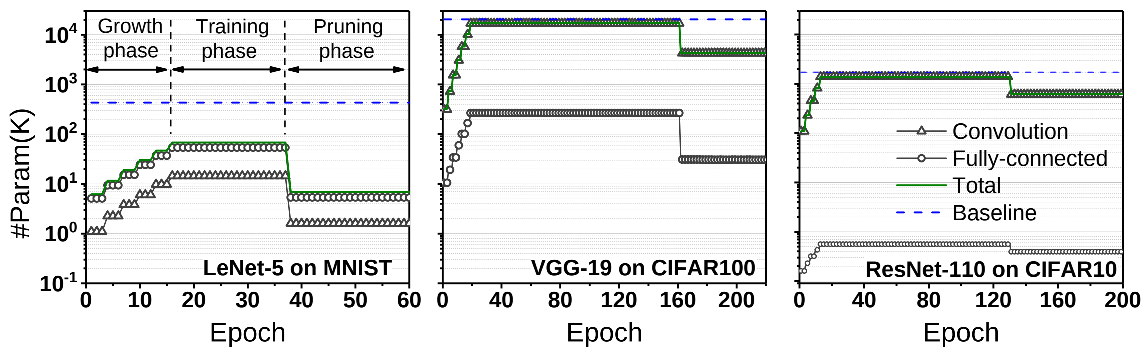


Figure 3.5: Number of Parameters During Training, Plotted at the End of Each Epoch. In the Beginning, the Model Size Increases Gradually Due to the Growth. After the Growth Ends and Several Epochs of Training on the Peak Model, One Drop Can Be Observed after the First Pruning. There Are Several Iterations of Pruning at a Frequency of 1.

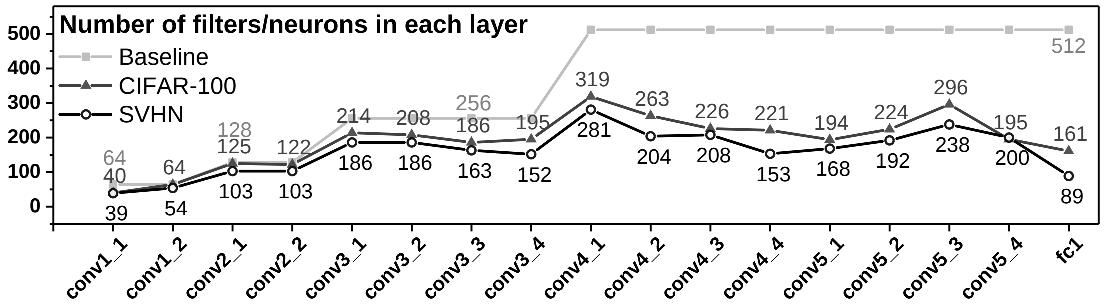


Figure 3.6: The VGG-19 Structures Learned by CGaP on CIFAR-100 and SVHN Datasets. The Shared Y-axis for Three Sub-figures Is the Number of Parameters of the Model.

5 without sparsity regularization has 20, 50 filters in conv-layer 1 and conv-layer 2, 500 and 10 neurons in fc-layer 1 and fc-layer 2, denoted as [20-50-500-10] (number of filters/neurons in [conv1-conv2-fc1-fc2]). The model achieved by CGaP contains only 8, 17 filters and 23, 10 neurons, denoted as [8-17-23-10]. Compared to baseline results, CGaP significantly decreases 60%, 66%, 95.4% units for each layer (the output layer should remain the same as the number of classes all the time). In this case, the pruned filters and neurons are skipped in the inference pass and thus accelerating the computation pipeline on hardware.

Another example is provided in Figure 3.6, which visualizes the VGG-19 structures from CGaP as well as the baseline structure on two different tasks. In the baseline model, the width (number of filters/neurons) of each layer is abundant, from 64 filters (the bottom conv-layers) to 512 filters (the top conv-layers). The baseline VGG-19 structure is designed to have a large enough size in order to guarantee the learning capacity. However, it turns out to be redundant, as proved by the structure that CGaP generated: 37.7% to 82.6% filters are pruned out in each layer. Meanwhile, in the baseline model, the top conv-layers are designed to have more filters than the

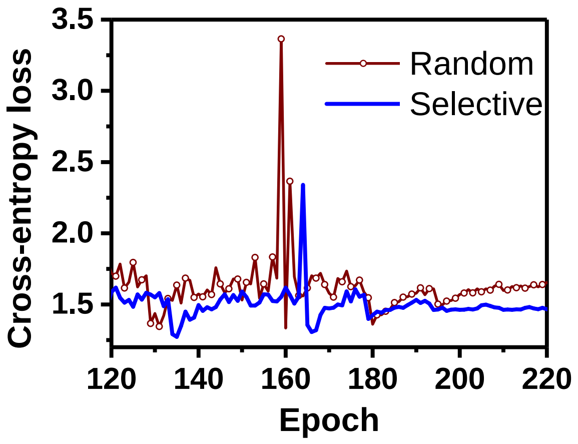


Figure 3.7: Saliency-based Growth Outperforms Random Growth. The Loss Is Monotonically Decreasing from Epoch 0 to 220 with Small Glitches. Here We Zoomed in from Epoch 120 to 220 to Show the Loss at the End of the Training.

bottom layers, but CGaP shows that it is not always necessary for top layers to have a relatively large size.

3.5.4 Validating the Saliency-based Growth

Figure 3.7 validates the efficacy of our saliency-based growth policy. Selective growth, which emphasizes the important units according to the saliency score, has lower cross-entropy loss than randomly growing some units. The spiking in Figure 3.7 is caused by the first iteration of pruning and this loss is recovered by the following iterative fine-tuning. In selective growth, this loss is 1.4× lower than that in random growth. This phenomenon supports our argument that selective growth assists the pruning phase. The detailed understanding of growth will be further discussed in Section 3.7.

To summarize the results from the algorithm simulations, the proposed CGaP approach:

- Largely compresses the model size by 37.6% (ResNet-56) to 98.1% (LeNet-5) for representative DNN structures.
- Decreases the inference cost, to be specific, number of FLOPs, by 32.5% (ResNet-56) to 90.4% (LeNet-5) on various datasets.
- Does not sacrifice accuracy and even improves accuracy.
- Outperforms the state-of-the-art pruning-only methods that deal with fixed structures.

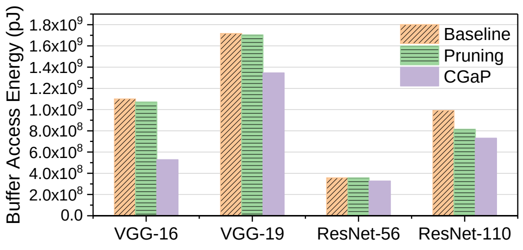
3.6 Experiments on FPGA Simulator

The results above demonstrate that CGaP generates an accurate and small inference model. In this section, we further evaluate the on-chip inference cost of the generated models and compare CGaP with previous non-structured pruning [38]. As CGaP achieves structured sparsity, CGaP outperforms the previous work on non-structured pruning in hardware acceleration and power efficiency. We validate this by performing the estimation of buffer access energy, DRAM access energy and latency using the performance model for FPGA [30].

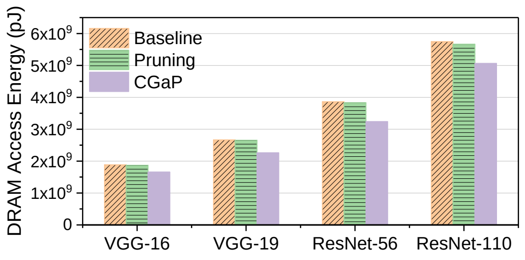
3.6.1 Overview of the FPGA Simulator

[30] is a high-level performance model designed to estimate the number of external and on-chip memory access, as well as the latency. The resource costs are formulated by the acceleration strategy as well as the design variables that control the loop tiling and unrolling. The performance model has been validated across several modern DNN algorithms in comparison to on-board testings on two FPGAs, with the differences within 3%, [30].

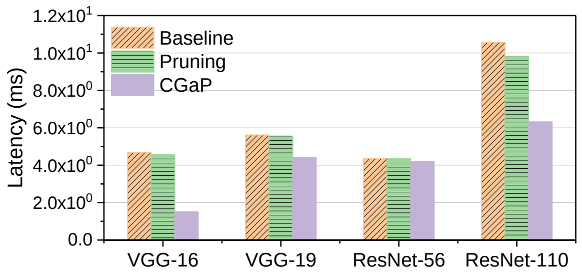
In the following experiments, the setup follows: the pixels and weights are both



(a) Comparison of Three Schemes in Buffer Access Energy (pJ) for VGG-16 on CIFAR-10, VGG-19 on CIFAR-100, ResNet-56 and ResNet-110 on CIFAR-10.



(b) Comparison of Three Schemes in DRAM Access Energy (pJ) for VGG-16 on CIFAR-10, VGG-19 on CIFAR-100, ResNet-56 and ResNet-110 on CIFAR-10.



(c) Comparison of Three Schemes in On-chip Inference Latency (ms) for VGG-16 on CIFAR-10, VGG-19 on CIFAR-100, ResNet-56 and ResNet-110 on CIFAR-10.

Figure 3.8: Estimation on FPGA Performance Model.

16-bit fixed point, the data width of DRAM controller is 512 bits, the accelerator operating frequency is 300 MHz, and the DRAM bandwidth is 19.2 GB/second. The parameters related to loop tiling and unrolling follow the setting in [30].

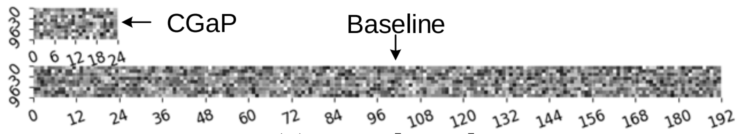
3.6.2 Results from FPGA Performance Model

The on-chip and external memory access energy across VGG-16, VGG-19, ResNet-56 and ResNet-110 is displayed in Figure 3.8(a) and Figure 3.8(b), respectively. The inference latency is shown in Figure 3.8(c). Though the models generated from weight magnitude pruning and CGaP have the same sparsity, CGaP outperforms non-structured magnitude weight pruning in hardware efficiency and acceleration. For example, with the same setup of the pruning ratio during training, magnitude weight pruning decreases 1.0% on-chip access energy, 1.0% DRAM access energy and 0.8% latency for VGG-19 on CIFAR-100, while the CGaP achieves 21.6%, 15%, and 21.1% reduction. The non-structured weight pruning [38] is able to improve the power and latency efficiency in comparison to baseline. However, the improvement is limited. In contrast, CGaP achieves significant acceleration and energy reduction. The reason is that the non-structured sparsity, i.e., scattered weight distribution, leads to irregular memory access that weakens the acceleration on hardware in a real scenario.

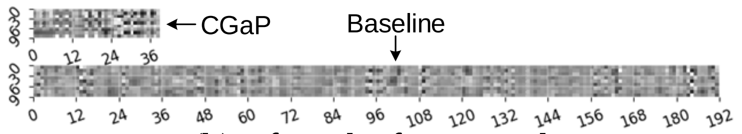
3.7 Discussion

In Section 3.5 and 3.6, the performance of CGaP has been comprehensively evaluated on algorithm platforms and hardware platforms. In this section, we provide a more in-depth understanding of the growth to explain why selective growth is able to improve the performance from the traditional pipelines. Furthermore, we provide a thorough ablation study to validate the robustness of the proposed CGaP method.

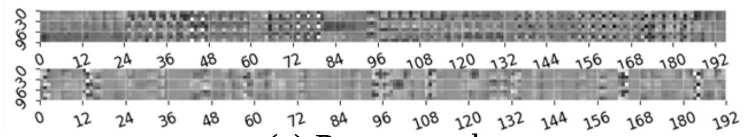
Understanding the growth Figure 3.9 illustrates a visualization of the weights in the bottom conv-layer (conv1_1) in VGG-19, at the moment of initialization, after the first growth, after the last growth and when training ends. Inside each figure, the upper bar is the CGaP model, whose size varies at different training moments. The lower bar is from the baseline model, whose size is static during training. At the initialization moment (Figure 3.9(a)), CGaP model only has 8 filters in this layer while the baseline model has 64 filters. Then the number of filters grows to 13 after one iteration growth (Figure 3.9(b)), meaning the most important 5 filters are selected



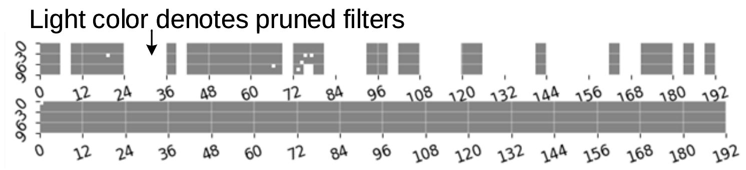
(a) Initial seed



(b) After The first growth



(c) Post-growth



(d) Final model (binary)

Figure 3.9: Visualization of the Filters in conv1_1 in VGG-19 on CIFAR-100 at Four Specific Moments (a-d). Inside Each Figure, the Top Bar Is CGaP Model and the Bottom Bar Is Baseline Model. X-axis Is the Index of Output-wise Weights and Y-axis Is the Index of Input-wise Weights.

and added. It is clear that the pattern in Figure 3.9(b) is more active than that in (a), indicating the filters have already fetched effective features from the input images. More important, along with the growing, the pattern in CGaP model becomes more structured than that in the baseline model, as shown in Figure 3.9(c). Benefiting from this well-structured pattern, our CGaP model has higher learning accuracy than the

Initial seeds		**‘2’**	**‘4’**	**‘6’**	**‘8’**	**‘10’**	**‘12’**
#filters	conv1_n	2	4	6	8	10	12
	conv2_n	4	8	12	16	20	24
	conv3_n	8	16	24	32	40	48
	conv4_n	16	32	48	64	80	96
	conv5_n	16	32	48	64	80	96
Initial #param (M)		0.01	0.06	0.13	0.23	0.36	0.53
Testing accuracy*		-0.69%	-0.2%	-0.16%	+0.37%	+0.04%	0.29%

*Relative accuracy of the final model on CIFAR-100 as compared to the baseline.

Table 3.5: The Impact of Various Structures and and Sizes of the Initial Seed of VGG-19.

baseline model. From Figure 3.9(c) to Figure 3.9(d), relatively unimportant filters are removed, and important ones are kept. We observe that most of the filters that are favored by the growth, such as filters at index 36, 48, 72, 96 in Figure 3.9(c), are still labeled as important filters in Figure 3.9(d) even after a long training process between the growth phase and the pruning phase. Leveraging the growth policy, the model is able to recover quickly from the loss caused by pruning (the spiking in Figure 3.7).

Robustness of the seed The performance of CGaP is stable under the variation of the initial seeds. To prove this, we scan several seeds in different size and present the variation in accuracy and inference model size. The structure of 6 scanned seeds is listed in Table 3.5. Each seed has a different number of filters in each layer, e.g., seed ‘2’ has 2 filters in block conv1. The size of the seeds varies from 0.01M to 0.53M. Figure 3.10 presents the final model size and the number of growth of each seed. A larger seed leads to a larger final model but requires fewer iterations of growth to reach the intended model size. Generally speaking, there is a trade-off between the

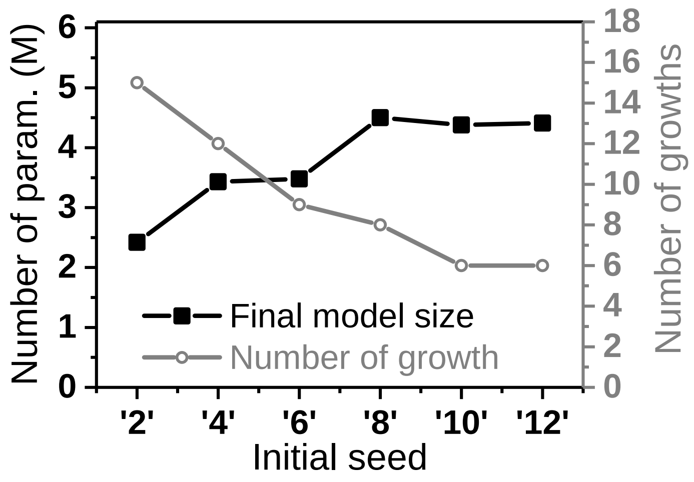


Figure 3.10: A Larger Seed Leads to a Larger Final Model but Fewer Iterations in the Growth Phase.

inference accuracy and the model size. Though the seed varies a lot from each other, the final accuracy is quite robust, as listed in the 'Accuracy' row in Table 3.5. It is worth mentioning that, even though the seed '2' degrades the accuracy of 0.69% from baseline, the inference model size is only 2.4M, significantly smaller than the baseline size (20.4M).

Robustness of the hyper-parameters CGaP is conditioned on a set of hyper-parameters to achieve optimal performance, while it is stable under the variation of these hyper-parameters. Empirically, we leverage the following experience to perform parameter optimization: a smaller growth rate β for a larger seed and vice versa; threshold τ_{capa} is set based on the user's intended model size; a smaller f_{growth} for a complicated dataset and vice versa; a relatively greedy growth (larger β and f_{growth}) prefers a larger noise μ but smaller σ to push the model away from sticking at a local minimum. Tuning of the pruning ratio of each layer is in a similar manner to that of the other pruning works [38] [39].

In particular, we scan 121 combinations of the scaling factor σ and noise μ in the range [0.0, 1.0] with the step=0.1 and provide the following discussion. For VGG16 on CIFAR-10, the accuracy of several corner cases are 90% (μ=1, σ=0, which is a case of random initialization), 89% (μ=1, σ=1), 84% (μ=0, σ=1, which is another case of mimicking its neighbor without scaling) and 10% (μ=0, σ=0, the training is invalid in this case), 93% (μ=0, σ=0, which is another case of mimicking its neighbor with scaling), 88% (μ=0.5, σ=0, which is another case of random initialization). The best accuracy of 93.6% is under μ=0.1, σ=0.5. The combinations in the zone that $\mu \in [0, 0.5]$ and $\sigma \in (0, 0.5]$ always provide >92% accuracy. To summarize, σ impacts more than μ as μ is relatively small; σ should not be too large and 0.5 is safe for future tasks and networks; adding a noise improves the accuracy (like from 93% to 93.6%) as it prevents local minimum; inheriting from the neighbor is more efficient than randomly initializing since the network is able to resume the learning right after the growth.

3.8 Conclusion and Future Work

Modern DNNs typically start training from a fixed and over-parameterized network, which leads to redundancy and is lack of structural plasticity. We propose a novel dynamic training algorithm, Continuous Growth and Pruning, that initializes training from a small network, expands the network width continuously to learn important learning units and structures and finally prunes secondary ones. The effectiveness of CGaP depends on where to start and stop the growth, which learning unit (filter and neuron) should be added, and how to initialize the newborn units to ensure model convergence. Our experiments on benchmark datasets and architectures demonstrate the advantage of CGaP on learning efficiency (accurate and compact). We further validate the energy and latency efficiency of the inference model generated by CGaP on FPGA performance simulator. Our approach and analysis will help shed light on the development of adaptive neural networks for dynamic tasks such as continual and lifelong learning.

Chapter 4

SINGLE-NET CONTINUAL LEARNING WITH PROGRESSIVE SEGMENTED TRAINING

4.1 Introduction

The rapid advancement of computing and sensing technology has enabled many new edge applications, such as the self-driving vehicle, the surveillance drone, and the robotic system. Compared to conventional edge devices (*e.g.* cell phone or smart home devices), these emerging devices are required to deal with much more complicated and dynamic situations with limited power budget. One of the necessary attributes is the capability of efficient continual learning (*i.e.* online learning): when encountering a sequence of tasks over time, the edge device should capture the new observation and update its knowledge (*i.e.* the network parameters [67, 68]) in real time, without interfering or overwriting previously acquired knowledge, and such a learning should be computationally efficient at the edge. Recent literature [69–76] have intensively studied this topic. It is believed that, to achive efficient online learning, such an edge computing system should have the following features:

Online adaption. The system should be able to update its knowledge according to a continuum of data, without independent and identically distributed (i.i.d.) assumption on this data stream. For a dynamic system (*e.g.* a self-driving vehicle), it is preferred that such adaption is completed locally and in real time.

Preservation of prior knowledge. When new data arrives in a stream, previous data are very limited or even no longer exist. Yet the acquired knowledge from previous data should not be *forgotten* (*i.e.* overwritten or deteriorated due to the learning of new data). In other words, the prior distribution of the model parameters should be preserved.

Single-head evaluation. The system should be able to differentiate the tasks and achieve successful inter-task classification without the prior knowledge of the task identifier (*i.e.* which task current data belongs to). In the case of single-head, the neural network output should consist of all the classes seen so far. In contrast, multi-head evaluation only deals with intra-task classification where the network output only consists of a subset of all the classes. Multi-head classification is more appropriate for multi-task learning than continual learning [69].

Resource constraint. Due to the limited power and memory budget at the edge, the resource usage such as the model size, the computation cost, and storage requirements should be bounded during continual learning from sequential tasks, rather than increasing proportionally or even exponentially over time.

(1) Dynamic network structure. These methods [75–80] usually expand the new knowledge by growing the network structure. For example, [77] progressively adds new network branches for new tasks and keeps previously learned features in lateral connections. In this case, prior knowledge and new knowledge are usually separated into different feedforward paths. Moreover, the newly added branches have never been exposed to the previous data and thus is blind to previous tasks. Due to these fundamental reasons, the performance of dynamic architectures on the single-head classification lags behind, although they were able to maintain the accuracy in multi-head classification with the priori of task identification. The second family is **(2) single network structure**. In contrast to a dynamic structure, these methods learn sequential tasks with a single, static network structure all the time. The knowledge of prior and new tasks are packed in a single network that is exposed to all tasks over time. In this case, the challenge is shifted to minimizing the interference among tasks and preserving prior knowledge in the same network. As a contemporary neural network has a large capacity to accommodate multiple tasks, we believe a single network provides a promising basis for continual learning.

