Study on Robust Large-scale Neural Network Optimization

堅牢な大規模ニューラルネットワーク最適化に関する研究

Enzhi Zhang

A Dissertation

submitted in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

in

Information Science and Technology

in the field of

Computer Science and Information Technology

at the

Graduate School of Information Science and Technology

Hokkaido University, Japan

2025

Abstract

Overfitting in machine learning and deep learning occurs when a model learns not only the underlying patterns in the training data but also the noise, outliers, and specific details that do not generalize to unseen data. This problem often arises when the model is too complex relative to the amount of training data or the variability in the data. When a model has too many parameters or is overly flexible, such as in deep neural networks with many layers and units, it has the capacity to perfectly fit the training data, capturing even the smallest irregularities or random fluctuations. As a result, the model becomes highly specialized to the training set and performs poorly on new data. A primary cause of overfitting is insufficient or unrepresentative training data. If the dataset is too small or lacks diversity, the model encounters only a limited range of examples, which leads it to memorize the data rather than learn generalizable features. In cases of deep learning, where models have vast numbers of parameters, the imbalance between data and model capacity is amplified, making the model prone to overfitting. The lack of variability in the data also contributes to overfitting; if the data is too homogenous or does not capture the full range of real-world variations, the model can latch onto irrelevant patterns specific to the training examples. Additionally, when the data is relatively noise-free, models may become too focused on precise details, learning patterns that are not truly reflective of the broader data distribution. Finally, the tendency of a model to overfit increases if it is trained for too many epochs, leading it to adapt more to the training data than to underlying, generalizable features. In essence, overfitting occurs when the model's complexity exceeds what the data can support, leading to a failure to generalize beyond the training examples.

To combat overfitting, several strategies can be employed. Data augmentation artificially increases the diversity of the training set by applying transformations such as rotations, flips, and color changes to input data, forcing the model to learn more generalized features. Model structure adjustments, such as introducing dropout (randomly deactivat-

ing neurons during training) and using batch normalization, help regulate the model's capacity and stabilize learning. Furthermore, incorporating regularization techniques like L1/L2 penalties constrains the magnitude of model weights, discouraging overly complex solutions. The choice of optimizer also influences overfitting; optimizers like AdamW, which decouples weight decay from gradient updates, provide proper regularization and prevent the model from overfitting by penalizing large weights. Additionally, techniques like learning rate schedules and early stopping halt training at the right moment, before the model begins to overfit. Combining these strategies helps build models that generalize well, achieving a balance between complexity and performance on unseen data.

In this thesis, we introduce several novel frameworks from different direction to address the persistent overfitting problem in deep learning models. By incorporating validation loss into the training objective, we enable the model to self-regulate its learning process, a technique we term "meta loss landscape search." To further enhance generalization, we propose a "meta sample generation/augmentation" method that generates adversarial examples based on the model's weaknesses, as identified by the testing loss. Finally, we introduce "Adaptive Patching," a technique tailored for Vision Transformer (ViT) models, which structurally organizes the data within samples to improve generalization. As a summary of the above methods, our research can be summarized as a strategy to solve the problem of model overfitting from three aspects: meta-optimization at the optimizer level, adversarial learning at the data level, and comprehensive optimization of specific models and data.

Acknowledgments

I would like to express my heartfelt gratitude to all those who have supported and assisted me throughout my doctoral journey. This work would not have been possible without your unwavering encouragement and invaluable help.

First and foremost, I owe my deepest gratitude to my supervisor, Professor Masaharu Munetomo. I still vividly recall our first remote interview during the COVID-19 pandemic in Beijing. His generosity, engaging conversation, and insightful ideas have consistently provided me with invaluable guidance at crucial moments throughout my doctoral journey.

I would also like to thank Dr. Wahib Muhamed for his profound insights into deep learning and high-performance computing, which laid the foundational concepts for this work. More importantly, his passion for academic research, pursuit of excellence, and approach to collaboration have deeply influenced my research methodology and working style, shaping both my current and future academic endeavors.

My sincere thanks also go to Dr. Peng Chen, Dr. Xiao Wang, and Dr. Isaac Lyngass. This work would not have been possible without their time and effort. I am also grateful to Professor Xiang Wang, Dr. An Zhang, and Dr. Zhiyuan Liu, whose insights into graph neural networks and molecular modeling have been a great source of inspiration. I am especially thankful to my collaborators, Special Assistant Professor Rui Zhong, Yang Cao, and Xingbang Du, for their unwavering support and fruitful collaboration throughout my doctoral journey.

I would like to extend my gratitude to all the reviewers of my published papers in journals, workshops, and international conferences. Their valuable comments and suggestions have greatly contributed to the development of my research.

I am also thankful to Hokkaido University, RIKEN, and ORNL for providing essential resources and creating a supportive research environment.

Last but certainly not least, I am deeply grateful to my family for their unwavering

support and encouragement throughout my doctoral studies. Without their constant love, patience, and belief in me, this dissertation would not have been possible. Their understanding and sacrifices have been my foundation, and I cannot thank them enough for being by my side every step of the way.

I would also like to express my thanks to Ms. Ruqing Wang, whose encouragement ignited my desire to pursue a Ph.D. during a walk one evening in 2017 at Kamisyakuji Park. Now, as she embarks on her own doctoral journey, I know that the path ahead will be challenging, but I offer my sincere wishes for her success and hope that her journey unfolds smoothly.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Deep learning, a subset of machine learning, has revolutionized artificial intelligence by enabling machines to learn complex patterns from large datasets. Through the use of artificial neural networks with multiple layers, deep learning models have achieved remarkable success in various domains. In computer vision, these models can accurately classify images, detect objects, and even generate realistic images. Natural language processing has also benefited significantly, with deep learning powering advanced language translation, text generation, and sentiment analysis. Beyond these, deep learning has found applications in healthcare for medical image analysis and drug discovery, in autonomous systems for self-driving cars and robotics, and in finance for fraud detection and algorithmic trading. The ability of deep learning to learn from data and make intelligent decisions has opened up new frontiers in artificial intelligence, promising to shape the future of technology and society.

Deep learning models, despite their impressive performance, are susceptible to overfitting, a phenomenon where the model becomes overly specialized to the training data, leading to poor generalization on unseen data. This occurs when the model learns the noise and idiosyncrasies of the training data rather than the underlying patterns. As a result, the model may perform exceptionally well on the training set but poorly on the validation and

test sets. To mitigate overfitting, various techniques are employed, such as early stopping, regularization (L1/L2 regularization, dropout), and data augmentation. These methods help to constrain the model's capacity and prevent it from memorizing the training data, thereby improving its generalization ability.

In this research, we introduce a novel framework to address the persistent overfitting problem in deep learning models. By incorporating validation loss into the training objective, we enable the model to self-regulate its learning process, a technique we term "meta loss landscape search." To further enhance generalization, we propose a "meta sample generation/augmentation" method that generates adversarial examples based on the model's weaknesses, as identified by the testing loss. Finally, we introduce "Adaptive Patching," a technique tailored for Vision Transformer (ViT) models, which structurally organizes the data within samples to improve generalization. As a summary of the above methods, our research can be summarized as a strategy to solve the problem of model overfitting from three aspects: meta-optimization at the optimizer level, adversarial learning at the data level, and comprehensive optimization of specific models and data.

1.1 Motivation for this Thesis

Overfitting is a critical problem in deep learning, where a model becomes overly specialized to its training data, sacrificing its ability to generalize to new, unseen data. This phenomenon occurs when a model captures noise and random fluctuations within the training dataset, leading to high variance and poor performance on unseen data. The model, essentially memorizing the training data, fails to learn underlying patterns and trends, resulting in a model that is highly accurate on the training set but performs poorly on validation and test sets. To combat overfitting, various techniques can be employed. One approach is to reduce model complexity by using simpler architectures with fewer parameters, limiting the model's capacity to memorize noise. Another effective method is to increase the size of

the training dataset, providing the model with more diverse examples to learn from. Regularization techniques, such as L1 and L2 regularization, introduce penalties to the model's loss function, discouraging it from assigning excessive weight to individual features. Early stopping is a technique that monitors the model's performance on a validation set and halts training when the performance starts to degrade, preventing the model from overfitting. Additionally, data augmentation techniques can be used to artificially expand the training dataset by creating modified versions of existing data points, improving the model's ability to generalize.

Despite the experts' intuition, theoretical, and/or empirical insight of the high dimensionality and non-linearity of the deep models,[6] a natural question arises: *"Can we enable the gradient descent optimizer to decrease the validation loss, even if zero training loss indicates learning has stopped?"*. To address this problem, meta-learning[7] and especially meta-HPO(Hyper Parameter Optimization) methods define a bi-level objective framework. Typically, there are two common approaches for such bi-level optimization. In the first approach, we optimize the outer target and let the controller search the parameter space. Then, we improve the controller using the meta-info from the inner optimization. This method is easy to implement, for example, grid search, yet is costly to compute due to the computationally-intense inner optimization.[8,9] In the second approach, we update the outer objective directly: a meta-gradient-based approach calculates the high-order gradient to the hyperparameters or settings.[10,11] However, despite the impractical cost of the high-order Hessian matrix, the degradation of the meta-gradient is also serious. Thus, decoupling the meta-gradient while training the inner model directly and efficiently is needed.

Therefore in summary, the motivation of this study is to increase the search space as little as possible, taking advantage of the nature of the data itself, the fitting ability of the deep model itself, and the training optimizer commonly used in deep learning. The overall generalization ability is improved based on not changing the original overall training. This motivation has the following considerations: inheritability, which facilitates the original

research work to make changes on this basis; feasibility, which facilitates users to repeat this implementation; and flexibility, which reserves room for model, optimizer, and data modification, further improving performance.

1.2 Objectives of this Thesis

This thesis focuses on solving the overfitting problem faced during neural network training. The solution is to reduce the gap between validation loss and train loss through the fitting ability of the neural network. It mainly consists of the following three goals:

- Development of a validation loss regression optimization system in a Q-network can play the role of a controller/optimizer that inherits the validation knowledge of changes in the weights space, which can then be used to decrease the validation loss of workers on the dataset and be transferable to other datasets by controlling the gradient in the action space.

- Achieves notable sample efficiency improvements for few-shot training on common architectures over several image-based and text-based datasets and deep neural network-based models.

- Proposal of a novel and general approach that can solve the long-sequence challenge in ViTs while avoiding overfitting; it preserves the dense self-attention merits and reduces sequence length dramatically to boost the segmentation efficiency. This paves the way for enhanced applications of ViTs in high-resolution scientific imaging domains..