In the family of the single-network methods, previous works have explored the regularization methods [67,68,70,81,82], the parameter isolation methods [83,84] and the memory rehearsal methods [71,73,74,85,86]. The regularization methods leverage a penalty term in the loss function to regularize the parameters when updating for new tasks. However, as more and more tasks appear, the parameters tend to be biased towards the new tasks, and the system gradually drifts away from previous distribution. To mitigate such a knowledge asymmetry, regularization methods can be combined with memory rehearsal methods [87,88]. Recent works such as iCaRL [74] and GEM [73] have proven the efficacy of replaying the memory (*i.e.* train the system with a subset of the previously seen data) in abating the network parameters drifting far away from previous knowledge. Parameter isolation approaches [83, 84] allocate subsets of parameters for previous tasks and prune the rest for learning new tasks. In this case, the rest of the parameters no longer contain prior knowledge, violating the aforementioned properties of an ideal continual learning system. For instance, PackNet [83] and Piggyback [84] achieve strong performance on multi-head evaluation but not on single-head.

To achieve continual learning with the preservation of prior knowledge, we propose single-net continual learning with Progressive Segmented Training, namely PST, as shown in Figure 3.2. When new data comes in, PST adapts the network parameters with memory-assisted balancing, then important parameters are identified according to their contribution to this task. Next, to alleviate catastrophic forgetting, PST performs model segmentation by reinforcing important parameters (through retraining) and then freezing them in the future training; while the secondary parameters will be saved (not pruned) and updated by future tasks. Through experiments on CIFAR-10 [25] and CIFAR-100 [25] dataset with modern deep neural networks, we demonstrate that PST achieves state-of-the-art single-head accuracy and successfully preserves the previously acquired knowledge in the scenario of continual learning. Moreover, benefiting from model segmentation, the amount of computation needed to learn a new task keeps reducing. This property brings PST high efficiency in computation as compared to other regularization methods. We prove the efficiency of PST with simulated results.

The contribution of this paper is as follows:

- We summarize important features of a successful continual learning system and propose a novel training scheme, namely Progressive Segmented Training (PST), to mitigate catastrophic forgetting in continual learning.

- Different from previous works in which new observation overwrites the entire acquired knowledge, PST leverages parameter segmentation for each task to prevent knowledge overwriting or deterioration. Experiments on CIFAR-10 and CIFAR-100 dataset prove that PST successfully alleviates catastrophic forgetting and reaches state-of-the-art single-head accuracy in the learning of streamed data.

- We present the advantage of PST in the scenario of edge computing from the perspective of accuracy and computation cost. With the FPGA-based 16-bit fixed-point training accelerator, we further validate that PST significantly reduces computational cost when learning at the edge.

- We demonstrate a detailed ablation study and discussion to analyze the role of each component in PST.

The rest of this paper is organized as follows: Section 4.2 describes previous efforts on continual learning. Section 4.3 describes the training routine of PST as well as a detailed description of each component. Section 4.4 demonstrates in-depth analysis of PST on CIFAR-10 and extensive results on CIFAR-100 when learning streamed tasks. Section 4.5 emphasizes the efficiency of PST when learning at the edge. Section 4.6 presents the ablation study of each components in PST and memory budget. Finally, we conclude this work in Section 4.7.

4.2 Related Work

In this section, we review previous efforts to alleviate catastrophic forgetting in continual learning. Basically, there are two families of these works: (1) dynamic network structure and (2) single network structure.

Dynamic network structure Methods with expandable or growing network structures are categorized in this family. [77] progressively adds a new branch of neural networks for each new task and leaves the old knowledge untouched. [79] expands fixed amount of neurons to learn new knowledge and partially retrains weights that are associated with old tasks. [75] combines two individual models that are trained on old and new classes through dual distillation. [78] uses reinforcement learning to adaptively expand the each layer of network when new task arrives. Due to the nature of dynamic structures, the inference of old and new tasks are separated in different paths and thus, these methods usually perform better on multi-head protocol with task identification provided. Compared to the dynamic network family, the proposed PST encodes entire knowledge of all the tasks into a single networks in order to achieve single-head evaluation.

Single network structure In contrast to dynamic family, some previous work embody all the tasks in a single network, *i.e.*static network structure. Techniques such as regularization, parameter isolation and memory rehearsal (including pseudo memory) are explored.

Regularization. [67, 68, 83] add penalty term in the objective function to regularize the parameter updating for new tasks, or use knowledge distillation [82, 89, 90] and bias correction [90] to constrain the learning between new and old classes. Along with learning more and more tasks, network parameters gradually drift away and become biased towards new tasks since regularization is soft constraint on parameter updating. Different from them, PST does not require additional term in loss function and applies hard constraint on parameter updating rather than soft constraint.

Parameter isolation. PackNet [84] iteratively prunes unimportant weights and fine-tunes them in the learning of new tasks. Similarly, Piggyback [83] prunes network parameters with learning a mask from network quantization. PackNet [83] and Piggyback [84] achieve strong performance on multi-head evaluation but not on single-head. We argue that pruning secondary parameters is sub-optimal in the case of single-head protocol since pruning destroys parameter distribution. Detailed discussion is provided in Section 4.4.1. Different from PackNet and Piggyback, PST implements segmentation by consolidating important parameters for past tasks and saving secondary parameters for new tasks. In other words, new tasks are learned from scratch (weights are zero) in PackNet and Piggyback and thus, old tasks and new tasks are disjoint; but in PST, new tasks are learned based on old tasks so that weight distribution can be preserved.

Memory rehearsal and pseudo memory rehearsal. To mitigate knowledge bias towards new tasks, some methods store previous data and retrain them [71, 73, 74, 85, 86], or train Generative Adversarial Networks (GANs) to generate and

Algorithm 4 PST training routine

```
Input: $\{X^s, \dots, X^t\}$
Require $\Theta = (\Theta_{fixed}; \Theta_{free})$
Require $\mathcal{P} = (P_1, \dots, P_{s-1})$
1: Memory-assisted training and balancing: $\Theta_{free} \rightarrow \Theta'_{free}$
2: Importance sampling: identify $\Theta_{important}$ in $\Theta'_{free}$
3: Model segmentation: $\Theta_{important} \rightarrow \Theta'_{important}$
4: $(\Theta_{fixed}; \Theta'_{important}) \rightarrow \Theta'_{fixed}$
5: $\Theta_{secondary} \rightarrow \Theta'_{free}$
Output: $\Theta' = (\Theta'_{fixed}; \Theta'_{free})$
Output: $\mathcal{P}' = (P_1, \dots, P_t)$
```

discriminate images and then learn the data distribution [91–93]. Memory rehearsal methods require additional storage to store previous data or extra model parameters to generate and discriminate data. However, the scalability is not a concern as long as the storage or the GAN model size is constrained in the learning of streamed data.

4.3 Method

In this section, we first describe the terminology and algorithm of PST. Then we interpret three major components: memory-assisted training and balancing, importance sampling and model segmentation in Section 4.3.2, Section 4.3.3 and Section 4.3.4, respectively.

4.3.1 Overview of PST

Terminology The continual learning problem can be formulated as follows: the machine learning system is continuously exposed to a stream of labeled input data

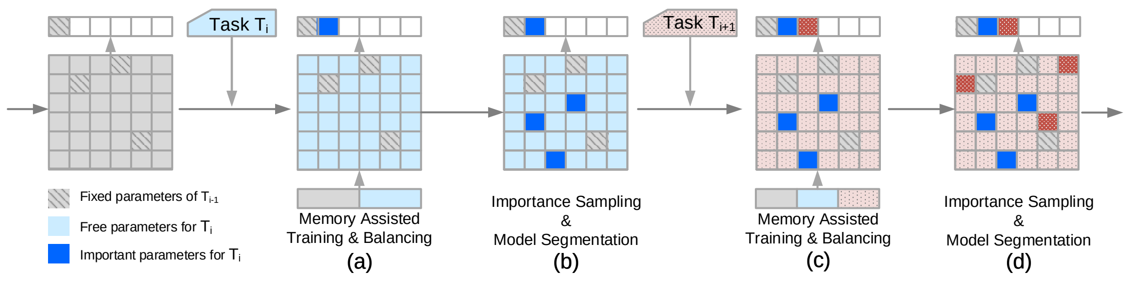

Task T_i
Task T_{i+1}
Fixed parameters of T_{i-1}
Free parameters for T_i
Important parameters for T_i
Memory Assisted Training & Balancing
(a)
Importance Sampling & Model Segmentation
(b)
Memory Assisted Training & Balancing
(c)
Importance Sampling & Model Segmentation
(d)

Figure 4.1: The Flow Chart of Progressive Segmented Training (PST). (A) We Allow the Current Task T_i and a Memory Set to Update the Free Parameters Θ_{free} (in Light Blue) in the Network While Sharing Fixed Parameters θ_{Fixed} (in Grey) Learned from Previous Tasks. The Fixed-size Memory Set Is Used to Keep the Balance of Training among Various Tasks. (B) We Sort and Select Important Parameters $\Theta_{important}$ (in Dark Blue) for Task T_i, and Reinforce Them by Retraining. These Important Parameters Are Kept Frozen and Will Not Be Updated by Future Tasks. Different from PackNet And Piggyback, the Secondary Parameters (in Light Blue) Are Not Pruned in PST. Instead, New Tasks Will Start from Secondary Parameters and Update the Network, Which Is Essential to Achieve Single-head Classification. For a New Task T_{i+1}, the above Training Routine Repeats in (c) and (d), so on and so Forth.

$X^1, X^2, \ldots,$ where $X^y = \left\{x_1^y, \ldots, x_{n_y}^y\right\}$ correspond to all examples of class $y \in \mathbb{N}$. When the new task $\{X^s, \ldots, X^t\}$ comes in, the data of old tasks $\{X^1, \ldots, X^{s-1}\}$ are no longer available, except a small amount of previously seen data stored in the memory set $\mathcal{P} = (P_1, \ldots, P_{s-1})$.

For a modern deep neural network such as VGG-Net [22] and ResNet [24], the network parameter Θ usually consists of feature extractor $\varphi : \mathcal{X} \to \mathbb{R}^d$ and classification weight vectors $w \in \mathbb{R}^d$. The network keeps updating its parameter Θ according to the previously seen data $\mathcal{X}$, in order to predict labels $\mathcal{Y}^*$ with its output $\mathcal{Y} = w^\top \varphi(\mathcal{X})$. During training the network with data corresponding to classes $\{X^1, \ldots, X^{s-1}\}$, our target is to minimize the loss function $\mathcal{L}(\mathcal{Y}; \mathcal{X}_{s-1}; \Theta)$ of this $(s-1)$-class classifier. Similarly, with the introduction of a new task with classes $\{X^s, \dots, X^t\}$, the target now is to minimize $\mathcal{L}(\mathcal{Y}; \mathcal{X}_t; \Theta)$ of this t-class classifier.

Training routine Every time when a new task is available, PST calls a training routine (Figure 4.1 and Algorithm 4) to update the parameter Θ to Θ', and the memory set $\mathcal{P}$ to $\mathcal{P}'$, according to the current training data $\{X^s, \dots, X^t\}$ and a small amount of previously seen data (memory set) $\mathcal{P}$. The training routine consists of three major components: (1) memory-assisted training and balancing, (2) importance sampling and (3) model segmentation, as illustrated in the following subsections.

4.3.2 Memory-assisted Training and Balancing

Figure 4.1 illustrates PST training routine for task T_i and task T_{i+1}. In Figure 4.1a, which is the moment that task T_i comes in, the network consists of two portions: parameters Θ_{fixed} (grey blocks) are fixed for previous tasks, and parameters Θ_{free} (light blue blocks) are trainable for current and future tasks. We allow Θ_{free} to be updated for task T_i, with Θ_{fixed} included in the feedforward path. To mitigate the parameters bias towards new task, a memory set is used to assist the training. The memory set is sampled uniformly and randomly from all the classes in previous tasks, which is a simple yet highly efficient approach, as explained in RWalk work [70]. For example, if the memory budget is K and $s-1$ classes have been learned in previous tasks, then the memory set stores $\frac{K}{s-1}$ images for each class. We mix samples from this memory set with equal samples per class from the current task, *i.e.* K samples of the memory and $\frac{K}{s-1} \times (t-s+1)$ samples from current task, and provide them to the network: *(i)* for a few epochs at the beginning of the training; *(ii)* periodically (*e.g.* every 3 epochs) during training; *(iii)* for a few epochs at the end of the training to fine-tune classification layer (*i, ii, iii* are noted in Figure 4.3). In comparison to most related works, which adopt the single-stage (step ii) optimization technique, the proposed three-step optimization strategy performs much better. One of the primary reasons behind catastrophic forgetting is knowledge drift in both feature extraction and classification layers. The three-pronged strategy helps minimize this drift in the following ways: step *i* provides a well-balanced initialization; step *ii* reviews previous data and thus, consolidates previous learned knowledge for the entire network; step *iii* corrects bias by balancing classification layers, which is simple yet efficient as compared to [90] that utilizes an extra Bias Correction Layer after the classifier. After memory-assisted training and balancing, the network parameters are updated from $\Theta = (\Theta_{fixed}; \Theta_{free})$ to $\Theta' = (\Theta_{fixed}; \Theta'_{free})$, as stated in Algorithm 1 line 1.

4.3.3 Importance Sampling

After the network has learned on task T_i, PST samples crucial learning units for the current task: for feature extraction layers (*i.e.* convolutional layers), PST samples important *filters*; for fully-connected layers, PST samples important *neurons*. The definitions of *filter* and *neuron* are as follows:

The l-th convolutional layer can be formulated as: the output of this layer $\mathcal{Y}_l = \mathcal{X}_l * \Theta_l$, where $\Theta_l \in \mathbb{R}^{O_l \times I_l \times K \times K}$. The set of weights that generates the o-th output feature map is denoted as a *filter* Θ_l^o, where $\Theta_l^o \in \mathbb{R}^{I_l \times K \times K}$. The l-th fully-connected layer can be represented by: $\mathcal{Y}_l = \mathcal{X}_l \cdot \Theta_l$, where $\Theta_l \in \mathbb{R}^{O_l \times I_l}$. The set of weights Θ_l^t that connected to the t-th class can be denoted as a *neuron*, where $\Theta_l^t \in \mathbb{R}^{1 \times I_l}$.

The filter/neuron sampling is based on an importance score that is adopted in PST to measure the effect of a single filter/neuron on the loss function, *i.e.* the importance of each filter/neuron. The importance score is developed from the Taylor Expansion of the loss function. Previously, Molchanov *et al.* [62] applied it on pruning secondary parameters. The importance score represents the difference between the loss with and without each filter/neuron. In other words, if the removal of a filter/neuron leads to relatively small accuracy degradation, this unit is recognized as an unimportant unit, and vice versa. Thus, the objective function to get the filter with the highest importance score is formulated as:

$$\underset{\Theta_l^o}{argmin}|\Delta\mathcal{L}(\Theta_l^o)| \Leftrightarrow \underset{\Theta_l^o}{argmin}|\mathcal{L}(\mathcal{Y};\mathcal{X};\Theta) - \mathcal{L}(\mathcal{Y};\mathcal{X};\Theta_l^o = \mathbf{0})| \tag{4.1}$$

Using the first-order of Taylor Expansion of $|\mathcal{L}(\mathcal{Y};\mathcal{X};\Theta)-\mathcal{L}(\mathcal{Y};\mathcal{X};\Theta_l^o = \mathbf{0})|$ at $\Theta_l^o = \mathbf{0}$, we get:

$$|\Delta\mathcal{L}(\Theta_l^o)| \simeq |\frac{\partial\mathcal{L}(\mathcal{Y};\mathcal{X};\Theta)}{\partial\Theta_l^o}\Theta_l^o| = \sum_{i=0}^{I_l}\sum_{m=0}^{K}\sum_{n=0}^{K}|\frac{\partial\mathcal{L}(\mathcal{Y};\mathcal{X};\Theta)}{\partial\Theta_l^{o,i,m,n}}\Theta_l^{o,i,m,n}| \tag{4.2}$$

where $\frac{\partial\mathcal{L}(\mathcal{Y};\mathcal{X};\Theta)}{\partial\Theta_l^{o,i,m,n}}$ is the gradient of the loss function with respect to parameter $\Theta_l^{o,i,m,n}$.

Similarly, the saliency score of a neuron is derived as:

$$|\Delta\mathcal{L}(\Theta_l^t)| \simeq |\frac{\partial\mathcal{L}(\mathcal{Y};\mathcal{X};\Theta)}{\partial\Theta_l^t}\Theta_l^t| = \sum_{i=0}^{I_l}|\frac{\partial\mathcal{L}(\mathcal{Y};\mathcal{X};\Theta)}{\partial\Theta_l^{t,i}}\Theta_l^{t,i}| \tag{4.3}$$

where $\frac{\partial\mathcal{L}(\mathcal{Y};\mathcal{X};\Theta)}{\partial\Theta_l^{t,i}}$ is the gradient of the loss with respect to parameter $\Theta_l^{t,i}$.

Based on the importance score, we sort the learning units layer by layer and identify the top β units (dark blue blocks in Figure 4.1b). In the following model segmentation step, we deal with the location of important parameters, rather than the value of these parameters, which will be explained in the next subsection. β is an empirical hyper-parameter that should be approximately proportional to the complexity of the current task. For example, when incrementally learning 10 classes of CIFAR-100 at a time, β can be 10%; when learning 20 classes per task, β can be 20%.

Due to the nature of continual learning, the total number of tasks is not known beforehand, so the network can be reserved with a larger capacity in order to freeze enough knowledge for previous tasks and leave enough space for future tasks. Once the continual learning is complete, one can leverage model compression approaches [1, 38,39,58,59] to compress the model size. It is also worth mentioning that importance sampling is only performed once after each task, so that the computation cost of this step is negligible.

4.3.4 Model Segmentation and Reinforcement

After important units are sampled according to the importance score, current network parameter $\Theta' = (\Theta_{fixed}; \Theta_{important}; \Theta_{secondary})$, where Θ_{fixed} are the frozen parameters for all the previous tasks, $\Theta_{important}$ are important parameters for the current task, and $\Theta_{secondary}$ are unimportant parameters for the current task, as stated in Algorithm 4 line 2. Our ideal target is to reinforce $\Theta_{important}$ in a way such that their contribution to the current task is as crucial as possible. Previously, Liu *et al.* [48] observed that the sampled network architecture itself (rather than the selected parameters) is more indispensable to the learning efficacy. Inspired by this conclusion, we keep the Θ_{fixed} and $\Theta_{secondary}$ intact, randomly initialize $\Theta_{important}$ and retrain them with current training data assisted by memory set to obtain $\Theta'_{important}$. This step reinforces the contribution of $\Theta_{important}$ to the learning, as proved by our experimental results demonstrated in Figure 4.2 and Table 4.2. After model segmentation, $\Theta'_{important}$ along with the aforementioned Θ_{fixed} will be kept frozen in the future tasks, and $\Theta_{secondary}$ will be used to learn new knowledge.

4.4 Accuracy: learning streamed tasks

In this section, we present experimental results to verify the efficacy of PST. The experiments are performed with PyTorch [65] on one NVIDIA GeForce RTX 2080 platform.

Datasets. The CIFAR [25] dataset consists of 50,000 training images and 10,000 testing images in color with size 32 $\times$ 32. There are 10 classes for CIFAR-10 and 100 classes for CIFAR-100. In Section 4.4.1, CIFAR-10 is divided into 2 tasks, *i.e.* 5 classes per task, to provide a comprehensive analysis of PST. In Section 4.4.2 and 4.5, following iCaRL [74], CIFAR-100 is divided into 5, 10, 20 or 50 classes per task, to demonstrate extensive experiments. For each experiment, we shuffle the class order and run 5 times to report the average accuracy.

Network structures. In the following experiment, the structure and size of VGG-16 [22] we use for CIFAR-10 dataset follows [22]. The structure and size of 32-layer ResNet for CIFAR-100 dataset follows the design of iCaRL [74]. Each convolutional layer in VGG-16 and ResNet is followed by a batch normalization layer [64]. As aforementioned in Section 4.3.3, the number of classes will occur is unknown in a continual learning scenario. Thus, we leave 1.2$\times$ space at the final classification layer in the following experiments, *i.e.* 12 outputs for CIFAR-10 and 120 outputs for CIFAR-100. It is worth mentioning that the number of classes reserved at the final classification layer does not affect the overall performance, as there is no feedback from vacant classes.

Experimental setup. Standard Stochastic Gradient Descent with momentum 0.9 and weight decay 5E-4 are used for training. The initial learning rate is set to 0.1 and is divided by 10 for every 40% and 80% of the total training epochs. On CIFAR-10 and CIFAR-100 datasets, we train 180 and 100 epochs at the stage of memory-assisted training and balancing, 120 and 60 epochs at the stage of model segmentation. β is set as proportional to the complexity of the current task. For example, when incrementally learning 5 classes of CIFAR-10 at a time, β is set to 50%; when incrementally learning 10 classes of CIFAR-100 at a time, β is set to 10%. The memory storage is set as $K = 2000$ images for a fair comparison with the

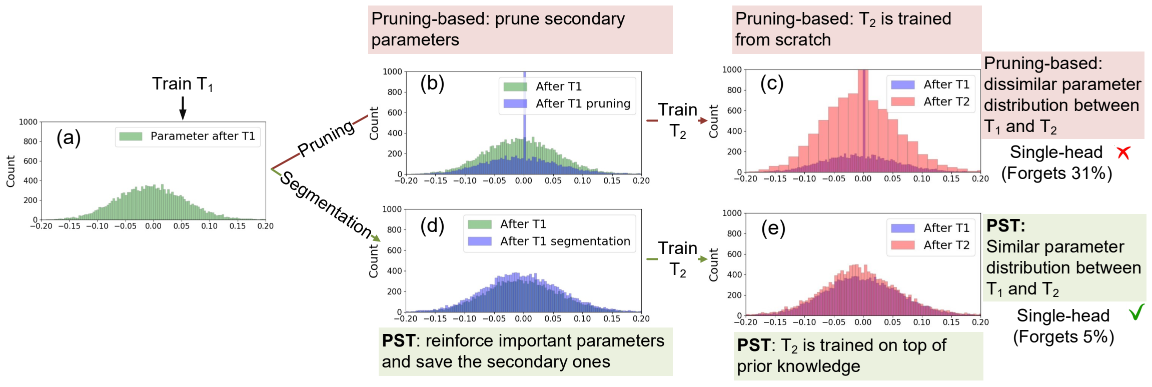


Figure 4.2: Comparison of Weight Distribution Between Pruning-based Approaches and Our PST. Pruning-based Approaches Lose Prior Knowledge Due to Pruning, and PST Preserves Prior Knowledge by Segmentation.

previous work [74].

Evaluation protocol. As mentioned in Section 4.1, single-head evaluation is more practical and valuable than multi-head evaluation in the scenario of continual learning Therefore, we evaluate single-head accuracy for the following experiments. To report the single-head *overall accuracy* if input data $\{X^1, \ldots, X^t\}$ have been observed so far, we test the network with testing data that sampled uniformly and randomly from class 1 to class t and predict a label out of t classes $\{1, \ldots, t\}$. For the *first task accuracy* (such as Fig 4.5) , we test the network with testing data collected from the first task T_1 (supposing classes $\{1, \ldots, g\}$) and predict a label out of t classes $\{1, \ldots, t\}$ to report single-head T_1 accuracy (Figure 4.5(a)); or, predict a label out of g classes $\{1, \ldots, g\}$ to report multi-head T_1 accuracy (Figure 4.5(b)).

4.4.1 In-depth Analysis

We divide CIFAR-10 into 2 tasks (5 classes each) and analyze the PST training routine step by step in this subsection. The learning curve is present in Figure 4.3.

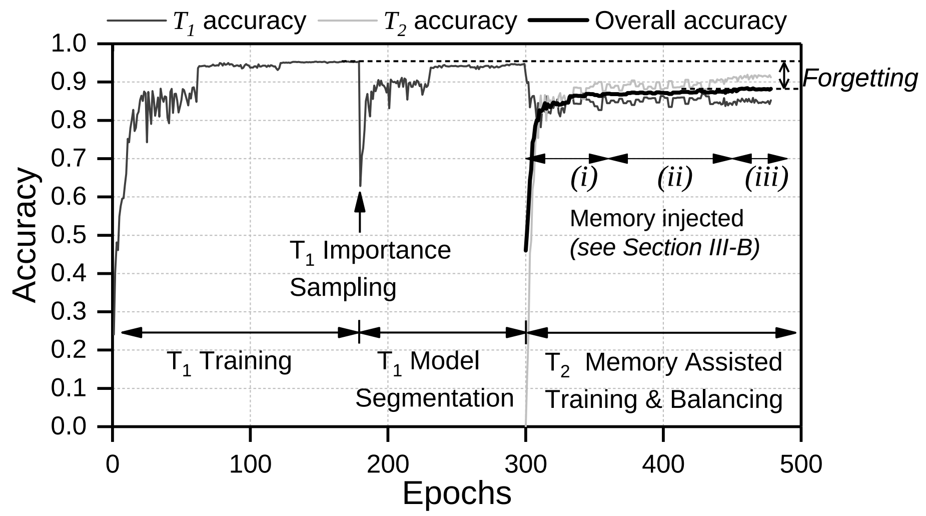


Figure 4.3: The Learning Curve of 2 Tasks on CIFAR-10 with Each Step Annotated.

From epoch 0 to epoch 180, T_1 is trained and reaches baseline accuracy. The weight distribution after training T_1 is present in Figure 4.2a. At epoch 180, we sample the top 50% (since there are 2 tasks totally) important parameters and retrain them with the secondary parameters untouched (epoch 180 to epoch 300), which is the model segmentation step. The weight distribution after this step is shown in Figure 4.2d. It is worth mentioning that previous works, such as PackNet [84] and Piggyback [83], prune the secondary parameters and thus, distort the weight distribution (Figure 4.2b). At epoch 300, task T_2 appears and updates the parameters.

At the same time, the acquired knowledge of T_1 is disturbed by T_2 updating, leading to an accuracy degradation on T_1 (see the green curve at epoch 300). From epoch 300 to the end is the step of T_2 training, during which the memory data is injected following *(i), (ii), and (iii)* to balance. After T_2 training, we again plot the weight distribution for the pruning-based approach (in Figure 4.2c) and PST approach (in Figure 4.2e). It is observed that the pruning approach fails to preserve the prior knowledge, as the weight distribution after learning T_2 shifts far away from the previous one. In contrast, PST well preserves prior knowledge (*i.e.* similar weight distribution after learning T_1 and after learning T_2). Compared to the baseline accuracy, pruning-based approaches forget 31% on overall accuracy while segmentation-based PST only forgets 5%.

4.4.2 Extensive Results

Accuracy for incrementally learning multi-classes. We compare PST with state-of-the-art approaches that reported single-head accuracy: MAS [83], EWC [67], RWalk [70], SI [68], LwF.MC [82], DMC [76], iCaRL.MC [74] and two baselines: *fixed representation*, *finetuning*. *Fixed representation* denotes the method that we fix the feature extraction layers for the previously learned tasks and only train classification layers for new tasks. *Finetuning* denotes the method that the network trained on previous tasks is directly fine-tuned by new tasks, without strategies to prevent catastrophic forgetting. LwF.MC denotes the method that uses LwF [82] but is evaluated with multi-class single-head classification. iCaRL.MC denotes the method uses iCaRL but replaces their Nearest-Mean-of-Exemplar [74] classifier with a regular output classifier for a fair comparison with PST. The results of MAS, EWC, RWalk, SI and DMC are from [76], which is implemented with the official code [1] . The results of *fix representation*, *finetune*, LwF.MC and iCaRL are from [74]. We adopt the same memory size for fair comparison between baselines and PST.

The single-head overall accuracy when incrementally learning 20 tasks (5 classes per task), 10 tasks (10 classes per task), 5 tasks (20 classes per task) and 2 tasks (50 classes per task) are reported in Figure 4.4. Among 9 different approaches, PST achieves the best accuracy on the 2-task scenario and the second best accuracy on the other scenarios. Compare to *finetuning*, PST largely prevents the model from catastrophic forgetting. Though PST achieves lower accuracy than iCaRL in some

[1]https://github.com/facebookresearch/agem

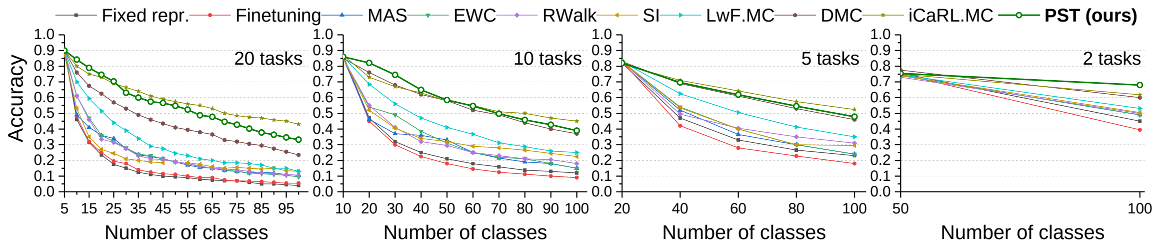
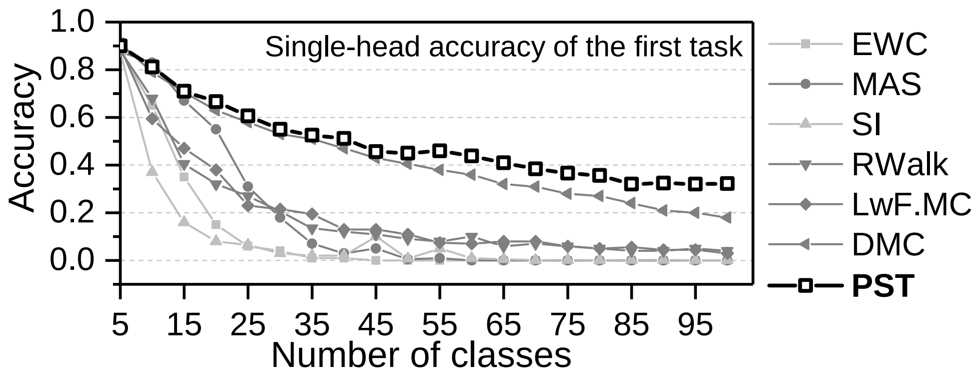


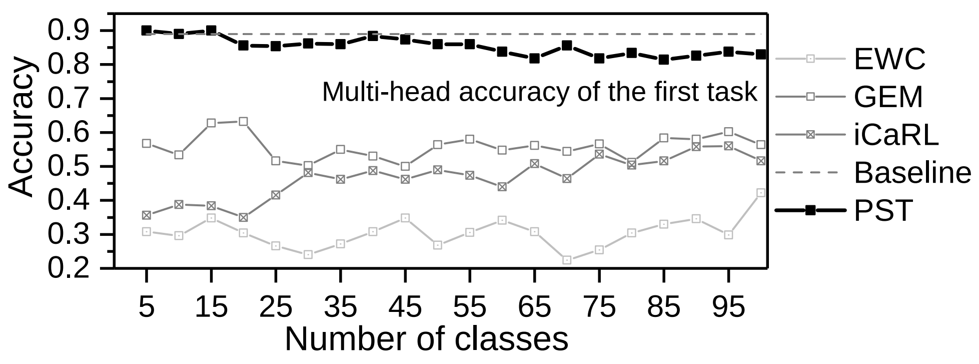
Figure 4.4: Single-head Overall Accuracy on CIFAR-100 When Incrementally Learning 20, 10, 5, 2 Tasks in a Sequence. PST Has the Best Accuracy of 2 Tasks and the Second Best Accuracy of 5, 10, 20 Tasks. Though iCaRL.MC Has Better Accuracy than PST, It Requires >24× Computation Cost than PST (See Figure 4.6 for Details)

cases, PST is more than 24× efficient in computation cost, as shown in Figure 4.6. This efficiency is benefiting from model segmentation: iCaRL has to update the entire network parameters for every new observation, but PST only requires updating partial network parameters, as the parameters related to previous tasks are frozen. Meanwhile, PST outperforms iCaRL in the multi-head protocol, as present in the following paragraph.

Accuracy of the first task. Figure 4.5(a) compares the single-head accuracy on the first task T_1 in PST with several previous approaches that reported T_1 accuracy in their papers. It also presents the multi-head accuracy on T_1 in PST and the baseline accuracy. PST achieves the best single-head accuracy on T_1 among all the approaches, *i.e.* the least forgetting. Moreover, when T_1 data is evaluated in a multi-head classification setting, as shown in Figure 4.5(b), PST is stable and always on par with the baseline (the model that is only trained on T_1, so without forgetting). This phenomenon demonstrates that PST effectively preserves the knowledge related to T_1 through model segmentation. Without these strategies, it is hard to keep the previously acquired knowledge. For example, GEM [73] reported unstable multi-head



(a) Single-head Accuracy



(b) Multi-head Accuracy

Figure 4.5: (a) Single-head Accuracy and (b) Multi-head Accuracy of the First Task T_1 over Time When the Model Is Trained with a Sequence of 20 Tasks on CIFAR-100.

T_1 accuracy, which is because the parameters gradually drift away from T_1 knowledge after a long period of learning on new tasks.

4.5 Computation Cost: Learning at the Edge

4.5.1 Simulated Results

In a more realistic situation, continual learning may not be used to train a model from scratch at the edge. Instead, we will have a model which is well trained in the cloud and once deployed, might only be required to learn a few new classes in an

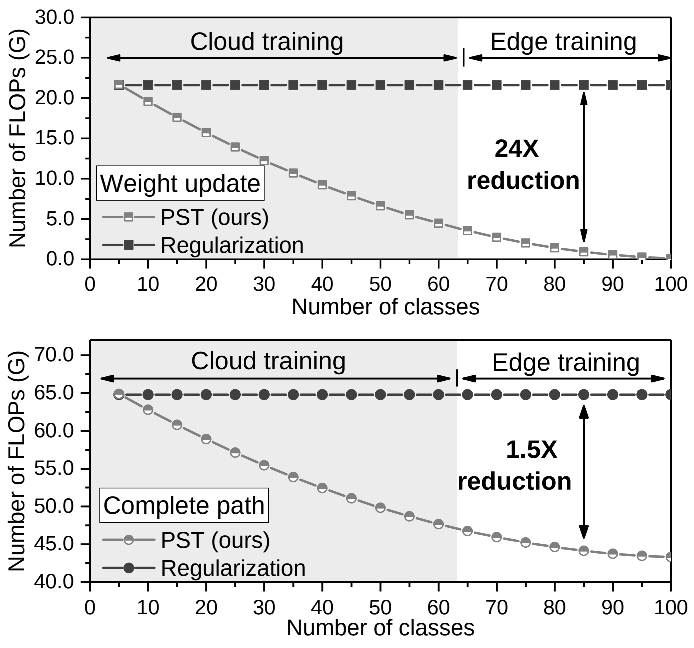


Figure 4.6: Comparison of the Computation Cost of PST and the Regularization Method. In the Scenario of Edge Learning, More than 24× Reduction in FLOPs for the Weight Update Path (Top), and 1.5× Reduction for Complete Path (Bottom) Are Achieved.

online manner on the edge devices. In this section, we developed experiments to show that PST benefits continual training at the edge from the perspective of accuracy and computation cost.

In Table 4.1, we test such a system where the base model is pre-trained (similar to training on the cloud) with 10, 30, 50, 70 or 90 classes of CIFAR-100 as task T_1 and the new task T_2 consisting of 10 disjoint classes has to be learned at the edge continually. The number of the trainable parameters for T_2 remains the same across these 5 experiments. As shown in Table 4.1, if large amounts of data have been well trained in the cloud and stored in the segmented PST model, the training of incremental data at the edge causes marginal forgetting (*e.g.* 0.08) of the acquired knowledge.

Table 4.1: With Increasing Data Trained in the Cloud, PST Effectively Mitigates Forgetting. Note That in This Experiment, the Network Size Is Much Smaller than That in Section 4.4.2.

Classes	Accuracy	Accuracy'	Forgetting
(T_1+T_2)	(after T_1)	(after T_2)	(ΔAccuracy*)
10+10	0.77	0.32	0.45
30+10	0.78	0.60	0.18
50+10	0.78	0.64	0.14
70+10	0.79	0.67	0.12
90+10	0.77	**0.69**	**0.08**

*ΔAccuracy = Accuracy-Accuracy'

Moreover, we estimate the computation cost during training, *i.e.* the number of floating point operations (FLOPs), required by PST and regularization approaches such as iCaRL [74] and EWC [67], as shown in Figure 4.6. Computation cost is a critical overhead when deploying Deep Neural Networks on edge devices [38, 39, 42, 94]. Edge learning prefers algorithms with low computation cost rather than that with higher one. Training at the edge includes three paths [95, 96], *i.e.* (1) Forward path (2) Backward path (3) Weight update path. As more and more tasks come in, the trainable parameters become fewer and fewer in PST, *i.e.*, weight update path gradually requires fewer operations, but regularization methods require a constant number of operations at all times, as the model is not segmented. Thus, given the model is pre-trained in the cloud with a large amount of data and loaded at the edge, PST reduces more than 24× FLOPs in the weight update path, and more than 1.5× FLOPs in complete path (including all three paths), as compared to the

Table 4.2: Switching off Different Components of PST Leads to Accuracy Drop to Different Extents. In the Table, Negative Numbers Indicate Accuracy Drop, *e.g.* -0.32 Means 32% Accuracy Drop. Removing Significance Sampling or Model Segmentation Leads to Significant Accuracy Drop, While Removing Memory Leads to Small Accuracy Drop. It Shows That Significance Sampling and Model Segmentation Are Indispensable Steps for PST, and Memory-assisted Balancing Is Supplementary.

Model	20 tasks	10 tasks	5 tasks
Hybrid 1 (removing significance sampling)	-0.32	-0.38	-0.45
Hybrid 2 (removing model segmentation)	-0.32	-0.38	-0.42
Hybrid 3 (removing memory balancing)	-0.06	-0.08	-0.11

regularization methods such as iCaRL [74]. Especially, weight update path usually costs 2× latency than the other two paths so that PST can largely speed up the training. Benefiting from segmentation, PST outperforms other continual learning schemes in computation efficiency.

4.6 Ablation Study and Discussion

In this section, we analyze the importance of each component in PST by performing an ablation study and demonstrate that PST is highly efficient in edge computing by virtue of single-net segmentation.

4.6.1 Analysis of Each Component in PST

We remove each component from PST and repeat the experiments performed in Figure 4.4. The overall accuracy *change* after the last task is reported in Table 4.2. Replacing significance sampling with a random sampling leads to model *Hybrid 1*;

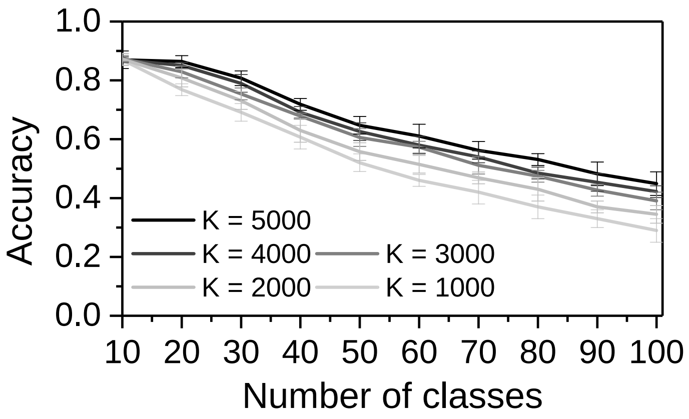


Figure 4.7: Overall Single-head Accuracy When Incrementally Learning 10 Tasks under Different Memory Budget.

removing model segmentation step (no reinforcement on $\Theta_{important}$) leads to model *Hybrid 2*; removing the memory-assisted balancing leads to model *Hybrid 3*. The results of hybrid models prove that each component in PST is contributing to the overall performance. Especially, significance sampling and model segmentation are indispensable steps for PST since removing them leads to significant accuracy drop, and memory-assisted balancing is supplementary.

4.6.2 Memory Budget

For PST, the accuracy gap between single-head and multi-head of T_1 mentioned above could be caused by the imbalance between old and new knowledge (the network is biased to new knowledge than old knowledge since old data are no longer used to train the network). Memory-assisted balancing in PST alleviates this obstacle but cannot completely prevent. Indeed, there has hitherto been no approach to prevent this knowledge asymmetry. With more data saved from previous tasks, the forgetting is reduced. But such a trend gradually saturates, as shown in Figure 4.7.

4.7 Conclusion

A successful continual learning system that is exposed to a continuous data stream should have the properties of online adaption, preservation of prior knowledge, single-head evaluation and resource constraint, to alleviate or even prevent catastrophic forgetting of previously acquired knowledge. To satisfy these properties and minimize catastrophic forgetting, we propose a novel scheme named single-net continual learning with Progressive Segmented Training (PST). Benefiting from its components (memory-assisted training and balancing, importance sampling, and model segmentation), PST achieves state-of-the-art single-head accuracy on incremental tasks on CIFAR datasets, with far lower computation cost. We further demonstrate that PST favors edge computing due to its segmented training method. In future work, we plan to study the detailed mechanism of catastrophic forgetting further and improve PST. Moreover, we plan to explore compressing or even eliminating the memory data without sacrificing the performance.

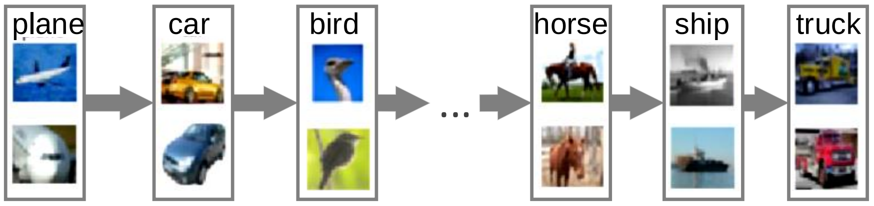
Chapter 5

ONLINE KNOWLEDGE ACQUISITION WITH SELECTIVE INHERITED MODEL

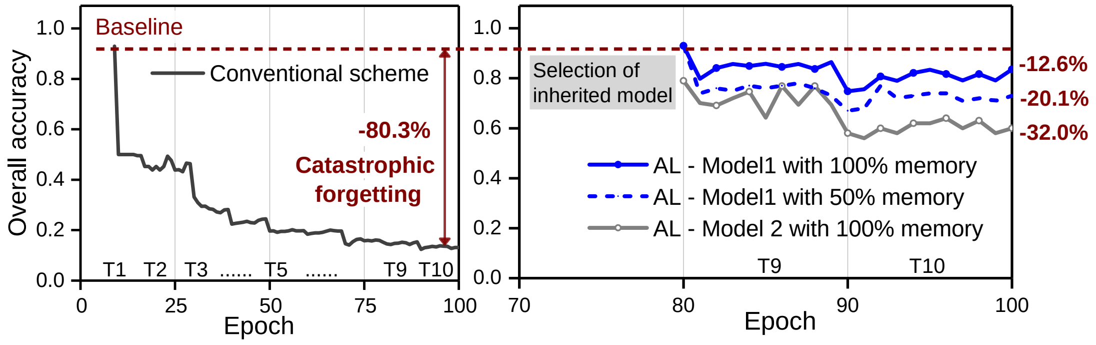
5.1 Introduction

The rapid development of machine learning algorithms and computing hardware has accelerated the implementation of many modern edge applications, such as autonomous vehicles, surveillance drones, and robots. These emerging edge devices are equipped with much more computing power than before. Meanwhile, they are required to handle more complicated and dynamic scenarios locally and in real-time, as compared to conventional edge devices such as mobile phones. One of the critical demands is the capability to learn from a data stream over time, *i.e.* the capability of *continual learning* (*a.k.a.* lifelong learning) [2,67,74,82,97]. Such a capability requires the system to learn from new observations without interfering or overwriting previous knowledge (*i.e.* model parameters). Furthermore, the learning should be bounded by computation and energy resources, including but not limited to the model size, the computation cost and storage, while still completing the process in real-time.

Today the biggest challenge in continual learning is known as *catastrophic forgetting* [98], as shown in Figure 5.1. When a model is updated to a sequence of new tasks with very limited or even no access to previous input data, prior knowledge is deteriorated, leading to severe accuracy drop (*i.e. forgetting*). While there have been multiple attempts to mitigate catastrophic forgetting [67, 73, 74, 82–84, 99], they all follow a conventional procedure of continual learning: updating the model class by class, from scratch, as shown in Figure 5.1(a). To be specific, when the network starts



(a) Conventional Continual Learning: Incrementally Learn One Class after Another from Scratch.



(b) Conventional Scheme Vs. Acquisitive Learning.