1.3 Contributions of this Thesis

The main contributions in this thesis are listed as follows:

Validation Loss Landscape Awareness Meta-optimizer via Transferable Q-Net: In comparison to other meta-learning methods, our method TLLE applies a Q-Net to learn the mapping from weights, instead of hyper-parameters, to the Q value, and such an approach can provide a benefit in two aspects. First, the weights-value pairs regression releases the online training of the controller during the inner optimization, which improves the sample collection speed and allows us to update the hyperparameters online as a scheduler. Second, a fine-trained Q-Net can provide a beneficial meta-gradient for continued training and avoid calculating the Hessian matrix. By adjusting the learning rate and the penalty factor, TLLE could ”plan” a path in the loss landscape similar to what was previously found and further explore the validation loss landscape. We experimentally demonstrate the transferability of the Q-Net to other datasets. We show how precisely the Q-Net can predict the Q-value from the source worker's weights and can later be used to evaluate the target worker's weights after fine-tuning. To make training the controller model computationally practical, we propose three improvements for TLLE: downsampling the weights of the workers, an online updating schema, and a normalized weight representation to improve generalization.

Validation loss aware sample generator for few-shot training: we show how our method MGDAO can generate the validation loss-oriented augmentation training samples for few-shot. To illustrate the generalization for different types of input samples, we conduct experiments on both image classification tasks and system intrusion classification tasks. Our scheduler generates samples that fit the validation loss compared to another automatic data-augmentation method. That is the reason we call the augmented samples fake test samples. As two approaches of MGDAO, first by pixel-wise similarity, we can ”rediscover” image data augmentation rules, for example, adding noise or masking, de-

signed by human experts. Then, by the feature-level similarity, we can even change the feature of samples which is hard to achieve by engineering. MGDAO achieves notable improvements for few-shot training on common architectures over several image-based and text-based datasets and models.

A Adaptive Patching(AP) for ViT-based High-resolution Segmentation: we show how our method AP solution to reduce the total number of patches extracted from an image, thereby reducing the overall training cost. This not only reduces the cost of computing and memory, it also allows for using small sizes for patches, e.g., 4x4 or 2x2, which is favorable for high segmentation quality. Our quantitative results demonstrate that at the same resolution levels $[512, 1024, 4096, 8192, 16384]$, a model using AP can employ nearly $8\times$ smaller patch sizes or $64\times$ longer sequence lengths, while maintaining the same cost of traditional patching. We conducted experiments on Frontier supercomputer, with up to $2,048$ MI250X using real-world high-resolution pathology datasets. At a fixed compute budget, and up to the depth of 13 multi-resolutions, we can scale to image resolutions up to $16K^2$, and lower the patch size from 16×16 to the minimum 2×2 on a vision transformer. Unlike existing methods that modify attention mechanisms, our solution preserves the original attention mechanism. This ensures seamless integration into any vision transformer. AP is a very low-overhead pre-processing solution, that is further amortized over epochs: the overhead is effectively negligible.

1.4 Problem Origin of the Deep Learning Framework

Overfitting is a notorious problem in deep neural networks. It manifests itself as a decrease in training loss to a certain value, followed by an increase in the test loss. Continuing to train an overfitted network eventually leads to almost zero train loss and high test loss, which the model is memorizing the training dataset.

There are many efforts to solve the overfitting problem, includes but not limited to:

regularization,[12] early stopping,[13] dropout,[14] adversarial training[15,16] and data augmentation.[17] These methods are designed to avoid or make it harder to memorize the train set by setting restrictions to the trainable weights or inputs. Although they have already worked well in the practice, setting penalty factors to the restrictions highly correlated to the training process, eg: small l_2 penalties still lead to zero train loss, and adversarial samples can hurt generalization.[18] However, a more serious fact is that the concept of avoiding memorization by adding randomness may be challenged because even if the dataset and label are totally randomly corrupted, the MLP model could still achieve zero train loss.[19]

An important question asked in[20] is: "do we need zero training loss after achieving zero training error?". As we described before, zero train loss often results in severe overfitting. What is even worse is that zero training loss cannot provide enough gradient magnitude for the model to jump out of local minima. Therefore, the authors let the model do gradient ascend instead of gradient descent when the training loss is lower than a threshold. When the training loss is equal to this constant, they let the model update itself by a random walk.

To answer the question asked by the authors, the problem is not that zero training *minimizes* the objective function (since the model is designed to optimize the objective function, i..e. zero training loss). The problem is that the approach of *minimizing* the objective function provides the wrong gradient for the model to optimize itself when the training loss is low.

1.5 Overview of methods for solving overfitting problem in Neural Network Training.

There are many efforts to solve the overfitting problem, includes but not limited to: regularization,[12] early stopping,[13] dropout,[14] adversarial training[15,16] and data augmentation.[17] These methods are designed to avoid or make it harder to memorize the train set by setting restrictions to the trainable weights or inputs. Although they have already worked

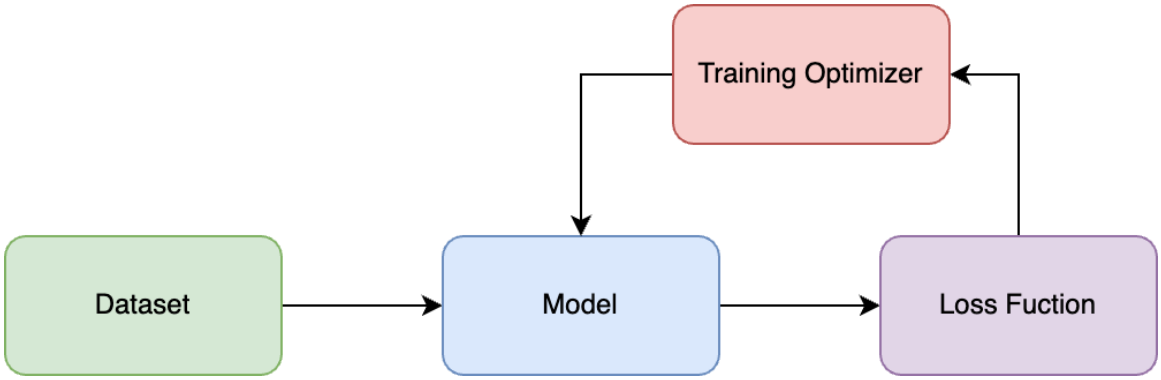


Figure 1.1: General framework for training deep learning model(Arrow represent tensor's flow direction)

well in the practice, setting penalty factors to the restrictions highly correlated to the training process, eg: small l_2 penalties still lead to zero train loss, and adversarial samples can hurt generalization.[18] However, a more serious fact is that the concept of avoiding memorization by adding randomness may be challenged because even if the dataset and label are totally randomly corrupted, the MLP model could still achieve zero train loss.[19]

Recent research has focused on automatically designing data augmentation techniques tailored to specific datasets and tasks to overcome this limitation. One approach is to efficiently select effective combinations of image transformations from a large pool of candidate operations. For instance, AutoAugment[21] and related methods[22–25] optimize the selection of augmentation operations to improve validation performance, achieving state-of-the-art results on several benchmarks. Another approach involves using conditional generative models[26,27] to improve model performance in low-data settings.

1.6 Benefits of the System

The proposed research work enables better training of deep neural networks to avoid overfitting. We achieved notable improvements for few-shot training on common architectures over several image-based and text-based datasets and models.

1.7 Organization of this Thesis

This thesis is organized into seven chapters as shown in Figure 1.2, with the core chapters comprising several symposia, conferences, and journal papers published during the Ph.D. course.

First, Chapter 1 presents general information of this thesis, the motivations of this thesis, contributions, and the objectives of this thesis. It also highlights the current problem situation and the reasons why we chose to do our research in this field of study. The possible benefits of our research to the society are also presented. Chapter 2 contains background and literature reviews of the related technologies of the thesis.

Chapter 3 proposed a supervisor-students training framework (REVAL) that could learn from past training trajectories to improve student generalization on the image classification task. However, some problems still need to be solved, including the supervisor's dimension disaster, the consistency of training the supervisor, and experiments on large deep models. Therefore, in this study, we extended our previous REVAL work with experiments on a large model(ResNet-56) and a consistency discussion about the supervisor to further improve our work. More importantly, through visualization, we explain why the gradients provided by the supervisor are more conducive to converging to a better solution.

Chapter 4 proposed a validation-loss landscape exploration/exploitation method called VKI (Validation Knowledge Inheritance). We reformulate the traditional gradient optimization problem as a reinforcement learning task to explore the validation-loss landscape. In particular, by treating the gradient descent process as a Markov Decision Process (MDP), where the validation losses are treated as the costs, we train a Q-network as a controller to learn the future rewards and later use it to decrease the validation loss of workers.

Chapter 5 proposed a new method named Meta Generative Data Augmentation Optimization (MGDAO) to solve the limited operations in the AutoAugment approach and the hard training in the generative adversarial model. More specifically, we replace the oper-

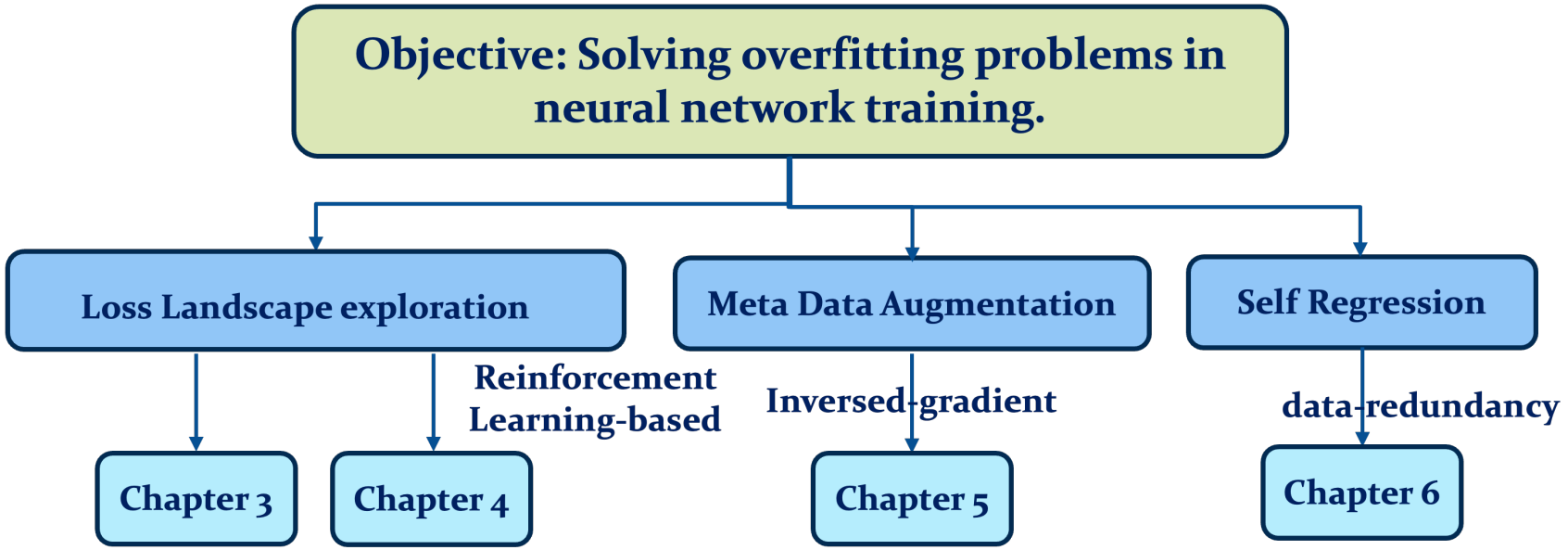


Figure 1.2: Thesis Outline

ation action space in the AutoAugment course with a deep-style generator and return the discriminator with the validation loss from the target model's evaluation process.