Figure 5.1: When Learning from a Data Stream of CIFAR-10, Conventional Continual Learning Suffers Catastrophic Forgetting, While the Proposed Acquisitive Learning Successfully Mitigates Such Forgetting by >6X on CIFAR-10. A Well Selective Inherited Model , Knowledge Acquisition and Memory Rehearsal All Contribute the Accurate Learning. Among Them, the Quality of the Inherited Model Is More Vital than the Amount of Memory Used to Replay. Model 1 Refers to ResNet-56 with Better Landscape; Model 2 Refers to ResNet-56-NS with Worse Landscape.

to learn new knowledge from a data stream, there is no prior knowledge embedded in this network. In this scenario, the network parameters are randomly initialized, and the learning process only focuses on model adaptation. Such a conventional learning flow is suffering from severe accuracy drop and excessive computation cost [2]. Moreover, focusing only on model adaption is not the complete picture from biology. It is

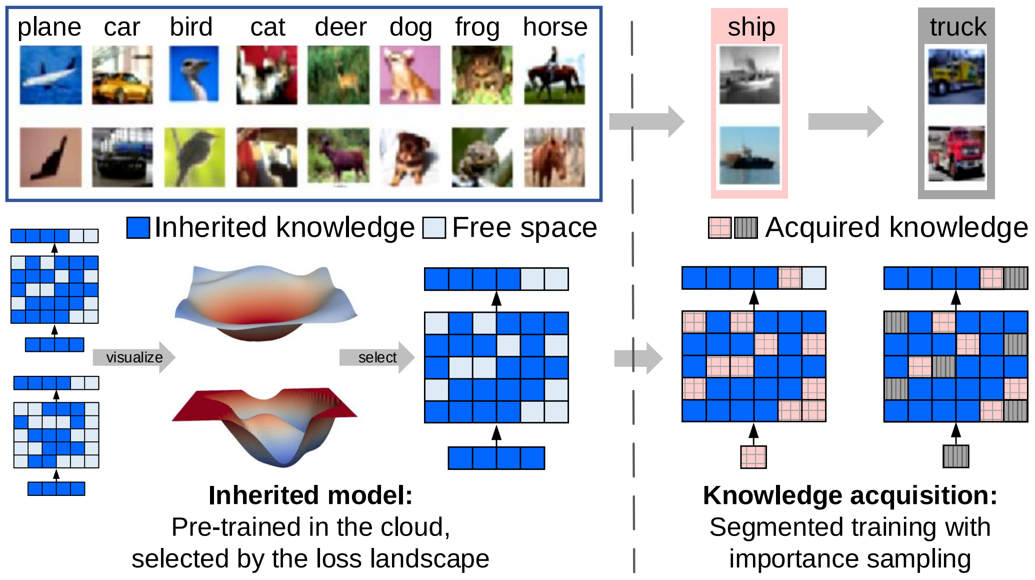


Figure 5.2: The Flow of Acquisitive Learning Emphasizes the Importance of Both Knowledge Inheritance and Acquisition.

observed in [100–102] that the brain inherits knowledge in specific neurophysiological structures (*i.e.* hardwired), through a long and careful evolution process. Besides model adaptation, the hardwired model that is selected and inherited is also critical to the quality of intelligence.

To overcome the above limitations of current continual learning, we propose a novel scheme, namely acquisitive learning (AL), as shown in Figure 5.2. Inspired by the inherited brain model, AL emphasizes the importance of both knowledge inheritance and acquisition: the majority of knowledge is first pre-trained and preserved in the inherited model, and then the model is adapted to new streaming data (the acquisition). More important, we confirm the vital correlation between *the quality of the inherited model* and its *acquisition capacity* on new knowledge. Though consolidating prior knowledge by pre-training feature extraction layers of a model has been previously explored in transfer learning [103], such a model is still suffering from accuracy drop when the feature space rarely overlaps between old and new data. Such an accuracy drop is because the one-shot pre-trained model is too stochastic to generalize better for new observations.

In this paper, we claim that the pre-trained inherited model should be elaborately selected to optimize future learning performance. Accordingly, we propose a noise-based approach to evaluate and select the inherited model with better stability. Such an approach is validated by visualizing the loss landscape [104] and measuring the roughness of the landscape with quadratic linear regression. That being said, we believe the selection criteria should not be limited to what is proposed in this paper. For the acquisition step, we leverage importance sampling from Progressive Segmented Training (PST) [2] to identify and freeze important parameters for the inherited model, and only train the secondary parameters to acquire new knowledge. In this process, a small and bounded memory set is used to retrieve the previous knowledge.

In summary, model inheritance, knowledge acquisition with importance sampling and memory replay all contribute to the final accuracy in such a learning from streaming data. The combination of these techniques reduces the accuracy drop due to catastrophic forgetting by 6.6X on CIFAR-10 (Figure 5.1(b)) and 11.5X CIFAR-100 dataset(Figure 5.8(b)), respectively. Among these techniques, selective inherited model plays an indispensable role in maintaining the accuracy, while memory replay plays a complementary role, as verified by the results in Figure 5.1(b) and in future sections. Further more, AL is efficient in hardware computation cost. AL reduces the latency per training image by 5X and overall training FLOPs by 150X as benchmarked by FPGA prototype.

To summarize, the contribution of this paper is as below:

- We propose a brain-inspired scheme for learning from streaming data, namely acquisitive learning (AL). Different from conventional continual learning that only focuses on model adaptation, AL emphasizes the importance of both knowledge inheritance and acquisition.

- With experiments on various deep neural networks and datasets, we demonstrate that the proposed AL effectively reduces catastrophic forgetting when learning from streamed data.

- Experiments show that the acquisition is strongly related to the quality of the inherited model and thus, the inherited model should be elaborately selected rather than being one-shot attained. In this paper, we leverage landscape visualization and roughness measurement to select the mode.

- We further implement the training of AL on FPGA and benchmark the significant reduction in computation cost, which enables AL on a edge device.

5.2 Preliminaries

This section presents the terminology, previous works the and biology background.

5.2.1 Terminology

A deep neural network (DNN) such as VGG-Net [22] and ResNet [24] usually consists of a feature extractor $\varphi : \mathcal{X} \rightarrow \mathbb{R}^d$ and classification weight vectors $w \in \mathbb{R}^d$, also known as convolutional layers and fully-connected layers. The network parameters Θ (φ and w) keep being updated according to input data $\mathcal{X}$, and calculating output $\mathcal{Y} = w^\top \varphi(\mathcal{X})$ in order to predict labels $\mathcal{Y}^*$.

When learning the first task with input data $\{X^1, \ldots, X^{s-1}\}$, DNN tries to minimize the loss $\mathcal{L}(\mathcal{Y}; \mathcal{X}_{s-1}; \Theta)$ of this $(s-1)$-class classifier. When a new task with input data $\{X^s, \ldots, X^t\}$ arrives, DNN tries to minimize $\mathcal{L}(\mathcal{Y}; \mathcal{X}_t; \Theta)$ of this t-class classifier

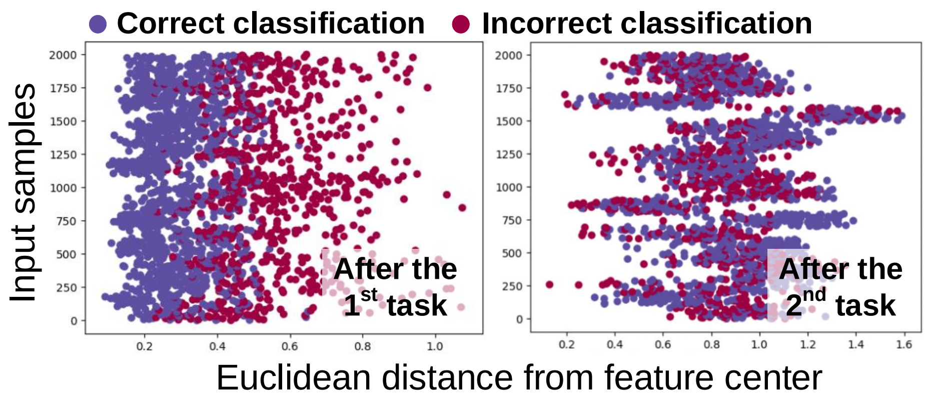


Figure 5.3: The Main Reason of Catastrophic Forgetting Is the Drift in the Feature Space. Left: Visualizing the Euclidean Distance Between $\varphi(\mathcal{X})$ Of Each Input Image and the Feature Center (*i.e.* the Normalized $\varphi(\mathcal{X})$ Of All the Input Images) after Learning 10 Classes from CIFAR-100 with ResNet-56. Wrongly Classified Samples Are Relatively Further from the Feature Center. Right: After Learning Another 10 Classes from CIFAR-100, the Feature Center Drifts so That the Correlation Between Euclidean Distance and Classification Is Deteriorated.

by updating Θ. Usually, after the input data of the new task $\{X^s, \ldots, X^t\}$ arrives, the input data of previous task $\{X^1, \ldots, X^{s-1}\}$ is no longer available, except a small subset stored in the memory set $\mathcal{P} = (P_1, \ldots, P_{s-1})$.

5.2.2 Conventional Approach of Continual Learning

The conventional approach of continual learning starts from a set of fresh, randomly initialized network parameters Θ, and each incoming task updates entire Θ or partial Θ. They leverage different techniques such as regularization [67,82], parameter isolation [83,84], memory replay [73,74], or network expansion [77,99] to mitigate catastrophic forgetting.

Regularization-based approaches add penalty term in the loss function to regularize the parameter updating space. Parameter isolation approaches assign a subset of network parameters to specific task updating. Memory replay approaches train the model with a small subset of previously seen data. Network expansion approaches expand network by adding new branches or parameters to include new knowledge. However, as the network is not inheriting any prior knowledge, each new task can easily update the weight distribution and cause feature drifting, as shown in Figure 5.3, and thus causing catastrophic forgetting. In other words, conventional approach focuses more on model adaptation to new tasks, without inheriting any prior knowledge. In contrast to them, acquisitive learning emphasizes both knowledge inheritance and acquisition.

5.2.3 Difference from Transfer Learning

It is worth some words here to differentiate transfer learning with the proposed acquisitive learning. Transfer learning (or domain adaptation) is a method where a network developed for one task is reused to learn a new task. It can be formulated as follows: for a new task with input data $\{X^s, \ldots, X^t\}$, DNN tries to minimize $\mathcal{L}(\mathcal{Y}; \mathcal{X}_{s\ldots t}; \Theta)$ of this $(t - s + 1)$-class classifier by reusing network φ pre-trained on $\{X^1, \ldots, X^{s-1}\}$ and fine-tuning classification weight vectors w. Thus, the differences between transfer learning and the proposed AL are: (1) transfer learning only focuses on the learning of new tasks while AL requires to learn new tasks as well as remember the old tasks; (2) transfer learning is usually one-time knowledge transfer, while AL requires to learn a sequential of tasks; (3) transfer learning usually freezes entire feature extractor φ and only fine-tune classification layers, limiting the acquisition of new knowledge.In AL, we only freeze selected parameters to help remember previous knowledge and leave enough Θ to acquire new knowledge; (4) transfer learning do not select pre-trained model, but directly use the one-shot trained model without quality evaluation.

5.2.4 Biology Background: Moravec's Paradox

There have been increasing evidences [100–102] showing that the brain inherits its knowledge in specific neurophysiological structures, through a long and careful evolution process, while the capability to adapt in the field is comparatively much more challenging. This was identified as the Moravec's paradox [102], and has led to research outcomes that support the hardwired model of learning. Indeed, the intelligence in nature may be determined more by the long-term genetics and inheritance rather than the short-term adaptation [100]. Therefore, to successfully learn new knowledge, the quality of both knowledge inheritance and acquisition is critical.

5.3 Acquisitive Learning

With preliminaries defined in the previous section, we describe acquisitive learning from two perspectives: model inheritance and knowledge acquisition.

5.3.1 Model Inheritance

Prepare inherited model In this subsection, we explain how to prepare the inherited model. Throughout this paper, we refer to a network that has been well pre-trained on some data as the inherited model.

Acquisitive learning first trains the network with randomly selected classes from a dataset, and then samples crucial learning units (convolution filters and fully-connected neurons) for the current task. The importance sampling is based on an important score that has been proven in [2,62]. The score is used to measure if a unit is important to the loss function.

For a filter $\Theta_l^o \in \mathbb{R}^{I_l \times K \times K}$, the score is formulated as:

$$|\Delta\mathcal{L}(\Theta_l^o)| \simeq |\frac{\partial\mathcal{L}(\mathcal{Y};\mathcal{X};\Theta)}{\partial\Theta_l^o}\Theta_l^o| \tag{5.1}$$

$$= \sum_{i=0}^{I_l}\sum_{m=0}^{K}\sum_{n=0}^{K}|\frac{\partial\mathcal{L}(\mathcal{Y};\mathcal{X};\Theta)}{\partial\Theta_l^{o,i,m,n}}\Theta_l^{o,i,m,n}|, \tag{5.2}$$

where $\frac{\partial\mathcal{L}(\mathcal{Y};\mathcal{X};\Theta)}{\partial\Theta_l^{o,i,m,n}}$ is the gradient of the loss function with respect to the parameter $\Theta_l^{o,i,m,n}$.

For a neuron $\Theta_l^t \in \mathbb{R}^{1 \times I_l}$, the score is formulated as:

$$|\Delta\mathcal{L}(\Theta_l^t)| \simeq |\frac{\partial\mathcal{L}(\mathcal{Y};\mathcal{X};\Theta)}{\partial\Theta_l^t}\Theta_l^t| = \sum_{i=0}^{I_l}|\frac{\partial\mathcal{L}(\mathcal{Y};\mathcal{X};\Theta)}{\partial\Theta_l^{t,i}}\Theta_l^{t,i}|, \tag{5.3}$$

where $\frac{\partial\mathcal{L}(\mathcal{Y};\mathcal{X};\Theta)}{\partial\Theta_l^{t,i}}$ is the gradient of the loss with respect to the parameter $\Theta_l^{t,i}$.

Based on the importance score, we sort the learning units layer by layer and identify the top β units for the inherited model. We following the same setting in [2] for β: it should be roughly proportional to the amount of the inherited knowledge. In the following adaptation, these important units are not updated but kept unchanged, in order to preserve inherited knowledge.

Noise injection After several candidate models are prepared, we use noise injection to evaluate the model stability. For each layer l in a neural network, we apply noise as below:

$$\tilde{\Theta}_l = \Theta_l + \alpha \cdot n_l, \tag{5.4}$$

where Θ_l is the noise-free weight tensor in the l-th layer, α is a constant scaling coefficient, and n_l is the noise tensor of the l-th layer that follows normal distribution $n_l \sim \mathcal{N}(0, \sigma_l^2)$ (σ_l is the standard deviation of Θ_l).

Noise injection methods have been used in other applications such as adversarial attack [105], where noise is treated as a trainable parameter during training. In our work, we perform a one-shot injection of noise to the candidate models. A drop in testing accuracy is observed for the model with noisy tensor $\tilde{\Theta}$ as compared to the model with Θ. Based on this accuracy drop, we are able to monitor the stability of the inherited model: with noise of the same σ injected, model that has a larger accuracy drop is considered less stable and vice versa. The intuition behind this claim is that a more stable model has a higher tolerance to disturbance.

Landscape visualization and roughness measurement Following the above-mentioned method, we are able to obtain inherited models with consolidated knowledge. We leverage landscape visualization tool [104] to visualize the minima of the loss function and then calculate the roughness using linear regression. In [104], filter normalization is used to remove the scaling effect, and a 3-dimension matrix (x, y, z, where x, y are the coordinates and z is the loss function) is finally extracted and plotted for visualization. To further quantify the roughness of the landscape, we fit this 3D data using quadratic linear regression and obtain mean square error (MSE) to represent the roughness:

$$\hat{z}_j = w_{j4}x_j^2 + w_{j3}y_j^2 + w_{j2}x_j + w_{j1}y_j + w_{j0} \tag{5.5}$$

$$\hat{w} = \underset{w_j}{\text{argmin}} \frac{1}{n} \sum_{j=0}^{n} (z_j - \hat{z}_j)^2, \tag{5.6}$$

where w_j represent the learned feature weights or coefficients. We define the roughness as $\text{MSE}(z; x^2, y^2, x, y; \hat{w})$. Models with smaller MSE are more flat and smooth, and vice versa. The roughness score can be used to represent model stability.

5.3.2 Knowledge Acquisition

With the inherited model fully pre-trained and important units selected, acquisitive learning leverages techniques proposed in PST [2] to learn new observations.

PST techniques include model segmentation, memory-assisted training and balancing. When a new observation arrives, only the secondary parameters (*i.e.* those are not frozen) in the inherited model are updated while the important parameters for the inherited model are frozen. In other words, the model is segmented to the inherited part and the acquisition part. Meanwhile, a small subset of data containing uniformly and randomly sampled images per class from all the trained classes so far is mixed with new observations (*i.e.* each class in $\{X^1, \ldots, X^t\}$ contains the same number of images) to train and balance the model.

By using techniques including importance sampling, model segmentation, memory-assisted training and balancing, the acquisitive learning scheme is able to acquire new knowledge based on an inherited model. It is worth mentioning that the techniques used to consolidate inherited knowledge and to acquire new knowledge are flexible. In this paper, we focus more on the acquisitive learning methodology.

5.4 Experimental Results

In this section, we develop various experiments to verify the efficacy of the proposed acquisitive learning flow.

5.4.1 Experiment Setup

The experiments in Section 5.4 B-D are performed with pytorch [65] on one NVIDIA GeForce RTX 2080 platform. We use stochastic gradient descent with momentum of 0.9 and weight decay of 0.0005. For each experiment, we shuffle the class order and run 5 times to report the average accuracy. In Section 5.4.5, Intel Stratix-10 GX equipped with the 4Gb DDR3 with 17Gb/s bandwidth was used as the target hardware. Latency was measured using simulation of the CNN training accelerator [106] at 240MHz.

Datasets The CIFAR dataset [25] consists of 50,000 training images and 10,000 testing images in color with size 32×32. CIFAR-10 consists of 10 classes, and CIFAR-100 consists of 100 classes. In the following experiments, we first train a subset of classes to produce the inherited model, and then we treat the unseen classes as new knowledge that needs to be acquired. The balanced memory set contains 200 and 20 images for each class for CIFAR-10 and CIFAR-100, respectively, so that the total memory size is bounded within 2,000 images for both CIFAR-10 and CIFAR-100 datasets , aligning with previous works [2, 74].

Network structure The network structures of VGG-16 [22], ResNets [24], DenseNet [107] used in the following experiment are standard structures following [104]. Since the total number of classes is unknown in a real-world application, we leave $1.2\times$ space at the final classification layer in the following experiments, *i.e.* 12 outputs for CIFAR-10 and 120 outputs for CIFAR-100. Note that the extra space reserved at the final classification layer does not affect the evaluation since there is no feedback from vacant outputs.

Evaluation protocol 'Pre-trained accuracy' or 'accuracy of the inherited model' refers to the testing accuracy of $(s-1)$-class classifier if input data is $\{X^1, \ldots, X^{s-1}\}$. 'Accuracy on the new task' refers to the testing accuracy of $(t-s+1)$-class classifier for input data $\{X^s, \ldots, X^t\}$ as new observations. 'Overall accuracy' refers to the testing accuracy of t-class classifier on all the data seen so far. 'Accuracy forgetting' refers to the accuracy drop from pre-trained accuracy to overall accuracy during continual learning.

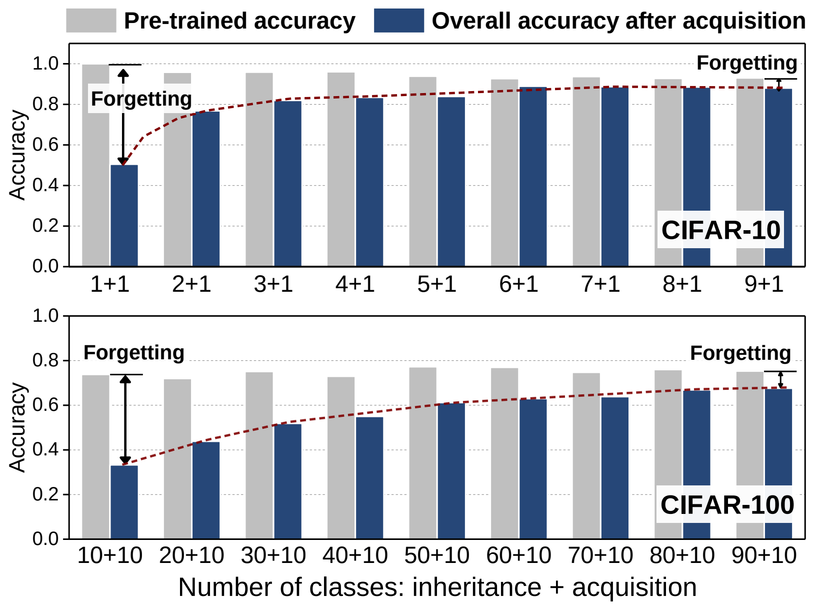


Figure 5.4: Accuracy Drop Is Minimized with the an Increasing Amount of Knowledge in the Inherited Model. Top: VGG-16 on CIFAR-10 Dataset. Bottom: ResNet-56 on CIFAR-100 Dataset. 'X+Y' Means That X Classes Are Pre-trained and Y Classes Are Learned as the New Acquisition. There Is No Overlapping of Classes in X and Y.

5.4.2 Amount of Inherited Knowledge

We first explore whether and how the amount of inherited knowledge impacts the acquisition capacity. We mimic the different amount of inherited knowledge using different numbers of pre-trained classes and plot the results in Figure 5.4. 'X+Y' refers to the scenario when X classes are pre-trained in the inherited model and Y classes need to be acquired. For the new task Y, we use the same number of classes across experiments and keep the number of active filters/neurons for this new task the same. The inherited model size (frozen filters/neurons) is proportional to the number of classes in X across experiments. In Figure 5.4 (top) for '1+1' case on CIFAR-10 with VGG-16 network, the accuracy drops from 100% to 50.5% (49.5% forgetting);

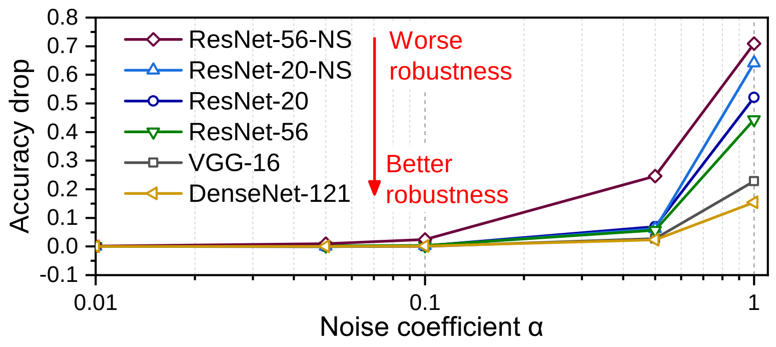


Figure 5.5: After injecting noise with the same α, more accuracy drop is observed for less stable models, and vice versa. This ranking is consistent with that from the landscape visualization and roughness measurement shown in Figure 5.6.

but for '9+1' case, the accuracy drops from 93.0% to 88.0% (5.0% forgetting). In Figure 5.4 (bottom), the accuracy forgetting is 40.5% for '10+10' case but only 7.7% for '90+10' case. It is concluded that, with more knowledge embedded in the inherited model, less forgetting is observed for acquisition, and such a trend gradually saturates.