Chapter 6 proposed an Adaptive Patch(AP) method that is compatible with any vision transformer. AP is a pre-processing solution that uses a quadtree to partition each image in the dataset into mixed-scale patches, based on the level of detail in different regions in the image. Larger patches, that carry fewer image details are then downscaled such that all patches become the same size when being fed to the model, while keeping the core attention mechanisms and ViT model architecture intact. To demonstrate AP's scalability, we conducted extensive training of transformer-based vision models with small patch sizes for long sequences of high-resolution images.

Chapter 7 concludes this thesis with a summary of the work conducted and suggestions for future research.

Chapter 2

Background

Overfitting in deep learning arises when a model captures not only essential patterns in training data but also noise and irrelevant details. This results in strong performance on training sets but poor generalization to unseen examples. Overfitting typically occurs when the model complexity exceeds the available training data, leading to memorization instead of generalization. Various strategies can address this issue effectively. Data augmentation increases training set diversity through transformations such as flips, rotations, and color adjustments, encouraging models to learn generalized features. Structural modifications like dropout, which randomly deactivates neurons during training, and batch normalization, which stabilizes learning, help regulate capacity. Regularization techniques, including L1 and L2 penalties, constrain weight magnitudes, discouraging overly complex solutions. The optimizer also plays a role; for instance, AdamW decouples weight decay from gradient updates, promoting proper regularization by penalizing large weights. Learning rate schedules and early stopping prevent overfitting by halting training at an optimal point before performance deteriorates. Combining these methods enables the creation of models that balance complexity with performance, ensuring strong generalization to new data.

2.1 Training of Model

We adopt the general settings in the training of neural network models for classification tasks. Consider an image dataset D consisting of input images $x \in R^d$ and the one-hot label $y \in R^K$ where K is the number of classes. We denote the network model as $f : R^d \to R^K$ and parameterized by the trainable weights θ. For each sample $(x, y) \in D$, the output prediction is given by $\tilde{y} = f(x; \theta)$. Let $l : R^K \times R^K \to R$ denote a loss function (commonly use the softmax cross-entropy loss l_{CE} for the classification tasks) and then the optimization process can be written as:

$$\min_{\theta} \quad L(x, y) = \frac{1}{N} \sum_{n}^{N} ||f(x_i; \theta) - y_i||_{l_{CE}} \tag{2.1}$$

Where N is the size of the training dataset D and the loss L is the training loss.

However, the gap between test and training losses is a manifestation of the problem of overfitting. Overfitting arises from the size and quality of datasets, and the design of the model. To tackle the overfitting problem, the most widely used method is *regularization* or called *weight decay*.[12,28,29] We rewrite the objective with regularization as:

$$\min_{\theta} \quad L(x, y) = \frac{1}{N} \sum_{n}^{N} ||f(x_i; \theta) - y_i||_{l_{CE}} + \lambda \|\theta\|_l \tag{2.2}$$

Where λ is the penalty factor and $\|.\|_l$ normalizes the magnitudes and range of the weights. Using the regularization term, the optimization decreases the training loss as well as the range of the weights.

To optimize this objective function Eq.5.1, there are many useful gradient-based optimization methods that can be employed. Here for simplicity, we consider the widely used stochastic gradient descent (SGD) to update the trainable weights θ:

$$\theta \leftarrow \theta - \alpha \nabla_\theta L(x, y; \theta) \tag{2.3}$$

Where α is the learning rate or step size, ∇_θ is the differentiate operator w.r.t θ.

2.2 Overftting

Overfitting is a notorious problem in deep neural networks. It manifests itself as a decrease in training loss to a certain value, followed by an increase in the test loss. Continuing to train an overfitted network eventually leads to almost zero train loss and high test loss, which the model is memorizing the training dataset.

There are many efforts to solve the overfitting problem, includes but not limited to: regularization,[12] early stopping,[13] dropout,[14] adversarial training[15,16] and data augmentation.[17] These methods are designed to avoid or make it harder to memorize the train set by setting restrictions to the trainable weights or inputs. Although they have already worked well in practice, setting penalty factors to the restrictions highly correlated to the training process, e.g., small l_2 penalties still lead to zero train loss, and adversarial samples can hurt generalization.[18] However, a more serious fact is that the concept of avoiding memorization by adding randomness may be challenged because even if the dataset and label are totally randomly corrupted, the MLP model could still achieve zero train loss.[19]

2.3 Methods for Solving Overfitting problem.

Overfitting can be mitigated through various strategies in Fig 2.1., including data augmentation, model architecture design, and the use of appropriate optimizers. Below, each of these methods is discussed in detail:

- **Data augmentation**: Data augmentation involves artificially increasing the size of

the training dataset by applying transformations to the original data. These transformations introduce variations that help the model generalize better.

- User-centric Interface: Cloud interfaces are location independent and can be accessed via well-established interfaces such as Web services and Internet browsers.

- **Model Architecture**: Designing an appropriate model architecture can significantly impact a model's ability to generalize. Choosing the right size, depth, and complexity of the model is critical.

- **Optimizers and Training Strategies**: The choice of optimizer and associated hyperparameters (like learning rate) plays a crucial role in controlling overfitting.

- **Loss function and Regularization**: Penalty-based regularization methods are a common approach in deep learning to prevent overfitting by adding a penalty term to the loss function.

Overfitting is a common challenge in deep learning, but it can be effectively reduced through strategies like data augmentation, careful model design, and the use of appropriate optimizers. Data augmentation increases dataset diversity, making the model more robust to variations. Model architecture techniques, such as dropout and weight regularization, control the model's capacity and encourage generalization. Finally, optimizers like AdamW, along with learning rate schedules, help achieve smooth and stable convergence. By employing a combination of these techniques, deep learning models can achieve better generalization, leading to improved performance on unseen data.

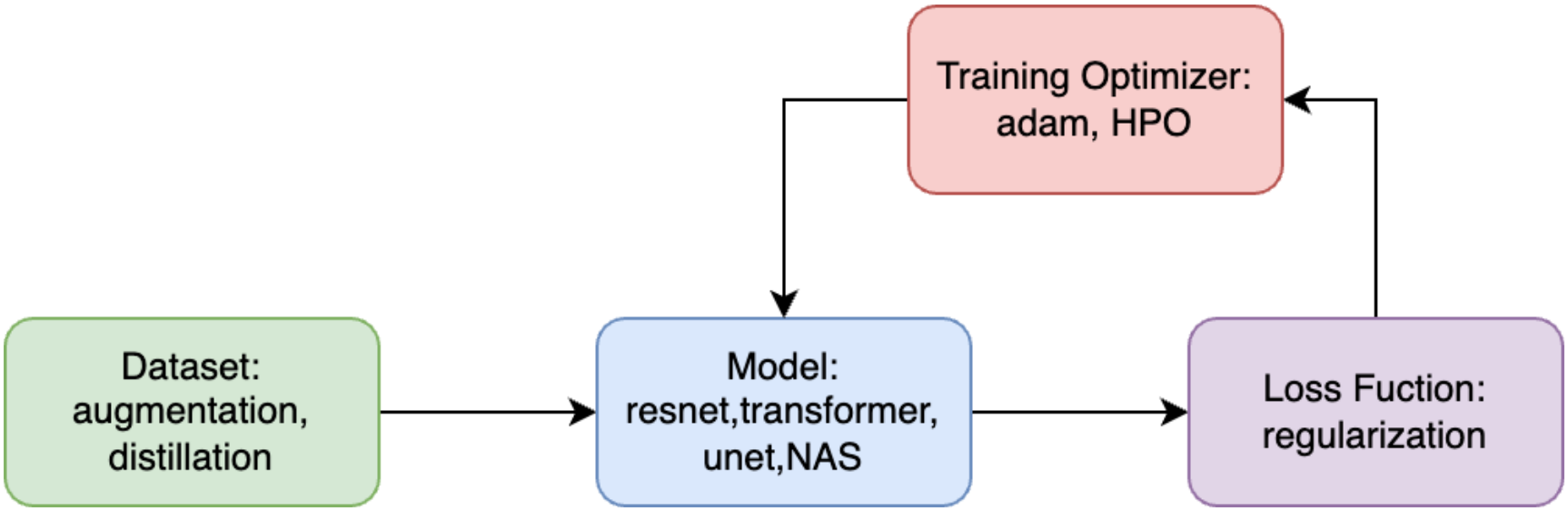


Figure 2.1: Methods for solving overfitting during training deep learning model(Arrow represent tensor's flow direction)

2.4 Loss Function and Regularization

2.4.1 Penalty Constrain

Penalty-based regularization methods are a common approach in deep learning to prevent overfitting by adding a penalty term to the loss function. This penalty term discourages complex models with large weights. L1 regularization adds the absolute value of the weights to the loss, encouraging sparsity and feature selection. L2 regularization adds the square of the weights, leading to smaller, more distributed weights. By carefully balancing the strength of the penalty term, we can improve the generalization performance of our models.

$$\text{L1 Regularization:} \quad \mathcal{L} = \mathcal{L}_{\text{original}} + \lambda \sum_{i=1}^{n} |w_i| \tag{2.4}$$

$$\text{L2 Regularization:} \quad \mathcal{L} = \mathcal{L}_{\text{original}} + \frac{\lambda}{2} \sum_{i=1}^{n} w_i^2 \tag{2.5}$$

In these formulas:

$\mathcal{L}$ is the overall loss function. $\mathcal{L}_{\text{original}}$ is the original loss function (e.g., cross-entropy loss, mean squared error). λ is the regularization strength, a hyperparameter that controls the impact of the penalty term. w_i is the i-th weight in the model. n is the total number of weights. L_1 regularization encourages sparsity by driving many weights to zero, effectively performing feature selection. L_2 regularization, on the other hand, prevents weights from growing too large, leading to more distributed weight distributions and improved generalization. These properties are shown in 2.2. L_1: The diamond-shaped contours of L_1 regularization push the solution towards the axes, encouraging sparsity. L_2: The circular contours of L_2 regularization push the solution towards the origin, encouraging smaller weights.

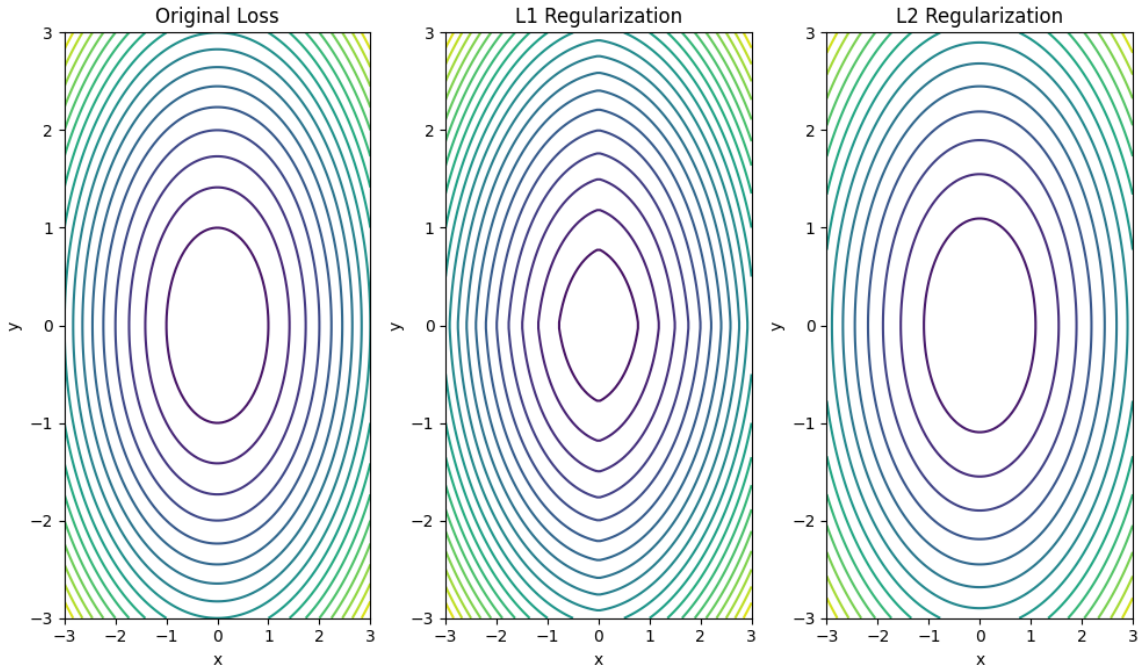


Figure 2.2: L_1: The diamond-shaped contours of L_1 regularization push the solution towards the axes, encouraging sparsity. L_2: The circular contours of L_2 regularization push the solution towards the origin, encouraging smaller weights.

2.5 Data Augmentaion

Data augmentation is a powerful technique used to artificially increase the size and diversity of a dataset, thereby improving the performance and generalizability of deep learning models. By applying various transformations to existing data, we can generate new, synthetic data points that are similar to the original data but have subtle differences.

2.5.1 Image Data Augmentation

Here are some common image data augmentation techniques in both Geometric Transformations style and Color Space Transformations style:

- **Rotation**: Rotating images by different angles.
- **Flipping**: Flipping images horizontally or vertically.
- **Scaling**: Resizing images to different scales.

- **Cropping**: Randomly cropping portions of images.
- **Translation**: Shifting images horizontally or vertically.
- **Color Jittering**: Randomly adjusting the brightness, contrast, saturation, and hue of images.
- **Color Shifting**: Shifting the overall color balance of images.
- **Grayscale Conversion**: Converting images to grayscale. Noise Addition:
- **Gaussian Noise**: Adding random Gaussian noise to images.
- **Salt and Pepper Noise**: Randomly adding black and white pixels to images.

Geometric data augmentation works by introducing variations in the spatial arrangement of data points while preserving the underlying structure and meaning. This technique helps deep learning models become more robust and generalize better to unseen data. The reason why it works can be concluded: First, Geometric transformations create new, synthetic data points from existing ones, effectively expanding the dataset. This helps the mode exposed to a wider range of variations in object orientation, scale, and perspective, enhancing its ability to recognize patterns under different conditions. Second, Geometric transformations let the model focus on Invariant Features. Geometric transformations force the model to learn features invariant to spatial transformations, such as shape, texture, and color. As a result, the data augmentation improved feature extraction, which made the model better at extracting meaningful features from data, even in the presence of noise or distortions.

2.5.2 Text Data Augmentation

Text data augmentation is a technique used to artificially increase the size and diversity of a text dataset. This is crucial for training robust natural language processing (NLP) models, especially when dealing with limited amounts of data.

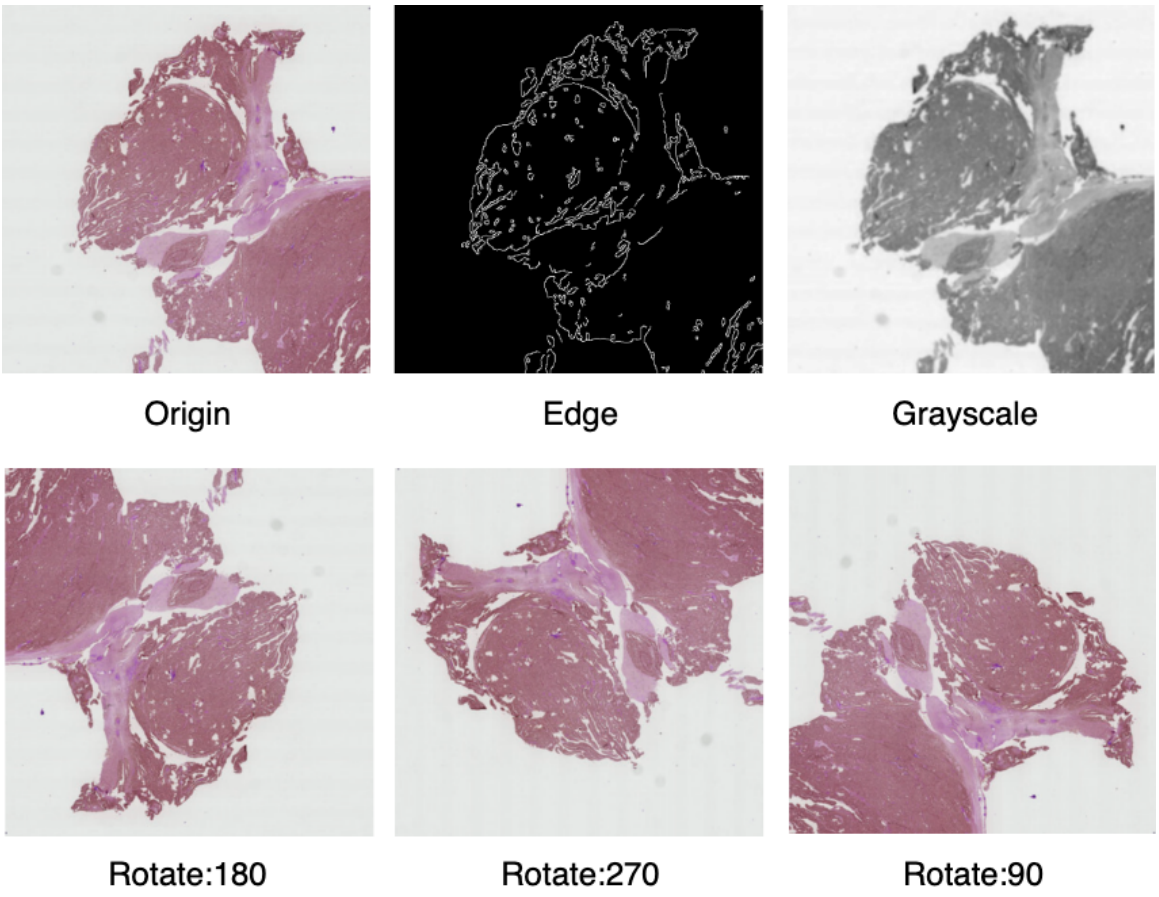


Figure 2.3: L_1: The diamond-shaped contours of L_1 regularization push the solution towards the axes, encouraging sparsity. L_2: The circular contours of L_2 regularization push the solution towards the origin, encouraging smaller weights.

By applying various transformations to existing text data, we can generate new, synthetic data points that are similar to the original data but have subtle differences. This helps the model learn to generalize better and become more resistant to overfitting. These common text data augmentation techniques include:

- **Synonym Replacement**: Replacing words with their synonyms. For example, ”good” could be replaced with ”excellent” or ”great.”
- **Random Insertion**: Inserting random words into the text. This can be done by randomly selecting a word and inserting it at a random position in the sentence.
- **Random Deletion**: Removing random words from the text. This can help the model become more robust to noise and missing information.
- **Random Swapping**: Swapping the positions of two random words in a sentence.
- **Back-Translation**: Translating the text into another language and then translating it back to the original language. This can introduce new variations and nuances in the text.
- **Text Generation**: Using language models like GPT-3 to generate new text that is similar to the original text. This can be helpful for generating more diverse and creative text data.

By applying these techniques, we can significantly increase the size and diversity of our text dataset, leading to improved performance of our NLP models.

2.6 Model Structure

Model structure reduces overfitting by limiting a model's capacity to memorize the training data and encouraging it to learn generalizable patterns instead. Overfitting typically occurs

when a model is excessively flexible, allowing it to capture noise and irrelevant details in the data. By introducing structural constraints, such as fewer layers or units, the model's expressiveness is restricted, making it less likely to overfit. Architectural choices like convolutional layers in CNNs inherently reduce overfitting by sharing weights across spatial dimensions, focusing on local patterns rather than memorizing entire images. Additionally, techniques like residual connections in deep networks help maintain the flow of gradients, enabling deeper models to generalize better without degradation in performance. Structural components such as dropout randomly deactivate neurons during training, effectively preventing co-adaptation and encouraging the model to rely on diverse feature representations. Batch normalization further stabilizes training by normalizing inputs to each layer, reducing internal covariate shifts and improving generalization. By carefully designing the model architecture to include these elements, overfitting is minimized as the model is guided to focus on essential features rather than noise, leading to better performance on unseen data.