5.4.3 Quality of the Inherited Model

Besides the amount of inherited knowledge, the inherited model itself is also a critical factor in acquisitive learning. After the preparation of several candidate models, we inject noise to each model following Equation 5.4 with $\alpha = 0.01$, 0.05, 0.1, 0.5 and 1.0, and document the accuracy drop caused by this disturbance in Figure 5.5. For example, with noise of $\alpha = 1.0$ injected, ResNet-56 without shortcuts (ResNet-56-NS) drops 70.9% in accuracy, while DenseNet-121 drops 15.4% in accuracy. Thus, we consider the former model is worse than the latter one in model stability.

As explained in [104], different deep learning models have a different landscape of the loss function, where wide and flat minima generalizes better and sharp minima with many small regions of convexity generalizes more poorly. The quality of the

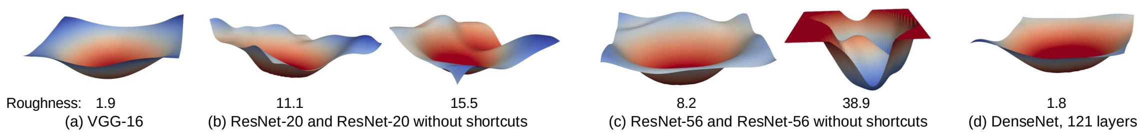


Figure 5.6: Landscape Visualization of the Loss Function for 6 Models. Shallow Models (like VGG-16) Have Smooth Landscapes. Deep Models with Shortcuts Has Smoother Landscapes than the One Without Shortcuts.

landscape is influenced by model depth, model size, batch size, and skip connections (*i.e.*'shortcuts') between layers. We select and plot six models that are pre-trained on the same 9 classes of CIFAR-10 but with different landscapes in Figure 5.6 and their corresponding roughness measurement in Table 5.1: VGG-16, ResNet-20 with and without shortcuts, ResNet-56 with and without shortcuts, and DenseNet-121. Among them, VGG-16, ResNet-20, ResNet-56 and DenseNet-121 have relatively flat landscapes and thus lower roughness; ResNet-20 without shortcuts (ResNet-20-NS) and ResNet-56 without shortcuts (ResNet-56-NS) have relatively sharp landscapes and higher roughness. The landscape of ResNet-56-NS is the most chaotic one. Note that these six models exhibit the same amount of inherited knowledge (9 classes) but show different quality in acquisition.

For each of these six inherited models, we add one new class to acquire and report the accuracy in Table 5.1. The first row shows the accuracy of the inherited models on 9 classes, and the second row represents the accuracy on the new task. We focus more on the relative accuracy between the first row to the second row, shown in the row 'accuracy drop', as this data represents the generalization ability of the pre-trained model on new observations, *i.e.* the acquisition capacity of the inherited model. On ResNet-20-NS and ResNet-56-NS models that have sharper landscapes, we observe 9.1% and 19.3% accuracy drop, respectively. This drop is more severe as compared to

Table 5.1: Acquisition Capacity for Different Models Shown in Figure 5.6. CIFAR-10 '9+1' Is Used Here.

Network	VGG-16	ResNet-20	ResNet-20-NS	ResNet-56	ResNet-56-NS	DenseNet-121
Pre-trained accuracy	0.927	0.915	0.901	0.923	0.790	0.935
Accuracy on the new task	0.865	0.851	0.810	0.860	0.597	0.883
Accuracy drop	0.062	0.064	0.091	0.063	*0.193*	**0.052**
Roughness ($\times 10^{-3}$)	1.9	11.1	15.5	8.2	*38.9*	**1.8**

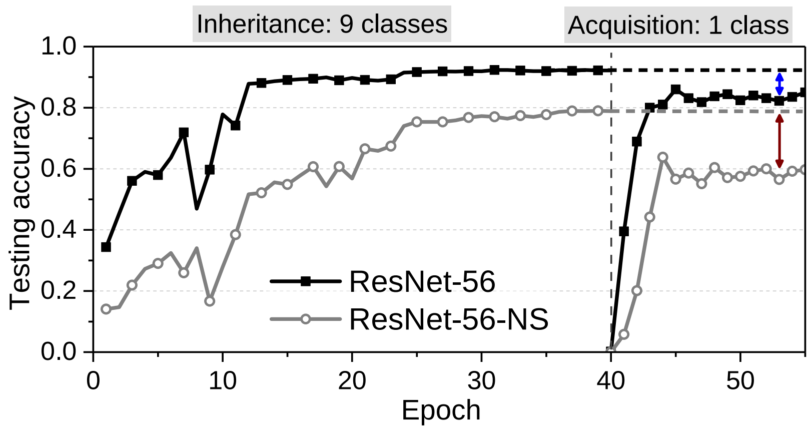


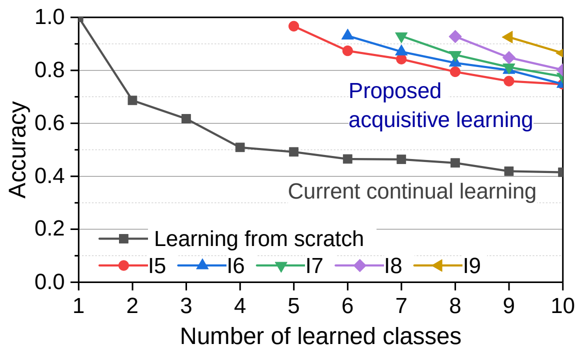
Figure 5.7: Learning Curve for '9+1' Experiment on CIFAR-10 with Two Models. Because of the Smoother Landscape of ResNet-56, Its Acquisition on New Task Is Better than ResNet-56-NS.

other models, indicating that the knowledge acquisition capacity of these two models are poor. In Figure 5.7, we also plot the learning curve for '9+1' simulation with ResNet-56 and ResNet-56-NS. ResNet-56-NS has worse acquisition capacity on new tasks than ResNet-56. These results indicate that the quality of the inherited model is another vital factor in knowledge acquisition.

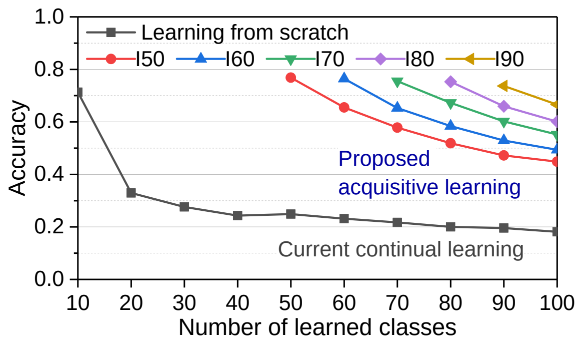
5.4.4 Learning from a Data Stream with AL

We design experiments to verify that acquisitive learning is a more effective approach to learn from streaming data as compared to current continual learning scheme. On one side, we simulate the conventional continual learning that starts learning from scratch and learns each task (1 class from CIFAR-10 or 10 classes from CIFAR-100) in a sequence. We follow the techniques described in Section 5.3.2 to learn new tasks. β is set as 0.1 for the first task. The overall accuracy of conventional method is plotted in gray in Figure 5.8(a) and Figure 5.8(b). On the other side, by assuming inherited knowledge contains much more classes than new observations, we prepare inherited models that are pre-trained on 5 to 9 classes for CIFAR-10 dataset and then incrementally train the network with 1 class from the rest of dataset (Figure 5.8(a)), following the techniques described in Section 5.3. β is set as 0.5 for the inherited model in 'I5' experiment, and 0.9 for the inherited model in 'I9' experiment, so on and so forth. Similarly, we pre-train 50 to 90 classes on the CIFAR-100 dataset and then incrementally learn 10 classes from the rest of the dataset (Figure 5.8(b)).

The results on acquisitive learning starting from different inherited models are plotted in colors in Figure 5.8. For CIFAR-10, with the conventional scheme, the final overall accuracy for 10 classes is 41.5%, while AL achieves 83.8% accuracy. The accuracy forgetting is 58.5% for the conventional scheme and 8.8% for AL, reducing the accuracy forgetting by 6.6X. For CIFAR-100, conventional scheme forgets 81.9% accuracy after learning 100 classes, while acquisitive learning forgets only 7.1% accuracy, reducing the accuracy forgetting by 11.5X.



(a) Incrementally Learning 1 Class of CIFAR-10 with VGG-16.

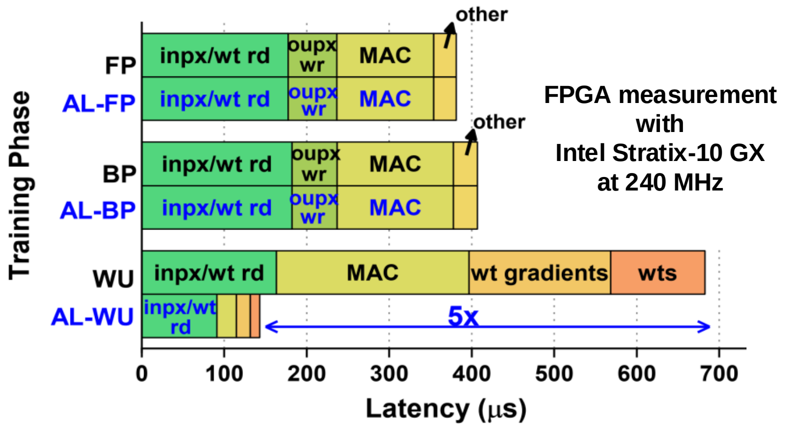


(b) Incrementally Learning 10 Classes of CIFAR-100 with ResNet-20.

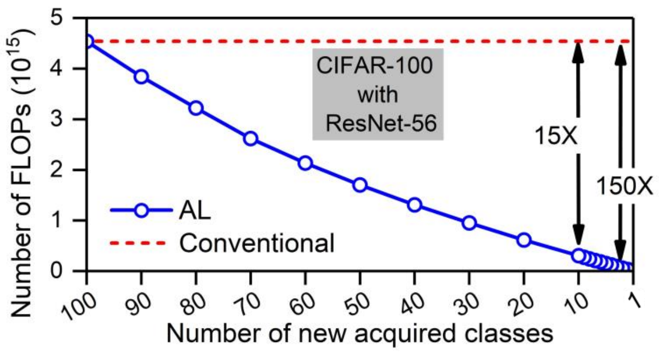
Figure 5.8: The Comparison of Overall Accuracy Between Conventional Continual Learning and the Proposed Acquisitive Learning When (a) Incrementally Learning 1 Class in a Sequence on CIFAR-10 (B) Incrementally Learning 10 Classes in a Sequence on CIFAR-100. In the Figure, 'I9' Means That AL Starts Training from a Model That Is Pre-trained on 9 Classes. Similarly, 'I50' Means the Inherited Model Is Pre-trained on 50 Classes.

5.4.5 Computation Cost and FPGA Prototyping

We benchmark the computation cost, including latency, energy efficiency and floating-point operations (FLOPs), of both conventional scheme and proposed AL.



(a) Latency per training image



(b) Number of FLOPs needed for acquisition of various number of new classes, derived from the FPGA result

Figure 5.9: Comparison on Computation Cost Between Conventional Continual Learning and Proposed Acquisitive Learning.

In FPGA simulation, the computation flow of the forward and backward pass of the proposed learning algorithm remains the same as the conventional training. However, during the weight update phase, we need to compute the selected weight gradients by convolving input activations with the activation gradients and update corresponding weights. The proposed learning approach was evaluated based on a FPGA training accelerator [106]. Details of the selected weights in each layer that have to be updated were given as an input to the accelerator. With the segmented training in AL, the control logic in FPGA completely skips the DRAM access of the frozen weights thereby reducing the off-chip communication and latency during the weight gradient computation. The weight gradient computation is performed only for the unfrozen weights. During the entire weight update phase, the frozen weights in DRAM remain untouched.

Figure 5.9(a) shows the latency breakdown of ResNet-20 for Forward Pass (FP), Backward Pass (BP) and Weight Update (WU) of training, from FPGA measurement. The bar graph highlighted with blue colored text shows the latency of the AL scheme to acquire one class of CIFAR-10. Using AL, we achieve 5X reduction in latency for WU phase per training image by only updating the selected weights ('AL-WU' in Figure 5.9(a)), compared to the conventional scheme.

Figure 5.9(b) shows the number of training FLOPs. As AL only needs to acquire a few classes with the main model segmented, the training FLOPs is largely reduced as compared to conventional training. Learning 1 class ('99+1' scheme) and 10 classes ('90+10' scheme) from CIFAR-100 with AL reduces FLOPs by 15X and 150X, respectively. Based on FPGA values, Table 5.2 further derives the simulated throughput (TFLOPs/s) required for training different numbers of new acquired classes, on CIFAR-10, CIFAR-100 and ImageNet [21] with ResNet-56. We assume a typical hardware platform (such as FPGA and GPU) that manages the input image stream at 30 frames/second [108], exhibits power budget of 100W [109] and energy efficiency of 20 GFLOPs/second/Watt per platform. As AL effectively reduces computation cost, such a platform is able to support the acquisition of as many as 50 classes with one platform for CIFAR-100, or 500 classes with 10 platforms for ImageNet.

5.4.6 FPGA Demonstration

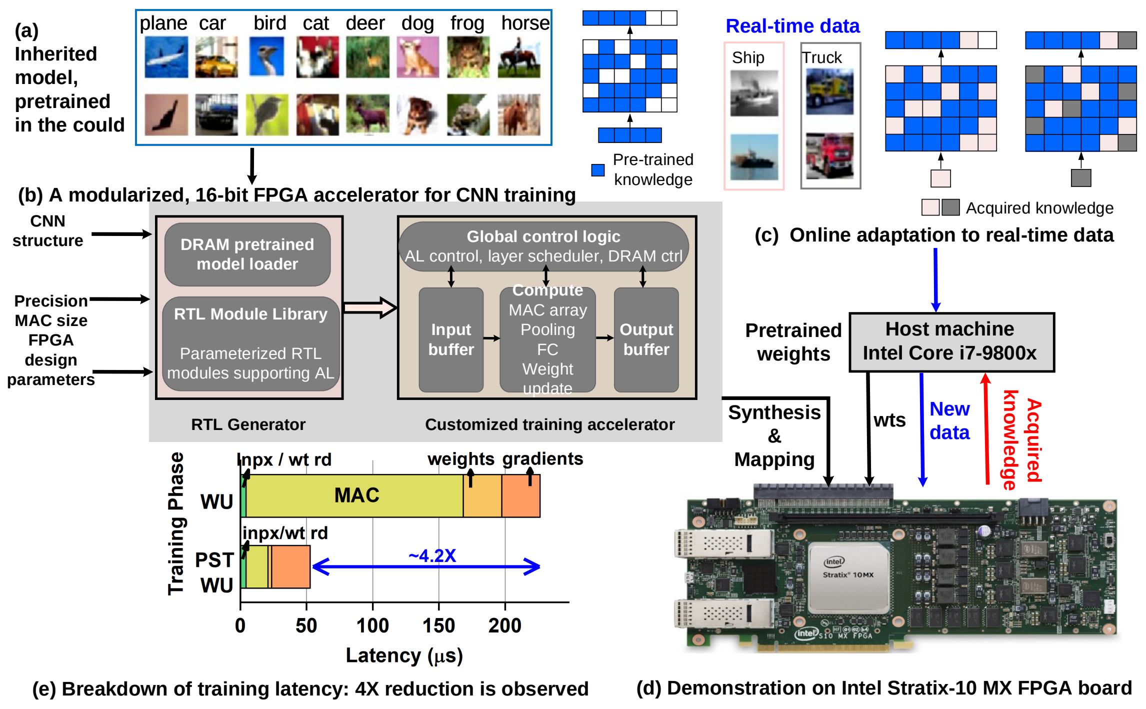
In this section, we demonstrate online CIFAR-10 CNN learning on an FPGA-based 16-bit fixed-point training accelerator [110, 111] on Intel Stratix-10 MX FPGA [112].

Table 5.2: Required Throughput (TFLOPs/Second) and the Number of Hardware Platforms* Needed to Learn Various Number of Classes with AL.

Number of Classes	1000	500	100	50	10	5	1
CIFAR-10	-	-	-	-	🟦 2.7	🟩 2.0	🟩 1.8
CIFAR-100	-	-	🟦 2.7	🟩 2.0	🟩 1.8	🟩 1.8	🟩 1.8
ImageNet	🟨 22.0	▨ 16.6	▨ 14.8	▨ 14.7	▨ 14.7	▨ 14.7	▨ 14.7

🟨 100 platforms ▨ 10 platforms 🟦 2 platforms 🟩 1 platform

*We assume that one hardware platform provides 20 GFLOPs/s/Watt with 100W.



(a) Inherited model, pretrained in the could
plane car bird cat deer dog frog horse
Pre-trained knowledge
Real-time data
Ship
Truck
Acquired knowledge
(c) Online adaptation to real-time data
(b) A modularized, 16-bit FPGA accelerator for CNN training
CNN structure
Precision MAC size FPGA design parameters
DRAM pretrained model loader
RTL Module Library
Parameterized RTL modules supporting AL
RTL Generator
Global control logic
AL control, layer scheduler, DRAM ctrl
Input buffer
Compute
MAC array
Pooling
FC
Weight update
Output buffer
Customized training accelerator
Pretrained weights
Host machine
Intel Core i7-9800x
Synthesis & Mapping
wts
New data
Acquired knowledge
Training Phase
WU
PST WU
Inpx / wt rd
weights
gradients
MAC
inpx/wt rd
~4.2X
0
50
100
150
200
Latency (μs)
(e) Breakdown of training latency: 4X reduction is observed
(d) Demonstration on Intel Stratix-10 MX FPGA board

Figure 5.10: FPGA Demonstration of PST Algorithm.

The CNN training hardware is flexible to support forward pass (FP), backward pass (BP) and weight update (WU) phases of training.

Figure 5.10 presents the overall FPGA system setup [5] to train CNNs using PST algorithm. For simplicity, the CNN structure used here is 16C3-16C3-MP-32C3-32C3-MP-64C3-64C3-MP-FC, where 'NCk' refers to convolution layer with 'N' output feature maps and kernel size of ‘k’, ‘MP’ refers to max pooling layer and ‘FC’ refers to a fully-connected layer. First, as shown in Figure 5.10a, a large amount of weights is pre-trained and selected with 9 classes from CIFAR-10 dataset. The pre-trained model and a binary mask representing the frozen weights are fed to the RTL generator. The RTL generator generates the customized training accelerator based on the pre-trained model structure and generates HBM2 memory initialization files to load the model parameters, as shown in Figure 5.10b. The frozen weights stored in HBM2 are used by the FPGA training accelerator to perform inference on pre-trained classes.

The model is then exposed to a new, unlearned class from CIFAR-10, and updated accordingly in real-time on the FPGA, as shown in Figure 5.10c. The entire system is demonstrated on Intel Stratix-10 MX FPGA board (Figure 5.10d). Benefiting from the model segmentation, the online training of new observations requires much less computation cost and lower latency, as compared to traditional continual learning scheme that updates the entire network. As shown in Figure 5.10e, the breakdown graph shows that the PST saves 4.2× latency per image in the weight update (WU) phase as compared to traditional algorithms.

5.5 Conclusion

In this paper, we propose a new perspective to mitigate catastrophic forgetting in continual learning: acquisitive learning. Different from previous continual learning that learns from scratch and focuses only on model adaptation, acquisitive learning (AL) addresses both knowledge inheritance and acquisition, inspired by the Moravec’s paradox. With AL, the accuracy drop in learning sequential tasks is reduced by 6.6X and 11.5X for CIFAR-10 and CIFAR-100 datasets, respectively, as compared to the conventional scheme. Meanwhile, we confirm that the amount of inherited knowledge and the quality of inherited model are important to the capacity of knowledge acquisition. Furthermore, benefiting from segmented training, the weight update latency is reduced by 5X as benchmarked by FPGA prototype, training FLOPs is reduced by 150X, enabling knowledge acquisition at the edge. In the future, we plan to investigate more criteria to select the inherited model, and techniques to automatically generate better models for more accurate and stable acquisition. We will also develop more flexible and efficient hardware techniques for the implementation of AL.

Chapter 6

EVOLUTIONARY NAS IN LIGHT OF MODEL STABILITY FOR ACCURATE CONTINUAL LEARNING

6.1 Introduction

As the cornerstone of deep learning, the advance of deep neural networks (DNN) has enabled great success in diverse real-world applications, such as image classification [20], speech recognition [28], object detection [113] and natural language processing [29]. Recent years, along with the rapid development of computation hardware, there is an emerging category of edge devices, including but not limited to autonomous vehicles and drones. These edge devices are equipped with much higher computation power (though still lower than the cloud center) than traditional edge devices (such as mobile phones). These emerging devices are required to deal with much more complicated and dynamic situations. In some scenarios, these intelligent systems have to quickly pick up a new task, learn it online in a continuous manner, and react immediately, rather than sending the data back to the cloud and waiting for the cloud to handle the training. Thus, the capability of continual learning is a necessary attribute for such an intelligent edge system.