Multi-Layer Perceptron (MLP) A Multi-Layer Perceptron (MLP) is a type of feed-forward neural network composed of multiple layers of neurons, where each layer is fully connected to the subsequent layer. The network consists of an input layer, one or more hidden layers, and an output layer. Each neuron in the MLP applies a linear transformation followed by a non-linear activation function. Mathematically, the operation in a single layer can be expressed as:

$$h^{(l)} = f\left(W^{(l)}h^{(l-1)} + b^{(l)}\right)$$

where $h^{(l-1)}$ is the input from the previous layer, $W^{(l)}$ is the weight matrix, $b^{(l)}$ is the bias vector, and f is the activation function (e.g., ReLU or sigmoid). MLPs are commonly used for tasks like classification and regression but struggle with high-dimensional data like images.

Convolutional Neural Network (CNN) A Convolutional Neural Network (CNN) is

specifically designed for processing grid-like data, such as images. Unlike MLPs, CNNs use convolutional layers to automatically extract spatial features. A convolution operation applies a filter (kernel) over the input to produce feature maps, capturing local patterns. The mathematical operation for a convolution can be expressed as:

$$y_{i,j} = \sum_{m,n} x_{i+m,j+n} \cdot k_{m,n}$$

where x is the input, k is the kernel, and y is the output feature map. CNNs also use pooling layers to downsample feature maps, reducing computational complexity and preventing overfitting. The combination of convolution, pooling, and fully connected layers makes CNNs highly effective for image classification and object detection.

Residual Network (ResNet) Residual Networks (ResNet) address the problem of vanishing gradients in deep networks by introducing residual connections, or skip connections, that bypass one or more layers. The core idea is to allow the network to learn residual mappings instead of directly learning the desired function. A residual block can be expressed as:

$$y = f(x) + x$$

where $f(x)$ represents the output of the intermediate layers and x is the input to the block. This design enables easier training of very deep networks by preserving gradient flow. ResNets are widely used in image classification and have achieved state-of-the-art performance on many benchmarks.

Vision Transformer (ViT) The Vision Transformer (ViT) applies the transformer architecture, originally developed for natural language processing, to image data. ViT splits an image into a sequence of patches, each treated as a token, and feeds them into a standard transformer. Each patch is flattened and embedded using a linear projection:

$$z_0 = [x_1 P; x_2 P; \ldots; x_N P] + E_{\text{pos}}$$

where x_i represents the patches, P is the projection matrix, and E_{pos} is the positional encoding. The transformer then processes these embeddings through multi-head self-attention and feedforward layers. ViT achieves competitive performance on image classification tasks and has gained popularity for its flexibility and scalability.

The above characteristics make Big Data difficult to process on a single system that highlights the criticality of high performance computational infrastructure. Digital media, pervasive sensing, web authoring, mobile computing, scientific and medical instruments, physical simulations and virtual worlds are all delivering vast new datasets relating to every aspect of our lives. Although there are several sources of data, this dissertation focuses on Big Data coming from scientific experiments such as bioinformatics, and environmental, astronomic field of areas.

2.7 Optimizer

An optimizer in deep learning adjusts a model's parameters (weights and biases) to minimize the loss function, which quantifies how well the model's predictions match the true labels. The process begins by initializing the model's parameters, often with random values. During each iteration (or epoch) of training, the model makes predictions on input data, and the loss function calculates the error. The optimizer then computes the gradient of the loss with respect to each parameter using backpropagation. These gradients indicate the direction and magnitude of changes needed to reduce the error. Based on the optimizer's update rule—such as Stochastic Gradient Descent (SGD), Adam, or AdamW—the parameters are adjusted accordingly. Learning rates, momentum, and adaptive techniques play key roles in controlling how much the weights are updated. The optimizer ensures efficient convergence by balancing speed and stability, helping the model reach an optimal or near-optimal set of parameters that generalize well to unseen data. Proper selection and tuning of the optimizer significantly impact model performance and training efficiency.

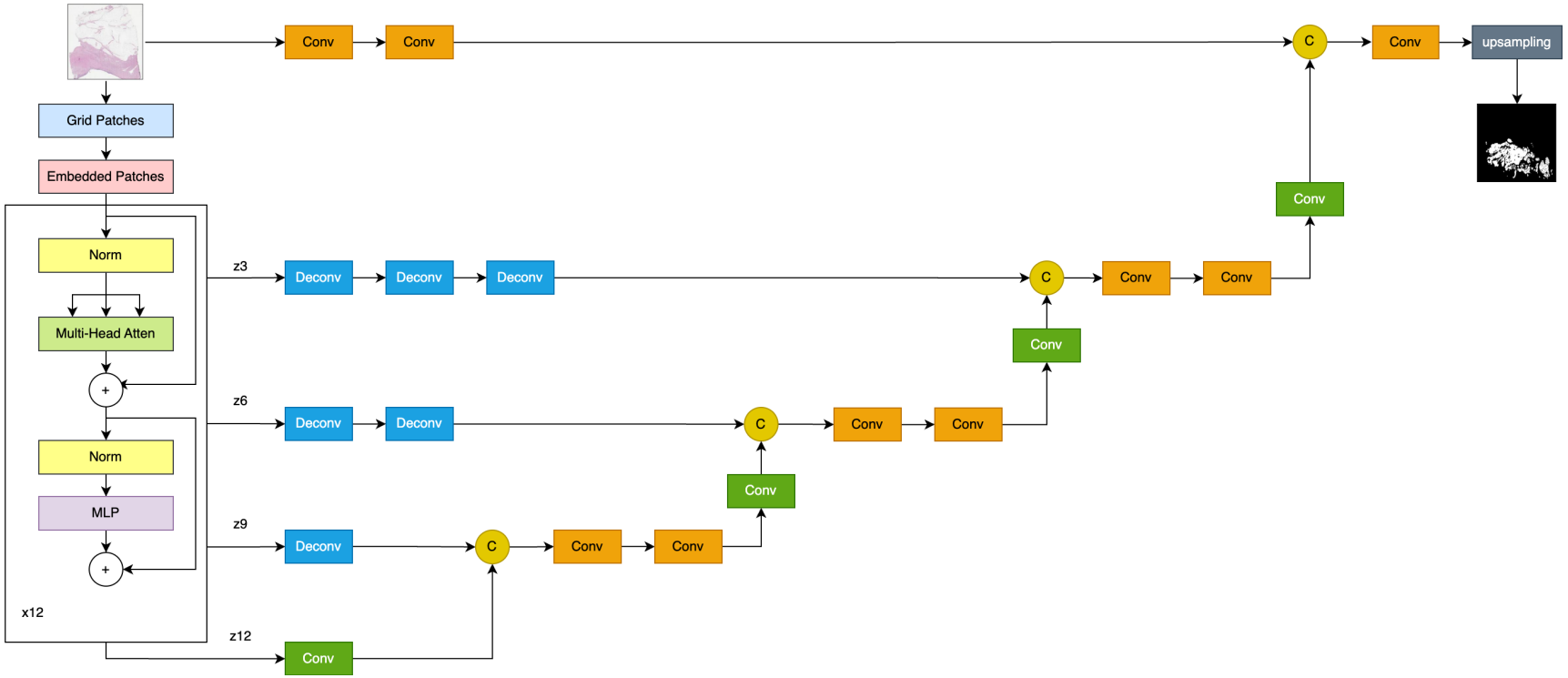


Figure 2.4: The UNETR model used for Image Segmentation. The downsampling encoder is a ViT model, the upsampling decoder is a convolution neural network, and the skip connection is an idea from a residual network. Model structure greatly reduced overfitting in vision tasks.

Stochastic Gradient Descent (SGD) is a widely used optimization algorithm for updating the parameters of a machine learning model in order to minimize a given loss function. It is a variation of the standard Gradient Descent algorithm but introduces randomness for efficiency and scalability. Here θ_t represents the parameters at iteration t. η is the learning rate, a hyperparameter that controls the step size. $\nabla_\theta J(\theta_t)$ is the gradient of the loss function $J(\theta)$ with respect to the parameters.

$$\theta_{t+1} = \theta_t - \eta \nabla_\theta J(\theta_t) \tag{2.6}$$

Momentum is a skill of the standard SGD algorithm that helps accelerate convergence by adding a velocity term. This term accumulates a fraction of the previous gradients, smoothing updates and reducing oscillations. Here v_t is the velocity term at iteration t, representing the exponentially decaying average of past gradients. γ is the momentum coefficient, typically $0 \leq \gamma < 1$. η is the learning rate. $\nabla_\theta J(\theta_t)$ is the gradient of the loss function with respect to the parameters θ. θ_t represents the model parameters at iteration t. The term v_t accumulates gradients over time, effectively acting as a "memory" of past updates. By combining past gradients, momentum reduces oscillations, especially in regions where gradients fluctuate. It helps the optimizer build up speed in directions of consistent gradients, leading to faster convergence.

$$v_{t+1} = \gamma v_t + \eta \nabla_\theta J(\theta_t) \tag{2.7}$$

$$\theta_{t+1} = \theta_t - v_{t+1} \tag{2.8}$$

Adam (Adaptive Moment Estimation) is an optimization algorithm that combines the benefits of both momentum and adaptive learning rates. It maintains running estimates of

both the first moment (mean) and the second moment (uncentered variance) of the gradients, and uses these estimates to adapt the learning rate for each parameter. where m_t is the first moment estimate (mean of gradients). v_t is the second moment estimate (uncentered variance of gradients). β_1 and β_2 are exponential decay rates for the moment estimates, typically $\beta_1 = 0.9$ and $\beta_2 = 0.999$. $\hat{m}_{t+1}$ and $\hat{v}_{t+1}$ are the bias-corrected moment estimates.η is the learning rate. ϵ is a small constant added for numerical stability, typically $\epsilon = 10^{-8}$. $\nabla_\theta J(\theta_t)$ is the gradient of the loss function with respect to the parameters θ. Adam combines the benefits of momentum and adaptive learning rates, making it a more robust and efficient optimizer compared to SGD with Momentum. While SGD with Momentum can perform well with proper tuning, Adam's adaptive updates and bias correction often lead to faster convergence and better performance across a wide range of tasks.

$$m_{t+1} = \beta_1 m_t + (1 - \beta_1)\nabla_\theta J(\theta_t) \tag{2.9}$$

$$v_{t+1} = \beta_2 v_t + (1 - \beta_2)(\nabla_\theta J(\theta_t))^2 \tag{2.10}$$

$$\hat{m}_{t+1} = \frac{m_{t+1}}{1 - \beta_1^{t+1}} \tag{2.11}$$

$$\hat{v}_{t+1} = \frac{v_{t+1}}{1 - \beta_2^{t+1}} \tag{2.12}$$

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_{t+1}} + \epsilon}\hat{m}_{t+1} \tag{2.13}$$

AdamW (Adam with Weight Decay) is an optimization algorithm that improves upon the standard Adam optimizer by decoupling weight decay from the gradient-based update step. This modification addresses issues related to regularization and generalization in deep learning models.Here m_t is the first moment estimate (mean of gradients). v_t is the second moment estimate (uncentered variance of gradients). β_1 and β_2 are exponential decay rates for the moment estimates, typically $\beta_1 = 0.9$ and $\beta_2 = 0.999$. $\hat{m}_{t+1}$ and $\hat{v}_{t+1}$ are the bias-corrected moment estimates. η is the learning rate. ϵ is a small constant added for

numerical stability, typically $\epsilon = 10^{-8}$. λ is the weight decay coefficient, controlling the magnitude of the regularization term. $\nabla_\theta J(\theta_t)$ is the gradient of the loss function with respect to the parameters θ. Adam and AdamW are both adaptive optimization algorithms that utilize first and second moment estimates to adjust learning rates for each parameter. The key improvement in AdamW is the way it handles regularization through weight decay, which leads to better generalization performance.

$$m_{t+1} = \beta_1 m_t + (1 - \beta_1)\nabla_\theta J(\theta_t) \tag{2.14}$$

$$v_{t+1} = \beta_2 v_t + (1 - \beta_2)(\nabla_\theta J(\theta_t))^2 \tag{2.15}$$

$$\hat{m}_{t+1} = \frac{m_{t+1}}{1 - \beta_1^{t+1}} \tag{2.16}$$

$$\hat{v}_{t+1} = \frac{v_{t+1}}{1 - \beta_2^{t+1}} \tag{2.17}$$

$$\theta_{t+1} = \theta_t - \eta \left(\frac{\hat{m}_{t+1}}{\sqrt{\hat{v}_{t+1}} + \epsilon} + \lambda \theta_t \right) \tag{2.18}$$

Chapter 3

Regularization by Validation

3.1 Introduction

Studying training trajectories and loss surfaces is a key focus in the deep learning community.[30–32] High-dimensional data generated during training is often discarded under standard procedures. However, two approaches utilize this high-dimensional information. The first explores generalization and visualizes the loss surface around local minima in random directions.[30] Researchers also examine properties of the loss landscape, such as sharpness,[31] smoothness, and connectivity,[32] to understand these surfaces better. The second approach accelerates training or finds generalized solutions by ensembling weights[32,33] and modifying optimization parameters.[31,34] [1].

In most cases, performance improvements are achieved through data augmentation, hyperparameter tuning,[35] or manually and automatically designing complex networks.[36,37] These methods typically discard training trajectory information, focusing instead on weight updates to enhance accuracy incrementally. Thus, we explored leveraging past trajectories to enhance training outcomes.

[1]This chapter was published as Zhang E, Wahib M, Zhong R, et al. Learning from the Past Training Trajectories: Regularization by Validation[J]. Journal of Advanced Computational Intelligence and Intelligent Informatics, 2024, 28(1): 67-78. doi:10.20965/jaciii.2024.p0067.

In this work, we propose a supervisor-students training framework (REVAL) that utilizes past training trajectories to improve student generalization on image classification tasks. However, REVAL faces challenges such as the supervisor's dimensional complexity, ensuring training consistency for the supervisor, and conducting experiments on large-scale models. To address these, we tested REVAL on smaller datasets and models while extending our work to include experiments with larger models (ResNet-56) and providing a consistent discussion about the supervisor. Additionally, through visualization, we demonstrate why supervisor-generated gradients are more effective for converging to superior solutions.

Our contributions include the following:

- We introduce REVAL, a student-supervisor framework that utilizes past training trajectories, including weights and validation losses, as evaluation metrics.
- We demonstrate the supervisor model's ability to predict validation losses from student weights effectively. To enhance practicality, we propose three optimizations under the iREVAL method: compressing the supervisor's weights, implementing an online update schema to mitigate prediction degradation from students, and applying normalized gradients to address gradient vanishing issues.
- Using *Azuma's inequality*, we establish the consistency of supervisor training. Building on this insight, we designed iREVAL to expedite training and reduce memory requirements.
- iREVAL significantly improved training across various architectures on multiple benchmarks. For instance, a ResNet-based student trained with iREVAL achieved a 1.07% test accuracy increase on CIFAR-10 and a 1.73% improvement on CIFAR-100.
- To explain the supervisor's impact, we visualized the loss landscape using gradients

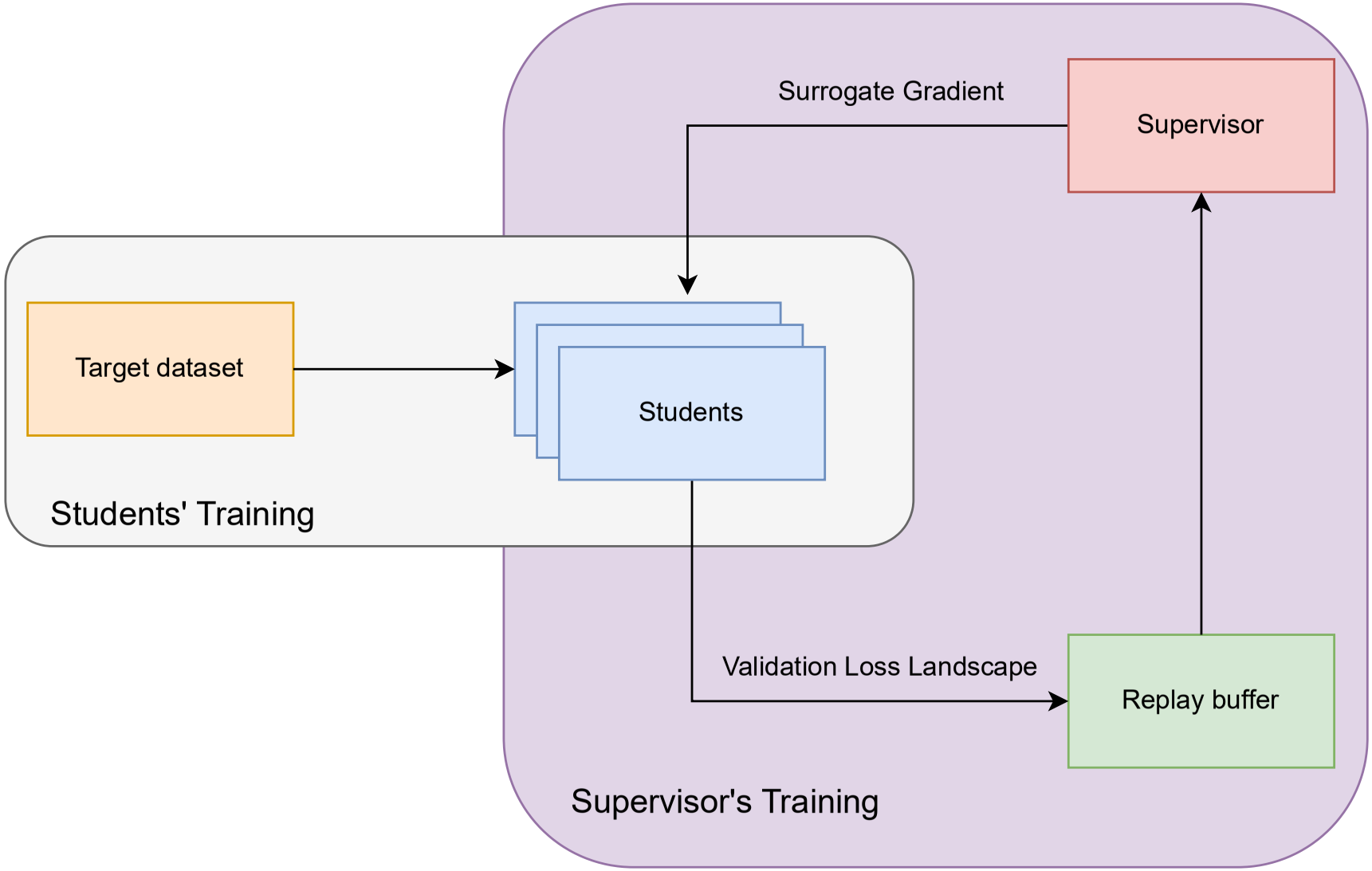


Figure 3.1: Overview of the proposed method: REgularization by VALidation(REVAL).

from both training and supervision. The regional supervision produced smoother trajectories, elucidating the observed generalization improvement.

- To enhance supervisor prediction accuracy, we normalized its inputs. This adjustment improved robustness, particularly in the later stages of student training.

3.2 Related works

In this study, our objective was to explore the validation loss landscape of deep learning models and leverage the accumulated knowledge to enhance the training process. Specifically, we aim to improve model generalization by analyzing the trajectory of weight updates during training.

Overfitting is a persistent issue in training deep neural networks. It occurs when training loss decreases to a low value but test loss begins to increase. Continued training of an overfitted model results in nearly zero training loss while test loss remains high, as the network essentially memorizes the training data instead of learning general patterns.

Attempts to mitigate overfitting have been a cornerstone of deep learning research, leading to approaches like regularization,[12] early stopping,[13] dropout,[14] adversarial training,[15,16] and data augmentation.[17] These strategies impose constraints on the model, such as penalties on weights, architectural changes, or transformations on input data, to hinder overfitting. While effective, these methods rely heavily on hyperparameters that interact intricately with the training dynamics. For instance, small penalties can still lead to nearly zero training loss, and adversarial samples may even degrade generalization.[18] Moreover, studies such as[19] challenge the assumption that randomness prevents memorization, showing that even models trained on entirely corrupted datasets can achieve zero training loss.

An important question posed in[20] is whether achieving zero training loss is necessary after eliminating training errors. Zero training loss not only exacerbates overfitting but also fails to provide gradients strong enough to escape local minima. To address this,

the authors proposed a novel approach: allowing gradient ascent when training loss drops below a threshold. By alternating between descent and ascent, the model avoids becoming trapped in suboptimal solutions.

Building on this question, we argue that zero training loss itself is not inherently problematic. Rather, the issue lies in the gradients produced during optimization at low training losses. Through visualizations (Sec. 3.5.2), we demonstrate why supervisor-provided gradients are more effective in guiding models toward better solutions.

Unlike traditional backpropagation, which uses training samples to determine optimization directions, we hypothesize that an alternative approach can guide models toward reducing validation or test loss. Inspired by,[20] we propose the REgularization by VALidation (REVAL) framework. Instead of directly training on validation data, REVAL employs a supervisor network to learn the relationship between student weights and corresponding validation losses. By leveraging past training trajectories, the supervisor provides guidance for minimizing test losses, even when training loss approaches zero.

Our approach draws conceptual parallels with Q-learning,[38,39] where actions and states are used to predict future rewards. Similarly, we treat the loss landscape as an environment and validation losses as optimization costs, applying dynamic programming principles to navigate the surface effectively.