In conventional continual learning algorithms, researchers usually split a full dataset into several sub-dataset as different tasks, and then incrementally train tasks one by one, starting from a randomly initialized, handcrafted model [67, 82, 114]. However, considering a real-life situation, continual learning may not be used to train a model from scratch at the edge. Instead, the model is usually pre-trained offline with large amounts of observations (*i.e.* cloud data) and deployed on edge devices. Once deployed, the edge device only needs to continuously learn a few new observations based on the pre-trained model. Such a pre-trained model is referred to as *inherited model*, and the pre-trained data is referred to as *inherited knowledge* in [4]. [4] further claims that different inherited models have different capabilities in learning new data (knowledge acquisition) while maintaining the accuracy of old knowledge. In the context of continual learning, most of the models quickly fall into the pain-point of *catastrophic forgetting* [67] while a few models are able to keep decent accuracy after learning a sequence of new tasks. Such capability of remembering old knowledge while acquiring new knowledge is referred to as the inherited model's *stability* [115].

However, it is non-trivial to construct an stable and optimal inherited model once for all due to the following reasons: (1) the model architectures are usually problem-dependent, if the edge application and distribution of the data are changed, the architectures must be redesigned accordingly; (2) the existing DNN architectures are all handcrafted by experts, but in practice, most end users have limited expertise in architecture design. Thus, neural architecture search (NAS), which automates the architecture designing according to diverse user needs, is able to effectively tackle the difficulties mentioned above.

As indicated by [116], NAS leverages a performance estimation strategy during the search to evaluate the architecture candidates. Most of the existing NAS algorithms use the classification accuracy (or error rate) on the validation dataset as the performance metrics for image classification tasks. However, accuracy itself only represents the model performance on the current task but overlooks the model's performance on future observations. Therefore, in this work, we propose to encompass model stability in the evaluation metrics during NAS so that the searched architecture not only learns well on the current task but also learns well on the incoming data stream without forgetting the previous knowledge. Such a pipeline is sketched in Figure. 6.1.

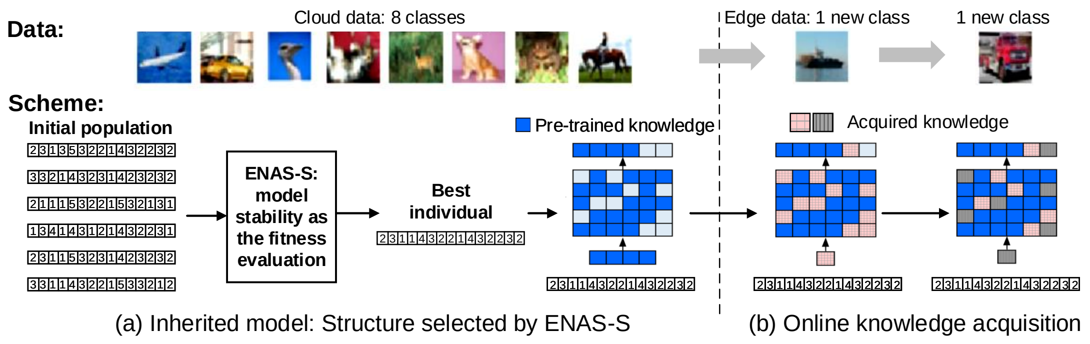


Figure 6.1: ENAS-S Leverages Model Stability to Search for an Optimal Architecture Through the Evolutionary Algorithm. Such an Architecture Is Pre-trained, Deployed at the Edge, and Then Exposed to a Few New Observations. The Edge Device Is Required to Acquire New Knowledge as Well as Remembering the Inherited Knowledge Successfully.

On the other side, the *Moravec's paradox* [100–102] claims that brains inherit knowledge in specific neurophysiological structures through a long and careful evolution process. In order to mimic such an evolution process, we use the evolutionary algorithm (EA) as the search strategy. Meanwhile, we use single-path backbone and block-based search space, which are budget-friendly in edge computing. Extensive experiments on popular benchmarks, including CIFAR-10 and CIFAR-100, validate that the proposed ENAS-S successfully finds architectures that outperform manually designed architectures when learning from a data stream at the edge.

To our best knowledge, this is the first NAS work leveraging model stability to address the continual learning scenario. The contribution of this paper is three-folded:

(1) We propose a neural architecture search algorithm, namely ENAS-S, which leverages model stability as the performance evaluation metrics during searching. ENAS-S aims to discover the optimal inherited architecture for accurate continual learning at the edge.

(2) To relieve computational budget of edge devices, ENAS-S uses the evolutionary algorithm as the search strategy, a single-path backbone and a block-based library as the search space.

(3) On CIFAR-10 and CIFAR-100, we demonstrate that ENAS-S finds architectures that outperform handcrafted models in continual learning with much competitive accuracy and much smaller model size. We further provide a comprehensive analysis of the evolutionary process, including model performance, loss landscape and architecture components.

6.2 Related work

As summarized by [116], NAS has three main components: search space, search strategy, and performance estimation strategy. In this section, we introduce the previous work from these perspectives.

Search space Usually, there are two categories of search space: cell-based and block-based. [117–121] use basic cells, such as 3×3 and 5×5 separable convolutions, as the search space. These methods usually use graph-based architecture as backbone and treat each cell as a node of the graph. On the other hand, [122–126] use basic blocks from mature DNNs as the search space and these blocks connect end to end, forming a single-path architecture. Single-path architecture is considered as a more efficient solution for edge devices to train and inference with limited computation budget (including memory and power).

Search strategy There are three mainstream algorithms for NAS: gradient-based, reinforcement learning, and evolutionary algorithm. [127–131] transform the search space to be continuous and optimize the searching by gradient descent. However, gradient-based searching usually requires a graph-based backbone, which is challenging and less efficient for edge devices. [126, 132–135] use reinforcement learning as the search strategy and use a performance evaluation metrics (which is usually the accuracy on validation dataset) as the reward signal to train the controller. However, reinforcement learning is computation intense, making it hard to be applied to edge devices. [122, 136, 136–139] leverage various evolutionary algorithms to search for the optimal architecture. However, they all use accuracy as the fitness score. Different from them, ENAS-S leverages both model stability and accuracy as the fitness score.

Performance estimation strategy For the above-mentioned previous work, most of them use validation accuracy as the fitness score, and the rest consider hardware cost simultaneously [124, 130]. These efforts aim to find an architecture that achieves the best accuracy on the full testing dataset. In our paper, ENAS-S aims to find an architecture that not only performs well on the inherited knowledge (cloud data) but is also able to acquire new knowledge and minimize the catastrophic forgetting. To our best knowledge, this paper is the first work encompassing model stability in the fitness evaluation and addressing searching for continual learning scenarios.

6.3 Methodology

This section describes the detailed implementation of three components in NAS, *i.e.*, search space, search strategy, and performance evaluation strategy.

6.3.1 Search Space

Considering the limited computation budget at the edge, we aim to search for hardware-friendly architectures so that the training of the inherited model at the edge is feasible. As discussed in Section 6.2, a smaller search space and a single-path

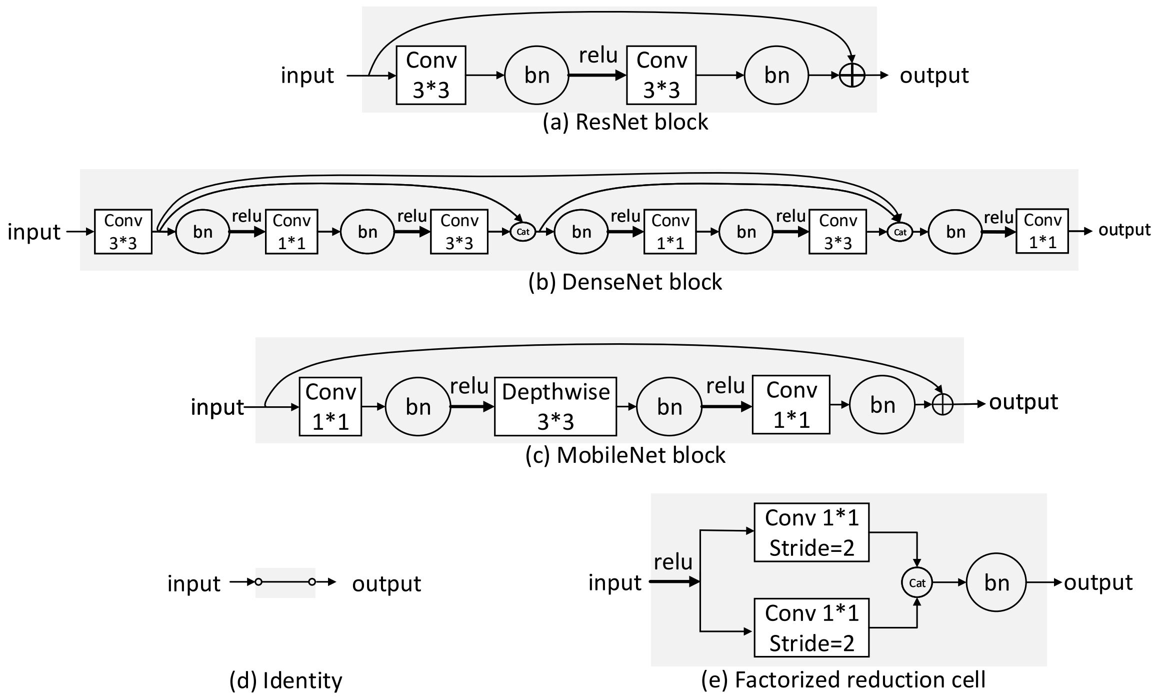


Figure 6.2: Structure of Each Block.

architecture are preferred. Thus, ENAS-S search space consists of (1) a library of mature DNN blocks, (2) a single-path module-based backbone formed by blocks, and (3) a variable-depth encoding strategy that maps each architecture to a gene sequence which is used in the evolutionary algorithm.

Block library Block is the most basic unit in ENAS-S. There are two categories of blocks in the block library: (1) regular blocks which have the stride of 1 and thus maintaining the output feature map dimension (2) reduction blocks which have a stride of 2 and thus down-sampling the size of the output feature map. We cover 4 regular blocks and 3 reduction blocks in the library, as shown in Figure 6.2 and Table 6.1. These blocks are all mature DNN basic units and widely used in previous NAS work [117, 125, 140, 141]. It worth mentioning that we add an *identity* block, which is a dummy block that directly connects input with output. With this block,

Encoding	Type	Name
0	Regular	ResNet block (RB)
1		DenseNet block (DB)
2		MobileNet block (MB)
3		Identity (ID)
4	Reduction	Factorized reduction
5		Average pooling
6		Max pooling

Table 6.1: Block Library in ENAS-S.

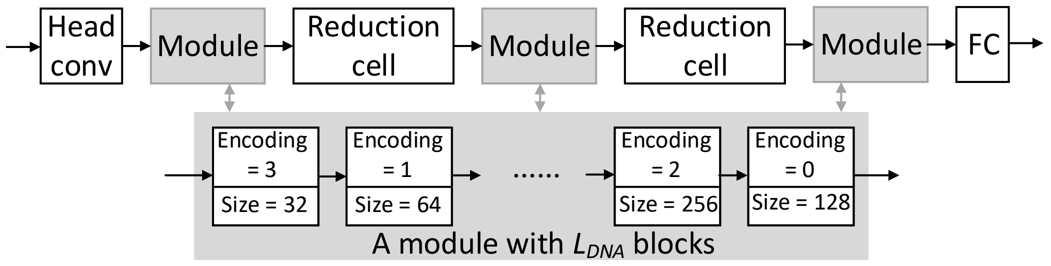


Figure 6.3: ENAS-S Uses a Module-based, Single-path Backbone Which Makes the Generated Architectures More Efficient to Be Deployed and Trained at the Edge.

ENAS-S is able to generate variable-depth architectures with fixed-length encoding sequences.

Backbone ENAS-S randomly samples blocks from the block library and links them together following the backbone structure shown in Figure 6.3. Given an input image, it is first forwarded to a head convolutional cell, which is a 3×3 convolutional layer. It is then forwards into three repeated modules with two reduction cells in between. Each module is a sequence of L_{DNA} regular blocks, and each block is randomly assigned with a size (architecture width). Finally, it is forwards to a fully-connected layer with N_o outputs, where N_o equals the number of outputs. Assuming there are N_{regu} choices of regular blocks and N_{redu} choices of reduction cells in the block library, such a backbone reduces the search space to $N_{regu}^{L_{DNA}} \times N_{redu}^2$ combinations. In our experiments, $N_{regu} = 4, N_{redu} = 3, L_{DNA} = 5$, which means there are approximately 10^3 architectures, which is much smaller than the previously 10^{15} [142]. Such a backbone is also validated by [87].

Gene encoding strategy Gene encoding strategy is a vital step in evolutionary algorithm. Each architecture is represented by a sequence of numbers, where each number represents a block. For example, a gene sequence '3-1-0-2-0' represents the architecture of 'ID-DB-RB-MB-RB.' Benefiting from the *identity* block, though the gene length is 5, the architecture is indeed 4-layer, achieving variable-depth encoding.

6.3.2 Fitness Evaluation

Fitness evaluation, *i.e.* $\mathcal{L}(\cdot)$ in Equation 6.3, determines the direction and destination of the NAS. Most of the previous NAS algorithms use *accuracy* on D_{valid} as the fitness evaluation metrics and thus, find the architecture with the best accuracy on the current task. However, in the continual learning scenario, we not only care about accuracy on the current task but also care about whether the accuracy can maintain when the model is exposed to a sequence of new tasks. In other words, it is critical that the model is stable so that the *catastrophic forgetting* is mitigated when the learning system is acquiring new knowledge from a data stream. Thus, we need a fitness evaluation metrics that reflects both the model accuracy and the model stability.

Previously, [4] uses one-shot noise injection to monitor the stability of the inherited model. For each layer l in a neural network, the applied noise is formulated as below:

$$\tilde{\Theta}_l = \Theta_l + \alpha \cdot n_l, \tag{6.1}$$

where Θ_l is the noise-free weight matrix in the l-th layer with the standard deviation σ_l. $\alpha \in (0.0, 1.0)$ is a constant scaling coefficient, and n_l is the noise matrix of the l-th layer that follows normal distribution $n_l \sim \mathcal{N}(0, \sigma_l^2)$. After such noise is added on the clean weights, an accuracy drop $\Delta Acc = Acc_{pre} - Acc_{post}$ is observed, where $\Delta Acc. \in (0.0, 1.0]$. The higher accuracy drop, the weaker model stability, and vice versa. Therefore, the fitness score of ENAS-S is defined as:

$$\mathcal{F}itness = \frac{1}{\Delta Acc} \times Acc_{pre}. \tag{6.2}$$

6.3.3 Search Strategy

We prefer a computation-friendly NAS algorithm in case the searching needs to be done at the edge. Mathematically, the NAS is modeled by an optimization problem formulated as:

$$\arg\min_{A} = \mathcal{L}\left(A, \mathcal{D}_{\text{train}}, \mathcal{D}_{\text{valid}}\right), \text{s.t. } A \in \mathcal{A}, \tag{6.3}$$

where $\mathcal{L}(\cdot)$ represents the fitness score on $\mathcal{D}_{\text{valid}}$ after training on $\mathcal{D}_{\text{train}}$, A is the architecture and $\mathcal{A}$ is the the search space. In this work, we use the simple yet effective evolutionary algorithm, to be specific, the genetic algorithm [143], to approach the optimization of $\mathcal{L}(\cdot)$. There are two main reasons to use evolutionary algorithm: (1) as mentioned in Section 6.1, the Moravec's paradox claims that the intelligence in nature may be established more by the long-term selection rather than the short-term adaptation. Therefore, we use the evolutionary algorithm to mimic such a long-term selection; (2) evolutionary algorithm is a classic and straightforward NAS search strategy, so it is easier to validate the novel fitness score that we propose. It

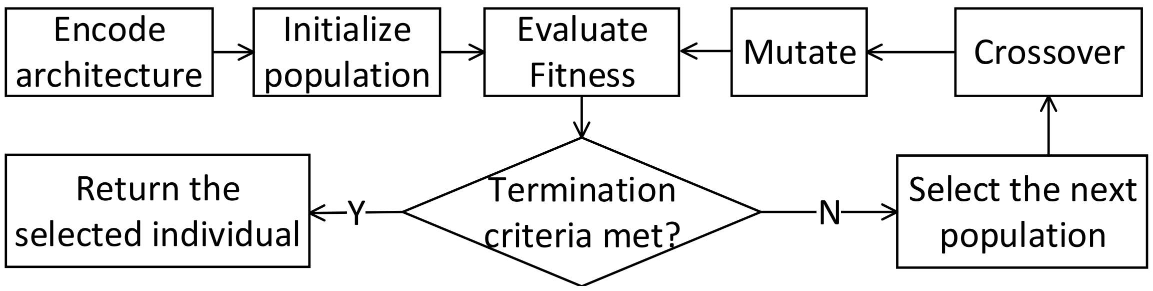

Encode architecture
Initialize population
Evaluate Fitness
Mutate
Crossover
Return the selected individual
Y
Termination criteria met?
N
Select the next population

Figure 6.4: Flowchart of the Evolutionary Algorithm.

is worth mentioning that, from implementation point of view, the proposed fitness score is general and can also be applied to other NAS algorithms.

Figure 6.4 shows the flowchart of evolutionary algorithm. First, an initial population of N_{idv} individuals is generated, where each individual is a randomly sampled gene sequence following the backbone and the encoding strategy described in Section 6.3.1. Secondly, each individual undergoes the fitness evaluation on $\mathcal{D}_{\text{valid}}$. Assuming inherited knowledge contains much more classes than new observations, the $\mathcal{D}_{\text{valid}}$ used here is partial dataset containing large amounts of classes, as annotated in Figure 6.1. We also leverage early stopping strategy [139, 144] to speed up the searching: each individual undergoes a fixed, small number of training epochs and non-converging individuals are terminated to save time. Through this fitness evaluation, the fitness score described in Section 6.3.2 is collected for each individual. The fitness score is proportional to the probability of whether this individual can retain in the population for the next generation. With the fitness score, promising individuals are selected from the current population and regarded as the *parent* individuals. If EA's termination criteria are not yet met, these parent individuals will generate offspring individuals through crossover and mutation operators. Then the offspring individuals replace the parent individuals, forming the next generation of the population. Such a recurrent loop continues until the termination criteria is met.

Algorithm 5 ENAS-S

Input: $\mathcal{D}_{\text{train}}$, $\mathcal{D}_{\text{valid}}$, L_{DNA}, N_{idv}, N_{gen}, μ, α, ν.

```
 1: $\mathcal{P}_0 \leftarrow$ Initialize a population with $N_{idv}$.
 2: for Generation $g$ from $g = 0$ to $N_{gen}$ do
 3:     Evaluate each individual in $\mathcal{P}_g$ with the proposed fitness score in Eq. 6.2 on
        $\mathcal{D}_{\text{valid}}$ after training $E$ epochs on $\mathcal{D}_{\text{train}}$
 4:     Randomly generate a population $\mathcal{P}_{sample}$ from $\mathcal{P}_g$ with the probabilities associ-
        ated with each entry in $\mathcal{P}_g$.
 5:     for each individual $p_1$ in $\mathcal{P}_{sample}$ do
 6:         $r \leftarrow$ Uniformly generate a number from [0,1]
 7:         if $r < \mu$ then
                Crossover. See Algorithm 6.
 8:         end if
 9:         $r \leftarrow$ Uniformly generate a number from [0,1]
10:         if $r < \nu$ then
                Mutate. See Algorithm 7.
11:         end if
12:     end for
13:     $\mathcal{P}_g \leftarrow \mathcal{P}_{sample}$ Update the population
14: end for
15: Select the individual with the highest fitness score and decode its gene sequence
    to DNN architecture $A$
```

Output: the optimal architecture A

```
Algorithm 6 Crossover.
Input: Population $\mathcal{P}_{sample}$. Individual $p_1$.
1: Randomly select another individual $p_2$ from $\mathcal{P}_{sample}$
2: Randomly select positions from $p_1$ and $p_2$
3: $p_{offspring} \leftarrow$ Combine the selected part of $p_1$ and the selected part of $p_2$
4: $p_1 \leftarrow p_{offspring}$ Replace $p_1$ with $p_{offspring}$ and put back in $\mathcal{P}_{sample}$
Output: $\mathcal{P}_{sample}$.
```

```
Algorithm 7 Mutate.
Input: Population $\mathcal{P}_{sample}$. Individual $p_1$.
1: Randomly select a position from $p_1$
2: Replace the selected position with another random encoding of a block with the
   same type (regular or reduction) and put back in $\mathcal{P}_{sample}$
Output: $\mathcal{P}_{sample}$.
```

The termination criteria used in ENAS-S is the number of generations. The detailed algorithm is illustrated in Algorithm 5.

6.3.4 Continual Learning at the Edge

Problem definition A deep neural network (DNN) usually consists of a feature extractor $\varphi : \mathcal{X} \rightarrow \mathbb{R}^d$ and classification weight vectors $w \in \mathbb{R}^d$, also known as convolutional layers and fully-connected layers. The network parameters Θ (φ and w) keep being updated according to input data $\mathcal{X}$, and calculating output $\mathcal{Y} = w^\top \varphi(\mathcal{X})$ in order to predict labels $\mathcal{Y}^*$. In the continual learning scenario, when learning the inherited knowledge with input data $\{X^1, \ldots, X^{s-1}\}$, DNN tries to minimize the loss $\mathcal{L}(\mathcal{Y}; \mathcal{X}_{s-1}; \Theta)$ of this $(s-1)$-class classifier. When a new edge task with input data $\{X^s, \ldots, X^t\}$ arrives, DNN tries to minimize $\mathcal{L}(\mathcal{Y}; \mathcal{X}_t; \Theta)$ of this t-class classifier by updating Θ. Usually, after the input data of the new task $\{X^s, \ldots, X^t\}$ arrives, the input data of previous task $\{X^1, \ldots, X^{s-1}\}$ is no longer available, except a small subset stored as the memory set $\mathcal{P} = (P_1, \ldots, P_{s-1})$.

Continual learning techniques After ENAS-S finds an optimal inherited architecture, we need to validate its performance in a continual learning scenario. In this paper, we focus more on this validation rather than the continual learning algorithm itself. Thus, the techniques used to perform continual learning are very flexible. Here, we use simplified progressive segmented training ([3]) as the continual learning algorithm. Following [3, 4], we leverage the importance sampling and memory-assisted balancing steps to learn new observations based on an inherited model. After ENAS-S found the optimal architecture, we pre-train this architecture with the cloud data, *i.e.* a large amount of classes from the full dataset. For example, 8 classes from CIFAR-10 are treated as cloud data (inherited knowledge), and the rest two classes are fed sequentially as edge data (new tasks), annotated as '8+1+1' scheme. Then we use importance sampling to select and freeze the top β units in the inherited model. In the online learning phase, these important units are not updated but kept unchanged in order to preserve inherited knowledge. When a new edge task arrives, only the secondary parameters in the inherited model are updated. Meanwhile, a small set of uniformly and randomly sampled images from all the trained classes so far (*i.e.* each class in $\{X^1, \ldots, X^t\}$ contains the same number of images) are mixed with new observations to train and balance the model.