REVAL utilizes a student-supervisor framework distinct from similar dual-model systems like knowledge distillation (KD).[40] In KD, a large teacher network guides smaller student models on identical tasks. Conversely, REVAL's supervisor and student perform different tasks: the supervisor analyzes weight changes and validation losses, guiding the student to avoid suboptimal minima. While KD focuses on model size differences for the same task, REVAL leverages task differences to improve training outcomes.

By guiding student training based on past trajectories, REVAL shares conceptual ground with transfer learning, where knowledge from a source domain aids learning in a target domain. However, REVAL operates within a single dataset, distinguishing it from standard

transfer learning tasks that involve differing datasets or distributions. Although transfer learning exploits structural similarities in data, REVAL hypothesizes analogous structures within loss landscapes as a promising area for future exploration.

Finally, REVAL draws inspiration from recent advancements in neural architecture search (NAS),[37] where reinforcement learning identifies optimal network designs. Unlike NAS, which focuses on architecture, REVAL treats parameter optimization as a regression task. By aggregating insights hidden within validation loss dynamics, the supervisor enhances subsequent student training, providing a robust method for improving generalization in deep learning.

3.3 Methodology

In this section, we present the foundational settings employed for training neural network models and introduce an innovative regularization strategy leveraging predictions derived from the validation loss landscape.

3.3.1 Preliminaries

We utilized standard settings for training neural networks in image classification tasks. Consider an image dataset D consisting of input samples $x \in \mathbb{R}^d$ and corresponding one-hot encoded labels $y \in \mathbb{R}^K$, where K denotes the total number of classes. Let the neural network model be represented as $f : \mathbb{R}^d \to \mathbb{R}^K$, parameterized by trainable weights θ. For each data point $(x, y) \in D$, the predicted output is $\tilde{y} = f(x; \theta)$. Define a loss function $l : \mathbb{R}^K \times \mathbb{R}^K \to \mathbb{R}$, typically using the softmax cross-entropy loss l_{CE} for classification tasks. The optimization problem can be formulated as follows:

$$\min_{\theta} \quad L(x, y) = \frac{1}{N} \sum_{n=1}^{N} l_{CE}(f(x_n; \theta), y_n), \tag{3.1}$$

where N represents the number of training samples in D, and L is the overall training loss.

Overfitting, evidenced by a disparity between training and test losses, arises due to factors such as limited dataset size, poor data quality, or suboptimal model design. Regularization techniques, including weight decay,[12,28,29] are widely employed to mitigate overfitting. The objective function with regularization is modified as follows:

$$\min_{\theta} \quad L(x, y) = \frac{1}{N} \sum_{n=1}^{N} l_{CE}(f(x_n; \theta), y_n) + \lambda \|\theta\|_l, \tag{3.2}$$

where λ is a regularization factor and $\| \cdot \|_l$ represents the weight norm. Regularization restricts weight magnitudes, reducing overfitting.

To solve Equation 3.2, gradient-based optimization techniques are frequently applied. For simplicity, we consider stochastic gradient descent (SGD):

$$\theta \leftarrow \theta - \alpha \nabla_{\theta} L(x, y; \theta), \tag{3.3}$$

where α is the learning rate, and ∇_{θ} denotes the gradient with respect to θ.

3.3.2 Regularization by Validation

Overfitting occurs when a model memorizes training data instead of generalizing. As training progresses, overfitting becomes evident when validation loss begins to increase. Strategies such as early stopping,[13] dropout,[14] data augmentation,[17] and adversarial training[15,16]

have been developed to address this issue.

We propose a novel framework to mitigate overfitting, drawing an analogy to a student learning under supervision. In this framework, the *student*, represented as $f(x;\theta)$, learns from a dataset using the loss function $L_{st}(x, y; \theta)$, optimized using Equation 4.3.

Additionally, we introduce a *supervisor* network, denoted $s(\theta;\phi) : \mathbb{R}^{d_\theta} \to \mathbb{R}$, where d_θ is the dimensionality of the student's parameters. The supervisor provides surrogate validation loss gradients to guide the student towards empirically optimal minima. The supervisor is trained using the following mean squared error (MSE) loss:

$$\min_{\phi} \quad L_{sp}(\theta;\phi) = \frac{1}{M}\sum_{m=1}^{M}\big(s(\theta_m;\phi) - v_m\big)^2, \tag{3.4}$$

where $s(\theta;\phi)$ approximates the validation loss v_m, and M represents the number of sampled trajectories. During training, student weights θ and corresponding validation losses v form an experience dataset D_e, used to optimize the supervisor parameters via Equation 4.3.

The proposed REVAL algorithm integrates the student and supervisor objectives into a bi-level optimization problem:

$$\begin{aligned} &\min_{\phi} \quad L_{sp}(\theta;\phi), \\ &\min_{\theta} \quad L_{st}(x;\theta) + \lambda L_{sp}(\theta;\phi), \end{aligned} \tag{3.5}$$

where λ adjusts the influence of the supervisor's gradient. In this formulation, the supervisor replaces traditional norm-based regularization methods (e.g., ℓ_2 norm). For instance, setting $s(\theta) = \|\theta\|_2$ recovers conventional ℓ_2 regularization, making REVAL a generalization of existing frameworks.

Under this paradigm, the weight update for the student model becomes:

$$\theta \leftarrow \theta - \alpha \nabla_\theta L_{st}(x;\theta) - \alpha\lambda \nabla_\theta L_{sp}(\theta;\phi). \tag{3.6}$$

Here, the first gradient term minimizes the training loss, while the second term, derived from the supervisor, encourages convergence towards optimal solutions informed by historical trajectories.

3.3.3 REVAL Algorithm

We provide a high-level view of the REVAL algorithm in Algorithm 1. The REVAL algorithm has two main parts: warm-up and main loop. First, we initialize N students to create a candidate pool P. Then for all of the students in $P = \{f_i | i = 1...N\}$, we train them in a parallel fashion (i.e., an ensemble) and collect their trajectories of weights θ_i and validation losses v_i for each n epochs. Here n denotes the frequency of sampling the (θ, v) pairs on the validation loss surface.

It is noteworthy that these fully trained N students followed a general optimization framework. Therefore, we used these N students as the baseline in the later experiments for comparison with our proposed method.

The purpose of the warm-up step was to collect baseline information for the supervisor, with the expectation of later being able to achieve *improvement* over the general training framework. After the supervisor $s(\theta;\phi)$ is warmed up, we can start the main iteration of REVAL.

The main iteration loop is similar to the warm-up stage, with the difference that the training of the students is also guided by the supervisor (as shown in Eq. 4.5). After each main iteration, the supervisor is also updated using the D_e.

Algorithm 1 Regularization by validation

Require: $D_t, D_e, f(x;\theta), s(\theta;\phi), N, T$
Initialize N students and add to pool $P = \{f_i | i = 1...N\}$
for each $f_i \in P$ **do** ▷ Warm-up Step
    Train f_i in Eq. 5.1 on the D_t for n epochs.
    Evaluate f_i on validation set and obtain v_i.
    $D_e = D_e \cup \{\theta_i, v_i\}$
end for
Initialize and train supervisor in Eq. 4.4 $s(\theta;\phi)$ on D_e.
for $t \leftarrow 1$ **to** T **do** ▷ Main Step
    Initialize N students and add to pool P:
    for each $f_i \in P$ **do**
        Train f_i in Eq. 4.5 on the D_t for n epochs.
        Evaluate f_i on validation set and obtain v_i.
        $D_e = D_e \cup \{\theta_i, v_i\}$
    end for
    Update supervisor $s(\theta;\phi)$ on D_e.
end for

3.3.4 Challenges and Improvements

The fundamental concept behind REVAL is straightforward, yet ensuring its strategic effectiveness involves addressing numerous challenges. In,[37] the validation loss estimator s was shown to model the performance of different network architectures sampled from the structure search space. Unlike this, REVAL employs s to predict validation loss directly based on the weights of the model. Specifically, the estimators s in[37] can be mathematically expressed as $s : R^{d_a} \rightarrow R$, whereas in our approach, the estimator operates as $s : R^{d_w} \rightarrow R$. Here, d_a denotes the dimension of the architecture space, and d_w represents the dimension of the weight space. Consequently, we reformulate the learning capability of the estimator in,[37] as demonstrated in Lemma 4.

Lemma 1. Let S represent a hypothesis class, L_N be the empirical loss, and L denote the expected loss. For any $\delta > 0$, with a probability of at least $1 - \delta$, the following inequality holds for all $s \in S$:

$$|L_N(s) - L(s)| < \sqrt{\frac{2(d + \ln \frac{2}{\delta})}{N}}$$

where d is Pollard's pseudo-dimension of the hypothesis class S, and N is the total number of training samples.

Lemma 4 establishes that the convergence rate of the error bound depends on the pseudo-dimension d of the hypothesis class and the sample size N. With a fixed d and an infinitely large N, the gap between empirical and expected losses asymptotes to a constant value. Although utilizing an infinite number of samples is infeasible in practical experiments, the lemma implies that reducing the input dimensionality of s and expediting student training can accelerate convergence toward the error bound.

3.3.4.1 Supervisor's Dimensional Complexity Challenge

For simplicity, assume both the student and supervisor models are multilayer perceptron (MLP) networks with identical structural configurations, particularly the number of units in each hidden layer. During REVAL-based training, the students use samples from the target dataset as input. Denoting the input dimensionality as d, the dimensionality of the student weights d_{st} can be expressed as:

$$d_{st} = d \cdot u_1 + \sum_{i=1}^{N} u_i \cdot (u_{i-1} + 1)$$

where N is the number of layers, u_i represents the number of units in the i-th layer, and $u_0 = d$ is the input dimensionality.

The supervisor $s(\theta; \phi)$, which takes the student weights θ as input, must process data of equivalent dimensionality:

$$d_{sp} = d_{st} \cdot u_1 + \sum_{i=1}^{N} u_i \cdot (u_{i-1} + 1).$$

For MLPs, the input dimension d is often significantly larger than $u_{i \neq 0}$. Consequently, the weights in the first layer typically dominate the overall weight count, leading to ap-

proximations $d_{st} \approx d \cdot u_1$ and $d_{sp} \approx d \cdot u_1^2$. Thus, the supervisor's dimensional complexity scales as $\Theta(u_1^2)$, which is u_1 times larger than the students' $\Theta(u_1)$. This issue, termed the *dimensional complexity challenge of the supervisor*, exacerbates with increasing dataset size and student model complexity.