6.4 Experimental Results

6.4.1 Experiments Setup

We perform experiments in this paper with PyTorch [145] on a single NVIDIA GeForce RTX 1080 platform. We use stochastic gradient descent with a momentum of 0.9 and a weight decay of 0.0005.

Hyper-parameter setting In the experiments, we set the ENAS-S hyper-parameters as follows: 60 individual are in the population and we search for 6 generations; each individual has gene length of $L_{DNA} = 5$; crossover rate $\mu = 0.8$ and mutation rate $\nu = 0.05$. Noise coefficient $\alpha = 0.8$. We following the same setting in [2] for β: it should be roughly proportional to the amount of the inherited knowledge. For example, $\beta = 0.8$ for the '8+1+1' scheme mentioned below.

Datasets and network The CIFAR-10/CIFAR-100 datasets [25] include 50,000 training images and 10,000 testing images in color with size 32×32. There are ten classes for CIFAR-10 and 100 classes for CIFAR-100. In this paper, we aim at searching for an architecture for the continual learning scenario, so we divide the full dataset into cloud data and edge data, following [4]. The balanced memory set contains 200 and 20 images for each class for CIFAR-10 and CIFAR-100, respectively, so that the total memory size is bounded within 2,000 images for both datasets, aligning with previous works [2, 114]. The network structures of ResNet56-NS [24], DenseNet [107] used as control group follow [104].

Evaluation protocol We use the single-head evaluation, which is considered more realistic and challenging than multi-head evaluation [87]. 'Single-head overall accuracy' refers to the t-class classifier's testing accuracy on all the data seen so far.

‘Overall accuracy’ represents the model’s capability of preserving inherited knowledge and achieving knowledge acquisition, thus reflecting the model stability in continual learning.

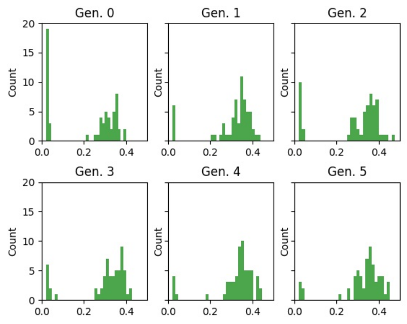
6.4.2 Effectiveness of ENAS-S

First of all, we validated the effectiveness of ENAS-S on evolving and searching architectures. In Figure 6.5(a) and 6.5(b), we present the distribution of individual’s Acc and ΔAcc over generations, respectively. In Figure 6.5(c), we plot the mean and standard deviation of Acc and ΔAcc. From these figures, we can see that along with evolution, the accuracy of the population is increasing, while the ΔAcc of the population is decreasing, indicating the improvement of accuracy and model stability. Furthermore, the standard deviation of both Acc and ΔAcc is decreasing over generations, meaning that the searching converges from disorder to order.

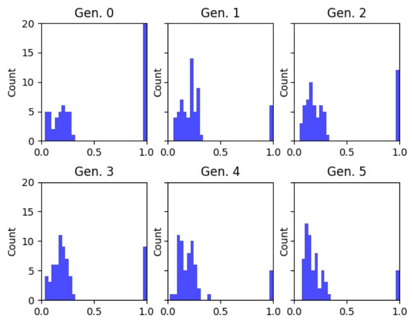
Moreover, we visualize the loss landscape of the best individual in each generation using the tool provided by [104], and plot in Figure 6.6. [104] and [3] point out that models with better stability have more flat and smoother loss landscape. 6.6 validates the improvement of model stability over generations.

6.4.3 Effectiveness of the Proposed Fitness Score

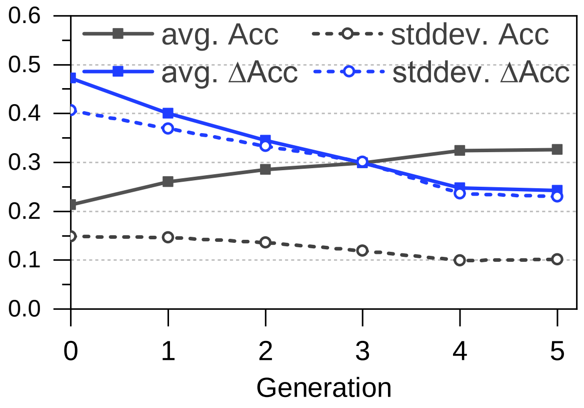
Previously, [4] proves that hand-designed DenseNet121 is a relatively stable model and performs well in the continual learning scenario; on the other side, ResNet56 without shortcuts (ResNet56-NS) has relatively weak model stability and lower accuracy in continual learning tasks. Thus, in this work, we use DenseNet121 and ResNet56-NS as the control group. Assuming the inherited knowledge contains much more classes than new observations, we use a subset of classes as cloud data to search for the architecture. Next, we treat the *unseen* classes as new edge tasks and use the architecture



(a) Distribution of Individuals' Acc over Generations.



(b) Distribution of Individuals' ΔAcc over Generations.



(c) Mean and Standard Deviation of Acc and ΔAcc over Generations.

Figure 6.5: Visualization of the Evolution Process over Generations. It Is Observed That along with the Evolution, the Acc of the Population Is Getting Higher and ΔAcc of the Population Is Getting Lower. Meanwhile, Their Standard Deviation Is Reducing Because the Searching Is Converging over Time.

found to acquire these new knowledge. For various amount of inherited knowledge, we use ENAS-S to search the optimal architectures, once the searching ends, we use this architecture to learn a sequence of new tasks and plot their single-head overall accuracy in Figure 6.7. For example, in Figure. 6.7(a)(i), ENAS-S uses 5 classes as

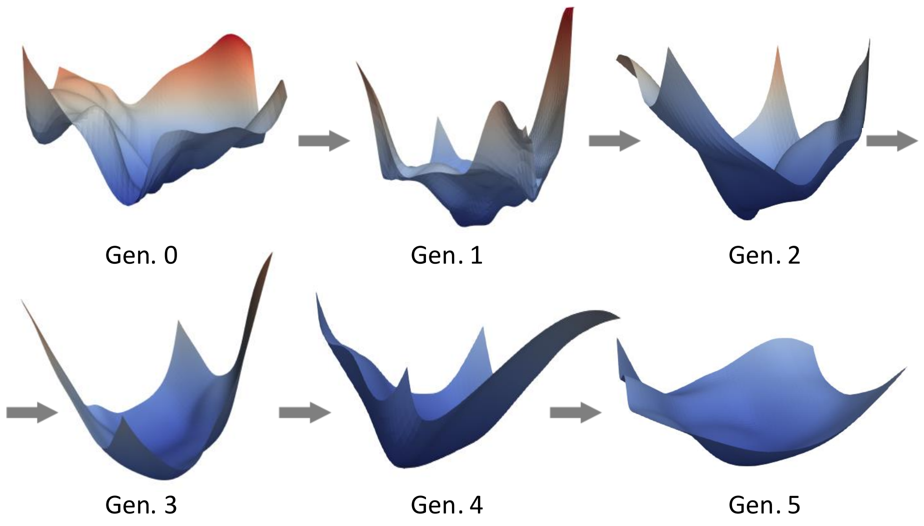
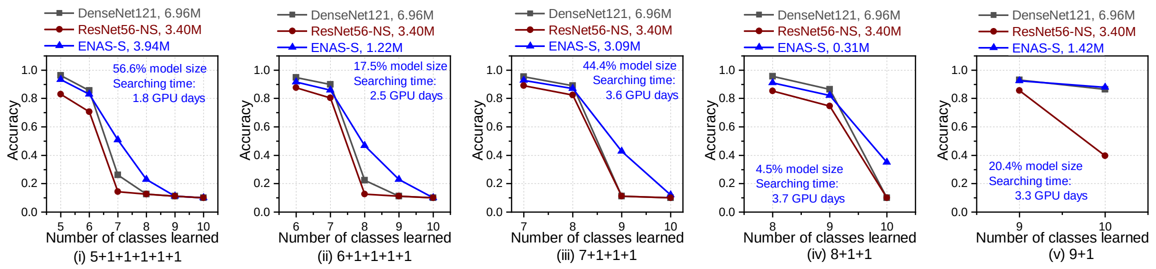
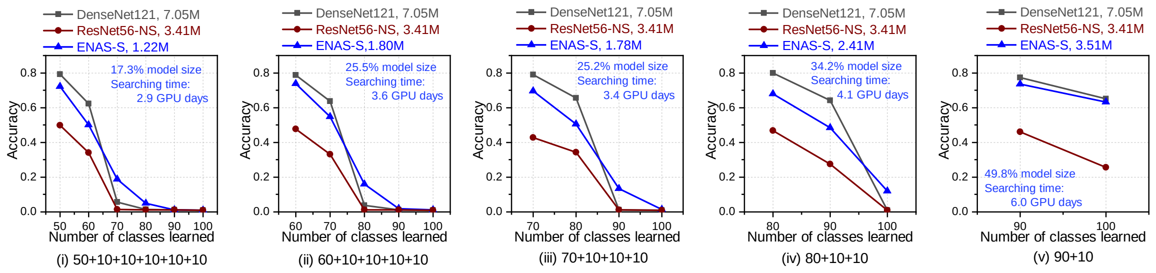


Figure 6.6: Loss Landscape of the Model Evolves from Roughness to Smoothness over Generations, Indicating the Improvement in Model Stability.

the cloud data to search for the inherited models, and then use the inherited model to learn a sequence of five 1 class online with the PST algorithm. For the cloud data, *i.e.* 5 classes, ENAS-S has a lower accuracy than DenseNet121, which is as expected since ENAS-S does not only address the accuracy in its fitness score. For the first, second, and the third edge class, ENAS-S shows less accuracy drop and flatter curve slope, indicating less catastrophic forgetting is happening. Note that all the models are suffering from extreme catastrophic forgetting when learning the last one or two edge classes. This is because the catastrophic forgetting problem in continual learning remains as unsolved challenge. Furthermore, the ENAS-S architectures are much smaller in model size, for example, in Figure. 6.7(a)(iv), ENAS-S model outperforms DenseNet121 with only 4.5% model size as compared to DenseNet121. Thus, ENAS-S architectures are more hardware-friendly than handcrafted models because edge computing favors smaller model size due to the limited computation and storage resource.



(a) Using the architecture discovered by ENAS-S to learn a a sequence of new tasks on CIFAR-10.



(b) Using the architecture discovered by ENAS-S to learn a sequence of new tasks on CIFAR-100.

Figure 6.7: Comparison among ENAS-S Architectures and Handcrafted Models in the Scenario of *Continual Learning at the Edge*. ENAS-S Architectures Outperform Hand-designed Models in Mitigating Catastrophic Forgetting with Much Small Model Size (as Noted in Figure).

Similarly conclusions are observed for CIFAR-100. In Figure. 6.7(b)(v), ENAS-S architecture achieve 73.5% accuracy on cloud data with 3.51M parameters, while DenseNet121 achieve 77.2% accuracy on the cloud data with 7.05M parameters. When these two models learn in edge data, 12.2% accuracy drop is observed for DenseNet121, but such a drop is only 10.3% for ENAS-S with half of the model size.

Furthermore, benefiting from early stopping policy, ENAS-S spends up to 6 GPU days in searching, which is much faster than the state-of-the-art NAS which takes >25 GPU days [122, 146].

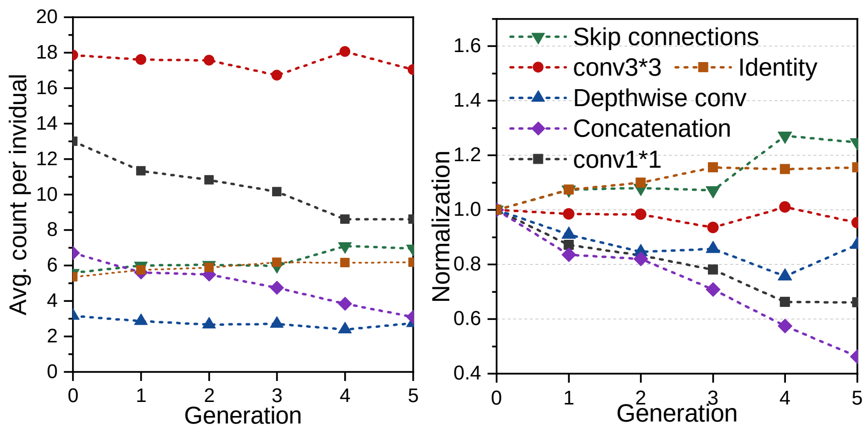


Figure 6.8: Left: Average Number of Each Component in an Individual; Right: The Normalized Trend over Evolutionary Generations.

6.4.4 Architecture Evolution

Each block in the ENAS-S block library consists of several kinds of architecture components. For example, each ResNet block contributes two 3×3 convolution cells (conv3×3), and one skip connection (a.k.a shortcut); each DenseNet block contains three 3×3 convolution cells, three 1×1 convolution cells (conv1×1) and three concatenation operations. For each generation, we calculate and plot the average amount of each component in one individual in Figure 6.8 (left). Meanwhile, we plot the normalized count of each component with respect to the first generation in Figure 6.8 (right). It is observed that the number of skip-connections and the number of identity are significantly increasing over generations. We can infer that skip-connection improves the model stability, and a shallower architecture (*i.e.* less number of layers) has better model stability. This is because shortcuts mitigate the explosion of non-convexity that occurs when networks grow deeper. Such a conclusion is also validated by [104]. It is also observed that the number of depthwise convolutions and conv1×1 cells are significantly decreasing. This could because the depthwise convolutions (existing in

MobileNet block) and conv1×1 cell (existing in DenseNet block, MobileNet block, and factorized reduction cell) are not favored by model stability though they reduce computational cost. As for the concatenation operation, it exists in the DenseNet block and thus, the reduction of conv1×1 may cause the reduction of concatenation at the same time.

6.5 Conclusion

This paper proposes an evolutionary algorithm-based neural architecture search method, ENAS-S, that leverages model stability to seek architectures that suffer from less catastrophic forgetting when learning from a data stream continuously. Meanwhile, benefiting from single-path backbone and block-based search space, ENAS-S generates architectures that conform edge computing. We validate the efficacy of ENAS-S on CIFAR-10 and CIFAR-100 datasets and provided a comprehensive analysis of the evolutionary process. Furthermore, the proposed fitness score is general and can be applied to different NAS search strategies. ENAS-S further inspires research in various topics related to model stability, such as adversarial attack, etc.

Chapter 7

SUMMARY

In this dissertation, different yet uniform methods surrounding model plasticity and model stability are proposed to achieve accurate, efficient and online deep learning. To be specific, the training scheme CGaP achieves small yet accurate inference model through network plasticity; PST alleviates catastrophic forgetting problem and outperforms state-of-the-art single-head accuracy in the continual learning scenario through importance sampling and model segmentation; based on PST, a novel diagram, acquisitive learning, is further proposed to achieve practical, reliable and scalable online learning based on the selected inherited models; meanwhile, experiments validate that model stability is a critical factor to accurate continual learning and thus, ENAS-S is proposed to automatically search for an optimal inherited model in light of model stability in order to achieve higher accuracy when learning from data stream at the edge. The aforementioned methods largely improve the accuracy and efficiency of deep neural networks in the application of continual learning at the edge, especially for emerging systems such as self-driving vehicles and drones.

Along this road, there are several perspectives deserve further thoughts: is architecture the only factor that decides model stability? How about initialization and hyper-parameters which also affect the shape of local minimal? Is continual learning the only application related to model stability? How about the adversarial attack and device variation? What is the key to entirely eliminate catastrophic forgetting? These are all vital topics that deserve more in-depth research.

REFERENCES

[1] X. Du, Z. Li, Y. Ma, and Y. Cao, "Efficient network construction through structural plasticity," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 9, no. 3, pp. 453–464, 2019.

[2] X. Du, G. Charan, F. Liu, and Y. Cao, "Single-net continual learning with progressive segmented training (pst)," in *The 18th International Conferences on Machine Learning and Applications*, 2019.

[3] X. Du, S. K. Venkataramanaiah, Z. Li, J.-s. Seo, F. Liu, and Y. Cao, "Online knowledge acquisition with the selective inherited model," in *2020 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2020, pp. 1–7.

[4] X. Du, Z. Li, J.-s. Seo, F. Liu, and Y. Cao, "Noise-based selection of robust inherited model for accurate continual learning," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, June 2020.

[5] S. K. Venkataramanaiah, X. Du, Z. Li, S. Yin, Y. Cao, and J.-s. Seo, "Efficient and modularized training on fpga for real-time applications," in *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI-20*, C. Bessiere, Ed. International Joint Conferences on Artificial Intelligence Organization, 7 2020, pp. 5237–5239, demos. [Online]. Available: https://doi.org/10.24963/ijcai.2020/755

[6] D. O. Hebb, *The organization of behavior: A neuropsychological theory.* Psychology Press, 2005.

[7] A. L. Samuel, "Computing bit by bit or digital computers made easy," *Proceedings of the IRE*, vol. 41, no. 10, pp. 1223–1230, 1953.

[8] F. Rosenblatt, "The perceptron: a probabilistic model for information storage and organization in the brain." *Psychological review*, vol. 65, no. 6, p. 386, 1958.

[9] B. Widrow and M. Hoff, "Adaptive switching circuits. 1960 ire wescon convention record, 4: 96–104," *New York: IRE. Reprinted in Anderson and Rosenfeld*, 1988.

[10] M. Minsky, S. A. Papert, and L. Bottou, *Perceptrons: An introduction to computational geometry.* MIT press, 2017.

[11] *https://www.tech-quantum.com/solving-xor-problem-using-neural-network-c/.*

[12] P. J. Werbos, "Applications of advances in nonlinear sensitivity analysis," in *System modeling and optimization.* Springer, 1982, pp. 762–770.

[13] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning internal representations by error propagation," California Univ San Diego La Jolla Inst for Cognitive Science, Tech. Rep., 1985.

[14] K. Fukushima, "Neural network model for a mechanism of pattern recognition unaffected by shift in position-neocognitron," *IEICE Technical Report, A*, vol. 62, no. 10, pp. 658–665, 1979.

[15] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[16] C. Cortes and V. Vapnik, "Support-vector networks," *Machine learning*, vol. 20, no. 3, pp. 273–297, 1995.

[17] J. R. Quinlan, "Induction of decision trees," *Machine learning*, vol. 1, no. 1, pp. 81–106, 1986.

[18] L. Breiman, "Random forests," *Machine learning*, vol. 45, no. 1, pp. 5–32, 2001.

[19] Y. Freund and R. E. Schapire, "A decision-theoretic generalization of on-line learning and an application to boosting," *Journal of computer and system sciences*, vol. 55, no. 1, pp. 119–139, 1997.

[20] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Communications of the ACM*, vol. 60, no. 6, pp. 84–90, 2017.

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

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

[23] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 1–9.

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

[25] A. Krizhevsky and G. Hinton, "Learning multiple layers of features from tiny images," 2009.

[26] Y. Netzer, T. Wang, A. Coates, A. Bissacco, B. Wu, and A. Y. Ng, "Reading digits in natural images with unsupervised feature learning," 2011.

[27] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," in *Advances in Neural Information Processing Systems*, 2015, pp. 91–99.

[28] A. Graves, A.-r. Mohamed, and G. Hinton, "Speech recognition with deep recurrent neural networks," in *Acoustics, Speech and Signal Processing (ICASSP), 2013 ieee international conference on.* IEEE, 2013, pp. 6645–6649.

[29] P. Zhang, Y. Goyal, D. Summers-Stay, D. Batra, and D. Parikh, "Yin and Yang: Balancing and answering binary visual questions," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 5014–5022.

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

[31] Y. Chen, T. Luo, S. Liu, S. Zhang, L. He, J. Wang, L. Li, T. Chen, Z. Xu, N. Sun *et al.*, "Dadiannao: A machine-learning supercomputer," in *Proceedings of the 47th Annual IEEE/ACM International Symposium on Microarchitecture.* IEEE Computer Society, 2014, pp. 609–622.

[32] Z. Du, R. Fasthuber, T. Chen, P. Ienne, L. Li, T. Luo, X. Feng, Y. Chen, and O. Temam, "Shidiannao: Shifting vision processing closer to the sensor," in *ACM SIGARCH Computer Architecture News*, vol. 43, no. 3. ACM, 2015, pp. 92–104.

[33] Y.-H. Chen, T. Krishna, J. S. Emer, and V. Sze, "Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks," *IEEE Journal of Solid-State Circuits*, vol. 52, no. 1, pp. 127–138, 2016.

[34] K. Guo, L. Sui, J. Qiu, J. Yu, J. Wang, S. Yao, S. Han, Y. Wang, and H. Yang, "Angel-eye: A complete design flow for mapping cnn onto embedded fpga," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 37, no. 1, pp. 35–47, 2017.

[35] A. Shafiee, A. Nag, N. Muralimanohar, R. Balasubramonian, J. P. Strachan, M. Hu, R. S. Williams, and V. Srikumar, "Isaac: A convolutional neural network accelerator with in-situ analog arithmetic in crossbars," *ACM SIGARCH Computer Architecture News*, vol. 44, no. 3, pp. 14–26, 2016.

[36] J. Qiu, J. Wang, S. Yao, K. Guo, B. Li, E. Zhou, J. Yu, T. Tang, N. Xu, S. Song *et al.*, "Going deeper with embedded fpga platform for convolutional neural network," in *Proceedings of the 2016 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays.* ACM, 2016, pp. 26–35.

[37] C. Farabet, C. Poulet, J. Y. Han, and Y. LeCun, "Cnp: An fpga-based processor for convolutional networks," in *2009 International Conference on Field Programmable Logic and Applications.* IEEE, 2009, pp. 32–37.

[38] S. Han, J. Pool, J. Tran, and W. Dally, "Learning both weights and connections for efficient neural network," in *Advances in neural information processing systems*, 2015, pp. 1135–1143.

[39] H. Li, A. Kadav, I. Durdanovic, and H. S. anda Hans Peter Graf, "Pruning filters for efficient convnets," *CoRR*, vol. abs/1608.08710, 2016. [Online]. Available: http://arxiv.org/abs/1608.08710

[40] H. Hu, R. Peng, Y.-W. Tai, and C.-K. Tang, "Network trimming: A data-driven neuron pruning approach towards efficient deep architectures," *arXiv preprint arXiv:1607.03250*, 2016.

[41] Z. Liu, J. Li, Z. Shen, G. Huang, S. Yan, and C. Zhang, "Learning efficient convolutional networks through network slimming," in *Proceedings of the IEEE International Conference on Computer Vision*, 2017, pp. 2736–2744.

[42] J.-H. Luo, J. Wu, and W. Lin, "Thinet: A filter level pruning method for deep neural network compression," *arXiv preprint arXiv:1707.06342*, 2017.

[43] V. Lebedev and V. Lempitsky, "Fast convnets using group-wise brain damage," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR).* IEEE, 2016, pp. 2554–2564.

[44] W. Wen, C. Wu, Y. Wang, Y. Chen, and H. Li, "Learning structured sparsity in deep neural networks," in *Advances in Neural Information Processing Systems*, 2016, pp. 2074–2082.

[45] J. H. Gilmore, W. Lin, M. W. Prastawa, C. B. Looney, Y. S. K. Vetsa, R. C. Knickmeyer, D. D. Evans, J. K. Smith, R. M. Hamer, J. A. Lieberman *et al.*, "Regional gray matter growth, sexual dimorphism, and cerebral asymmetry in the neonatal brain," *Journal of Neuroscience*, vol. 27, no. 6, pp. 1255–1260, 2007.

[46] S. J. Lipina and J. A. Colombo, *Poverty and brain development during childhood: An approach from cognitive psychology and neuroscience.* American Psychological Association, 2009.

[47] M. Butz and A. van Ooyen, "A simple rule for dendritic spine and axonal bouton formation can account for cortical reorganization after focal retinal lesions," *PLoS computational biology*, vol. 9, no. 10, p. e1003259, 2013.

[48] Z. Liu, M. Sun, T. Zhou, G. Huang, and T. Darrell, "Rethinking the value of network pruning," *arXiv preprint arXiv:1810.05270*, 2018.

[49] Y. He, G. Kang, X. Dong, Y. Fu, and Y. Yang, "Soft filter pruning for accelerating deep convolutional neural networks," *arXiv preprint arXiv:1808.06866*, 2018.

[50] S. Han, X. Liu, H. Mao, J. Pu, A. Pedram, M. A. Horowitz, and W. J. Dally, “EIE: efficient inference engine on compressed deep neural network,” in *Computer Architecture (ISCA), 2016 ACM/IEEE 43rd Annual International Symposium on.* IEEE, 2016, pp. 243–254.

[51] B. Liu, M. Wang, H. Foroosh, M. Tappen, and M. Pensky, “Sparse convolutional neural networks,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2015, pp. 806–814.

[52] T. Ash, “Dynamic node creation in backpropagation networks,” *Connection science*, vol. 1, no. 4, pp. 365–375, 1989.

[53] B. Briedis and T. Gedeon, “Using the grow-and-prune network to solve problems of large dimensionality,” in *Proc. Austral. Conf. Neural Netw.*, 1998.

[54] A. Sakar and R. J. Mammone, “Growing and pruning neural tree networks,” *IEEE Transactions on Computers*, vol. 42, no. 3, pp. 291–299, 1993.

[55] G.-B. Huang, P. Saratchandran, and N. Sundararajan, “A generalized growing and pruning rbf (ggap-rbf) neural network for function approximation,” *IEEE Transactions on Neural Networks*, vol. 16, no. 1, pp. 57–67, 2005.

[56] S. Hussain and A. Basu, “Multiclass classification by adaptive network of dendritic neurons with binary synapses using structural plasticity,” *Frontiers in neuroscience*, vol. 10, p. 113, 2016.

[57] X. Dai, H. Yin, and N. K. Jha, “Nest: A neural network synthesis tool based on a grow-and-prune paradigm,” *arXiv preprint arXiv:1711.02017*, 2017.

[58] Y. Gong, L. Liu, M. Yang, and L. Bourdev, “Compressing deep convolutional networks using vector quantization,” *arXiv preprint arXiv:1412.6115*, 2014.

[59] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio, “Quantized neural networks: Training neural networks with low precision weights and activations,” *The Journal of Machine Learning Research*, vol. 18, no. 1, pp. 6869–6898, 2017.

[60] E. L. Denton, W. Zaremba, J. Bruna, Y. LeCun, and R. Fergus, “Exploiting linear structure within convolutional networks for efficient evaluation,” in *Advances in neural information processing systems*, 2014, pp. 1269–1277.

[61] C. Leng, Z. Dou, H. Li, S. Zhu, and R. Jin, “Extremely low bit neural network: Squeeze the last bit out with admm,” in *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[62] P. Molchanov, S. Tyree, T. Karras, T. Aila, and J. Kautz, “Pruning convolutional neural networks for resource efficient inference,” *arXiv preprint arXiv:1611.06440*, 2016.

[63] V. Nair and G. E. Hinton, "Rectified linear units improve restricted boltzmann machines," in *Proceedings of the 27th international conference on machine learning (ICML-10)*, 2010, pp. 807–814.

[64] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," *arXiv preprint arXiv:1502.03167*, 2015.

[65] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer, "Automatic differentiation in pytorch," 2017.

[66] Y. He, J. Lin, Z. Liu, H. Wang, L.-J. Li, and S. Han, "AMC: Automl for model compression and acceleration on mobile devices," in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018, pp. 784–800.

[67] J. Kirkpatrick, R. Pascanu, N. Rabinowitz, J. Veness, G. Desjardins, A. A. Rusu, K. Milan, J. Quan, T. Ramalho, A. Grabska-Barwinska *et al.*, "Overcoming catastrophic forgetting in neural networks," *Proceedings of the national academy of sciences*, vol. 114, no. 13, pp. 3521–3526, 2017.

[68] F. Zenke, B. Poole, and S. Ganguli, "Continual Learning Through Synaptic Intelligence," *arXiv:1703.04200 [cs, q-bio, stat]*, Mar. 2017, arXiv: 1703.04200.

[69] R. Aljundi, M. Lin, B. Goujaud, and Y. Bengio, "Online continual learning with no task boundaries," *arXiv:1903.08671 [cs, stat]*, Mar. 2019, arXiv: 1903.08671.

[70] A. Chaudhry, P. K. Dokania, T. Ajanthan, and P. H. S. Torr, "Riemannian Walk for Incremental Learning: Understanding Forgetting and Intransigence," *arXiv:1801.10112 [cs]*, vol. 11215, pp. 556–572, 2018, arXiv: 1801.10112.

[71] A. Chaudhry, M. Ranzato, M. Rohrbach, and M. Elhoseiny, "Efficient lifelong learning with a-gem," *arXiv preprint arXiv:1812.00420*, 2018.

[72] S.-W. Lee, J.-H. Kim, J. Jun, J.-W. Ha, and B.-T. Zhang, "Overcoming catastrophic forgetting by incremental moment matching," in *Advances in neural information processing systems*, 2017, pp. 4652–4662.

[73] D. Lopez-Paz and M. Ranzato, "Gradient Episodic Memory for Continual Learning," in *Advances in Neural Information Processing Systems 30*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, Eds. Curran Associates, Inc., 2017, pp. 6467–6476.

[74] S.-A. Rebuffi, A. Kolesnikov, G. Sperl, and C. H. Lampert, "iCaRL: Incremental Classifier and Representation Learning," *arXiv:1611.07725 [cs, stat]*, Nov. 2016, arXiv: 1611.07725.

[75] J. Zhang, J. Zhang, S. Ghosh, D. Li, J. Zhu, H. Zhang, and Y. Wang, "Regularize, Expand and Compress: Multi-task based Lifelong Learning via NonExpansive AutoML," *arXiv:1903.08362 [cs]*, Mar. 2019, arXiv: 1903.08362.

[76] J. Zhang, J. Zhang, S. Ghosh, D. Li, S. Tasci, L. Heck, H. Zhang, and C.-C. J. Kuo, "Class-incremental Learning via Deep Model Consolidation," *arXiv:1903.07864 [cs]*, Mar. 2019, arXiv: 1903.07864.

[77] A. A. Rusu, N. C. Rabinowitz, G. Desjardins, H. Soyer, J. Kirkpatrick, K. Kavukcuoglu, R. Pascanu, and R. Hadsell, "Progressive neural networks," *arXiv preprint arXiv:1606.04671*, 2016.

[78] J. Xu and Z. Zhu, "Reinforced continual learning," in *Advances in Neural Information Processing Systems*, 2018, pp. 899–908.

[79] J. Yoon, E. Yang, J. Lee, and S. J. Hwang, "Lifelong Learning with Dynamically Expandable Networks," *arXiv:1708.01547 [cs]*, Aug. 2017. [Online]. Available: http://arxiv.org/abs/1708.01547

[80] J. Serrà, D. Surís, M. Miron, and A. Karatzoglou, "Overcoming catastrophic forgetting with hard attention to the task," *arXiv:1801.01423 [cs, stat]*, Jan. 2018, arXiv: 1801.01423.

[81] R. Aljundi, F. Babiloni, M. Elhoseiny, M. Rohrbach, and T. Tuytelaars, "Memory aware synapses: Learning what (not) to forget," in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018, pp. 139–154.

[82] Z. Li and D. Hoiem, "Learning without forgetting," *IEEE transactions on pattern analysis and machine intelligence*, vol. 40, no. 12, pp. 2935–2947, 2017.

[83] A. Mallya, D. Davis, and S. Lazebnik, "Piggyback: Adapting a single network to multiple tasks by learning to mask weights," in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018, pp. 67–82.

[84] A. Mallya and S. Lazebnik, "Packnet: Adding multiple tasks to a single network by iterative pruning," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 7765–7773.

[85] F. M. Castro, M. J. Marín-Jiménez, N. Guil, C. Schmid, and K. Alahari, "End-to-end incremental learning," in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018, pp. 233–248.

[86] K. Javed and F. Shafait, "Revisiting distillation and incremental classifier learning," *arXiv preprint arXiv:1807.02802*, 2018.

[87] S. Farquhar and Y. Gal, "Towards robust evaluations of continual learning," *arXiv preprint arXiv:1805.09733*, 2018.

[88] C. V. Nguyen, Y. Li, T. D. Bui, and R. E. Turner, "Variational continual learning," *arXiv preprint arXiv:1710.10628*, 2017.

[89] P. Dhar, R. V. Singh, K. Peng, Z. Wu, and R. Chellappa, "Learning without memorizing," *CoRR*, vol. abs/1811.08051, 2018. [Online]. Available: http://arxiv.org/abs/1811.08051

[90] Y. Wu, Y. Chen, L. Wang, Y. Ye, Z. Liu, Y. Guo, and Y. Fu, "Large scale incremental learning," *CoRR*, vol. abs/1905.13260, 2019. [Online]. Available: http://arxiv.org/abs/1905.13260

[91] C. Wu, L. Herranz, X. Liu, J. van de Weijer, B. Raducanu *et al.*, "Memory replay gans: Learning to generate new categories without forgetting," in *Advances In Neural Information Processing Systems*, 2018, pp. 5962–5972.

[92] A. Rios and L. Itti, "Closed-loop gan for continual learning," *arXiv preprint arXiv:1811.01146*, 2018.

[93] A. Seff, A. Beatson, D. Suo, and H. Liu, "Continual learning in generative adversarial nets," *arXiv preprint arXiv:1705.08395*, 2017.

[94] X. Du, Z. Li, and Y. Cao, "CGaP: Continuous Growth and Pruning for Efficient Deep Learning," *arXiv preprint arXiv:1905.11533*, 2019.

[95] S. Kolala Venkataramanaiah, Y. Ma, S. Yin, E. Nurvithadhi, A. Dasu, Y. Cao, and J.-s. Seo, "Automatic compiler based fpga accelerator for cnn training," in *2019 29th International Conference on Field Programmable Logic and Applications (FPL).* IEEE, 2019.

[96] Y. Ma, Y. Cao, S. Vrudhula, and J.-s. Seo, "Optimizing the convolution operation to accelerate deep neural networks on fpga," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 26, no. 7, pp. 1354–1367, 2018.

[97] A. Chaudhry, M. Ranzato, M. Rohrbach, and M. Elhoseiny, "Efficient lifelong learning with a-gem," *arXiv preprint arXiv:1812.00420*, 2018.

[98] M. McCloskey and N. J. Cohen, "Catastrophic interference in connectionist networks: The sequential learning problem," in *Psychology of learning and motivation.* Elsevier, 1989, vol. 24, pp. 109–165.

[99] J. Yoon, E. Yang, J. Lee, and S. J. Hwang, "Lifelong learning with dynamically expandable networks," *arXiv preprint arXiv:1708.01547*, 2017.

[100] A. M. Zador, "A critique of pure learning and what artificial neural networks can learn from animal brains," *Nature communications*, vol. 10, no. 1, pp. 1–7, 2019.

[101] M. Ingalhalikar, A. Smith, D. Parker, T. D. Satterthwaite, M. A. Elliott, K. Ruparel, H. Hakonarson, R. E. Gur, R. C. Gur, and R. Verma, "Sex differences in the structural connectome of the human brain," *Proceedings of the National Academy of Sciences*, vol. 111, no. 2, pp. 823–828, 2014.

[102] H. Moravec, *Mind children: The future of robot and human intelligence.* Harvard University Press, 1988.

[103] S. J. Pan and Q. Yang, "A survey on transfer learning," *IEEE Transactions on knowledge and data engineering*, vol. 22, no. 10, pp. 1345–1359, 2009.

[104] H. Li, Z. Xu, G. Taylor, C. Studer, and T. Goldstein, "Visualizing the loss landscape of neural nets," in *Advances in Neural Information Processing Systems*, 2018, pp. 6389–6399.

[105] Z. He, A. S. Rakin, and D. Fan, "Parametric noise injection: Trainable randomness to improve deep neural network robustness against adversarial attack," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019, pp. 588–597.

[106] S. K. Venkataramanaiah, Y. Ma, S. Yin, E. Nurvithadhi, A. Dasu, Y. Cao, and J.-s. Seo, "Automatic compiler based fpga accelerator for cnn training," in *2019 29th International Conference on Field Programmable Logic and Applications (FPL)*. IEEE, 2019, pp. 166–172.

[107] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 4700–4708.

[108] X. Long, S. Hu, Y. Hu, Q. Gu, and I. Ishii, "An fpga-based ultra-high-speed object detection algorithm with multi-frame information fusion," *Sensors*, vol. 19, no. 17, p. 3707, 2019.

[109] C. Liu, B. Yan, C. Yang, L. Song, Z. Li, B. Liu, Y. Chen, H. Li, Q. Wu, and H. Jiang, "A spiking neuromorphic design with resistive crossbar," in *2015 52nd ACM/EDAC/IEEE Design Automation Conference (DAC)*. IEEE, 2015, pp. 1–6.

[110] S. K. Venkataramanaiah, Y. Ma, S. Yin, E. Nurvithadhi, A. Dasu, Y. Cao, and J.-s. Seo, "Automatic compiler based fpga accelerator for cnn training," in *2019 29th International Conference on Field Programmable Logic and Applications (FPL)*. IEEE, 2019, pp. 166–172.

[111] S. Venkataramanaiah, H. sok Suh, S. Yin, E. Nurvithadhi, A. Dasu, Y. Cao, and J. Seo, "FPGA-based Low-Batch Training Accelerator for Modern CNNs Featuring High Bandwidth Memory," in *IEEE International Conference On Computer Aided Design (ICCAD)*, 2020, p. accepted for publication.

[112] M. Deo, J. Schulz, and L. Brown, "Intel stratix 10 mx devices solve the memory bandwidth challenge," *Intel White Paper*, 2016.

[113] Z. Li, X. Du, and Y. Cao, "Gar: Graph assisted reasoning for object detection," in *The IEEE Winter Conference on Applications of Computer Vision*, 2020, pp. 1295–1304.

[114] S.-A. Rebuffi, A. Kolesnikov, G. Sperl, and C. H. Lampert, "icarl: Incremental classifier and representation learning," in *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, 2017, pp. 2001–2010.

[115] M. Mermillod, A. Bugaiska, and P. BONIN, "The stability-plasticity dilemma: investigating the continuum from catastrophic forgetting to age-limited learning effects," *Frontiers in Psychology*, vol. 4, p. 504, 2013.

[116] T. Elsken, J. H. Metzen, and F. Hutter, "Neural architecture search: A survey," *arXiv preprint arXiv:1808.05377*, 2018.

[117] H. Liu, K. Simonyan, and Y. Yang, "Darts: Differentiable architecture search," *arXiv preprint arXiv:1806.09055*, 2018.

[118] C. Gao, Y. Chen, S. Liu, Z. Tan, and S. Yan, "Adversarialnas: Adversarial neural architecture search for gans," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 5680–5689.

[119] M. Suganuma, S. Shirakawa, and T. Nagao, "A genetic programming approach to designing convolutional neural network architectures," in *Proceedings of the genetic and evolutionary computation conference*, 2017, pp. 497–504.

[120] C. He, H. Ye, L. Shen, and T. Zhang, "Milenas: Efficient neural architecture search via mixed-level reformulation," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 11 993–12 002.

[121] Y. Sun, B. Xue, M. Zhang, and G. G. Yen, "Evolving deep convolutional neural networks for image classification," *IEEE Transactions on Evolutionary Computation*, vol. 24, no. 2, pp. 394–407, 2019.

[122] Y. Sun, B. Xue, M. Zhang, and G. Yen, "Completely automated cnn architecture design based on blocks," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 31, no. 4, pp. 1242–1254, 2019.

[123] C. Li, J. Peng, L. Yuan, G. Wang, X. Liang, L. Lin, and X. Chang, "Blockwisely supervised neural architecture search with knowledge distillation," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 1989–1998.

[124] A. Wan, X. Dai, P. Zhang, Z. He, Y. Tian, S. Xie, B. Wu, M. Yu, T. Xu, K. Chen *et al.*, "Fbnetv2: Differentiable neural architecture search for spatial and channel dimensions," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 12 965–12 974.

[125] D. Stamoulis, R. Ding, D. Wang, D. Lymberopoulos, B. Priyantha, J. Liu, and D. Marculescu, "Single-path nas: Designing hardware-efficient convnets in less than 4 hours," in *Joint European Conference on Machine Learning and Knowledge Discovery in Databases.* Springer, 2019, pp. 481–497.

[126] Z. Zhong, J. Yan, W. Wu, J. Shao, and C.-L. Liu, "Practical block-wise neural network architecture generation," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 2423–2432.

[127] R. Shin, C. Packer, and D. Song, "Differentiable neural network architecture search," 2018.

[128] R. Luo, F. Tian, T. Qin, E. Chen, and T.-Y. Liu, "Neural architecture optimization," in *Advances in neural information processing systems*, 2018, pp. 7816–7827.

[129] A. Brock, T. Lim, J. M. Ritchie, and N. Weston, "Smash: one-shot model architecture search through hypernetworks," *arXiv preprint arXiv:1708.05344*, 2017.

[130] B. Wu, X. Dai, P. Zhang, Y. Wang, F. Sun, Y. Wu, Y. Tian, P. Vajda, Y. Jia, and K. Keutzer, "Fbnet: Hardware-aware efficient convnet design via differentiable neural architecture search," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019, pp. 10 734–10 742.

[131] G. Bender, P.-J. Kindermans, B. Zoph, V. Vasudevan, and Q. Le, "Understanding and simplifying one-shot architecture search," in *International Conference on Machine Learning*, 2018, pp. 550–559.

[132] B. Zoph and Q. V. Le, "Neural architecture search with reinforcement learning," 2017.

[133] I. Bello, B. Zoph, V. Vasudevan, and Q. V. Le, "Neural optimizer search with reinforcement learning," *arXiv preprint arXiv:1709.07417*, 2017.

[134] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le, "Learning transferable architectures for scalable image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 8697–8710.

[135] V. Nekrasov, H. Chen, C. Shen, and I. Reid, "Fast neural architecture search of compact semantic segmentation models via auxiliary cells," 2019.

[136] H. Liu, K. Simonyan, O. Vinyals, C. Fernando, and K. Kavukcuoglu, "Hierarchical representations for efficient architecture search," *arXiv preprint arXiv:1711.00436*, 2017.

[137] T. Elsken, J. H. Metzen, and F. Hutter, "Efficient multi-objective neural architecture search via lamarckian evolution," *arXiv preprint arXiv:1804.09081*, 2018.

[138] E. Real, A. Aggarwal, Y. Huang, and Q. V. Le, "Regularized evolution for image classifier architecture search," in *Proceedings of the aaai conference on artificial intelligence*, vol. 33, 2019, pp. 4780–4789.

[139] D. R. So, C. Liang, and Q. V. Le, "The evolved transformer," *arXiv preprint arXiv:1901.11117*, 2019.

[140] X. Dong and Y. Yang, "Searching for a robust neural architecture in four gpu hours," in *Proceedings of the IEEE Conference on computer vision and pattern recognition*, 2019, pp. 1761–1770.

[141] Y. Sun, B. Xue, M. Zhang, and G. G. Yen, "Automatically evolving cnn architectures based on blocks," *arXiv preprint arXiv:1810.11875*, 2018.

[142] H. Pham, M. Y. Guan, B. Zoph, Q. V. Le, and J. Dean, "Efficient neural architecture search via parameter sharing," *arXiv preprint arXiv:1802.03268*, 2018.

[143] D. Whitley, "A genetic algorithm tutorial," *Statistics and computing*, vol. 4, no. 2, pp. 65–85, 1994.

[144] F. Assunção, J. Correia, R. Conceição, M. J. M. Pimenta, B. Tomé, N. Lourenço, and P. Machado, "Automatic design of artificial neural networks for gamma-ray detection," *IEEE Access*, vol. 7, pp. 110 531–110 540, 2019.

[145] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer, "Automatic differentiation in pytorch," 2017.

[146] M. Suganuma, M. Kobayashi, S. Shirakawa, and T. Nagao, "Evolution of deep convolutional neural networks using cartesian genetic programming," *Evolutionary Computation*, vol. 28, no. 1, pp. 141–163, 2020.