To mitigate this challenge, the primary objective is to reduce the supervisor's input dimensionality while preserving its generalization capabilities and differentiation with respect to θ. A naive approach of discarding unrelated weights sacrifices differentiability. To balance these trade-offs, we propose an effective method of averaging weights across each layer:

$$V_i = \frac{1}{u_i} \sum_{j=1}^{u_i} W_{ij}$$

where W_{ij} denotes the j-th unit's weights in the i-th layer, u_i is the number of units in the i-th layer, and V_i represents the aggregated weights. By employing this technique, the supervisor's dimensionality d_{sp} can be approximated as $d \cdot u_1$, aligning closely with the student dimensionality d_{st}.

This aggregation is motivated by the observation that, during inference, weights within the same layer are largely independent and can be represented as points in a weight space. The aggregated weight V_i serves as a central representative of these points, enabling the supervisor to maintain efficiency without compromising predictive accuracy.

3.3.4.2 Supervisor's Failure

The updating rules for REVAL in Eq. 4.6 have two parts: the first part is the student's gradients $\nabla_\theta L_{st}(x, y)$ and the second part is the supervisor's gradients $\nabla_\theta L_{sp}(x, y)$. Because the supervisor $s(\theta; \phi)$ is only updated after each main iterator, this means during training the weights ϕ of the supervisor are fixed. Therefore, using the second component of the gradients $\nabla_\theta L_{sp}(x, y)$, the optimizer often causes the supervisor to quickly converge to the local minima.

In our experiments, we observed that the predicted validation loss decreased almost monotonically and converged at a certain constant. We call this problem *supervisor's failure* because it cannot generalize on the prediction task when training students and it manifests as students following adversarial directions through the supervisor's gradient.

To solve this problem, we set the supervisor to update itself to adapt to student training. Because we want to update the supervisor while training the students, the online objective function can be rewritten as

$$\min_{\phi} \quad L_{sp}(\theta, v) \tag{3.7}$$

$$\min_{\theta} \quad \left\{ L_{st}(x, y) + \lambda \min_{\phi} \quad L_{sp}(\theta; \phi) \right\} \tag{3.8}$$

Compared to the original REVAL's objective function Eq. 4.5, here we updated the supervisor during the student training. Once the samples are received from the training students, the supervisor can adapt to the online samples and avoid the loss of generalization to predict student loss.

3.3.4.3 Vanishing Gradients

The gradient provided by the supervisor was $\nabla_{\theta} s(\theta; \phi)$. If we assume that the supervisor uses an MLP model, we can expand the gradient using the *chain-rule* as follows

$$\frac{\partial s(\theta; \phi)}{\partial \theta} = \frac{\partial s(\theta; \phi)}{\partial f_n} \frac{\partial f_n}{\partial f_{n-1}} \frac{\partial f_{n-1}}{\partial f_{n-2}} \cdots \frac{\partial f_1}{\partial \theta}$$

where f_i is the i-th layer's output, and $\frac{\partial f_i}{\partial f_{i-1}}$ is the Jacobian matrix. This equation shows that the gradients provided by the supervisor are similar to the back-propagation gradient of the first layer. Multiplying with the Jacobian matrix exponentially decreases or increases the radius of spectra, and the gradient is observed to vanish or explode on many tasks.

During our experiment, in comparison with the gradients given by the students $\nabla_{\theta} f(x; \theta)$,

we found that the magnitude of the supervisors' gradients was small. Based on this observation, we propose the use of a normalized gradient descent to replace the vanished gradient in Eq.4.6. We write the updated rules as follows:

$$\theta \leftarrow \theta - \alpha \frac{\nabla_\theta L_{st}(x, y)}{||\nabla_\theta L_{st}(x, y)||} - \alpha\lambda \frac{\nabla_\theta L_{sp}(\theta; \phi)}{||\nabla_\theta L_{sp}(\theta; \phi)||} \tag{3.9}$$

Where $||.||$ is the norm operator that we used during the experiments (l_2 norm) and λ is the penalty factor that could be used to control the convergence rate of the supervisor.

3.4 Experiments

This section describes three categories of experiments designed to evaluate the proposed REVAL framework. First, we validate the effectiveness of using the supervisor to predict the validation loss based on the weights of student models. Second, we compare REVAL against standard stochastic gradient descent (SGD) training applied to MLP models across various datasets, including MNIST, CIFAR-10, and CIFAR-100. Finally, we explore how the enhancements detailed in Section 3.3.4 influence the overall training dynamics and performance of REVAL.

3.4.1 Datasets

The datasets utilized in our experiments are widely adopted benchmarks for image classification tasks. The MNIST dataset[41] consists of $60,000$ training examples and $10,000$ test examples of handwritten digits. CIFAR-10[42] and CIFAR-100[43] contain $50,000$ training images and $10,000$ test images, categorized into 10 and 100 classes, respectively. To ensure consistency, we shuffled the training and test datasets and performed an 80-20 split on the training set to allocate 20% for validation and 80% for training. During the training process, we recorded the validation losses and corresponding weights to build the dataset

required for training the supervisor.

3.4.2 Student and Supervisor Configurations

3.4.2.1 Student Models

Two types of student models were employed in our experiments. The first type consists of fully connected multilayer perceptron (MLP) models, each comprising four layers with units configured as $[128, 64, 32, 10]$ and activation functions $[ReLU, ReLU, ReLU, Softmax]$. These MLP models exclude batch normalization to simplify analysis. To evaluate the applicability of REVAL in realistic scenarios, the second type of student model is the ResNet-56 architecture,[44] a widely recognized convolutional neural network for image classification tasks.

The students were initialized using the $glorot$ initializer.[45] For training, we used the cross-entropy loss function and the standard SGD optimizer with a learning rate of 0.1. In experiments involving learning rate decay, the decay was applied at $(50\%, 75\%)$ of the total epochs, with a decay factor of 0.1. We deliberately chose naive SGD over adaptive optimizers like Adam to eliminate confounding effects from adaptive gradient mechanisms. The batch size was set to 128, and each student was trained for 300 epochs. For experiments incorporating data augmentation (DA), image processing followed the guidelines in.[44]

To efficiently collect samples for training the supervisor, validation losses were recorded at intervals defined by a validation gap. By adjusting this gap, we controlled the frequency of data collection from the loss surface. For our experiments, the validation gap was set to 100 iterations.

In the warmup phase, we trained 20 students using the standard training framework (Eq. 4.3). The resulting weights and validation losses formed the initial dataset D_e for supervisor training. During the main training phase, $N = 5$ students were trained in each iteration. Their weights and losses were used to update the supervisor, and the data was

subsequently added to D_e. The students trained during the warmup stage served as a baseline for comparison in subsequent experiments.

3.4.2.2 Supervisor Model

The supervisor model is a four-layer MLP with architecture similar to the student models but adapted for regression. Each layer comprises $[128, 64, 32, 1]$ units with activation functions $[ReLU, ReLU, ReLU, Linear]$. The weights were initialized using the $glorot$ initializer with parameters $(0, 0.01)$.

As the supervisor's task is to regress the validation loss, we used the mean squared error (MSE) as the loss function. The optimizer for the supervisor was an SGD optimizer with a learning rate of 0.01 and a batch size of 128.

During the warmup phase, the supervisor was trained for 100 epochs using the dataset D_e. In the main training phase, the supervisor was also trained for 100 epochs after every iteration, using N_n new students from the current iteration and N_o old students randomly sampled from D_e. This balance between new and old samples ensured consistent supervisor performance across iterations.

3.4.3 Results

In this section, we discuss the results, including the generalization of students obtained by baseline, REVAL, and iREVAL (improved REVAL). Next, we present the training curve of the supervisor. Finally, we discuss the challenges with REVAL training and the effects of these improvements.

3.4.3.1 REVAL's Training

Fig. 3.2 shows a naive comparison of the test losses for baseline, REVAL, and iREVAL on CIFAR-10 with the 4-layer MLP models and without data augmentation. The baseline student was overfitted after training for 10 epochs. REVAL obtains a more generalized

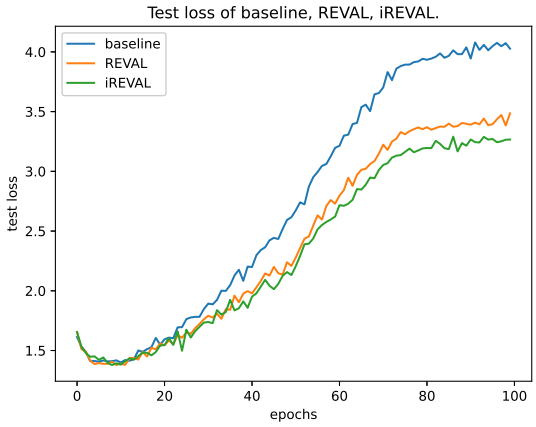


Figure 3.2: Test loss of student trained by baseline, REVAL, and iREVAL on the CIFAR-10 dataset with the 4-layer MLP models and without data augmentation.

solution yet still overfits after a long training period. Finally, iREVAL achieved the lowest test loss and reduced overfitting over a long training period. Note that the test-loss curve is not smooth compared with the baseline, because of the adversarial effects of the student and supervisor objectives.

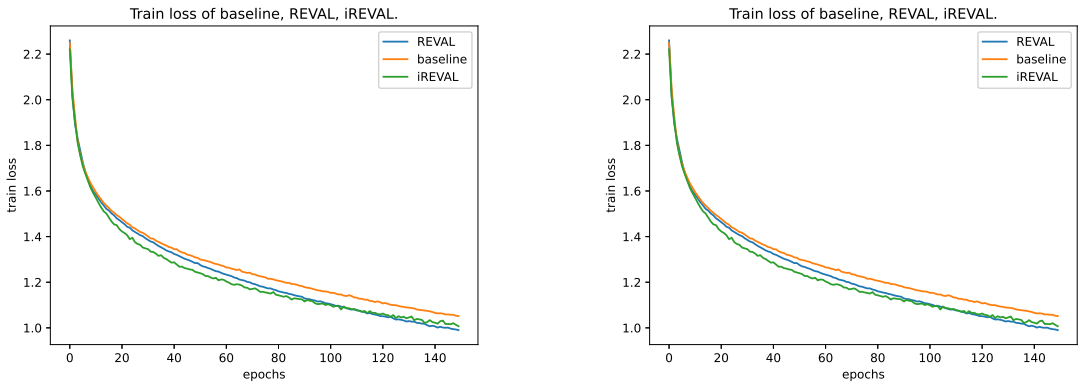


Figure 3.3: Training loss(a) and test accuracy(b) of the baseline, REVAL, and iREVAL on the CIFAR-10 dataset with the 4-layer MLP models.

To explain the training curve in detail, the training loss and test accuracy are in Fig. 3.3. From the figure, we can observe that because REVAL and iREVAL obtain lower validation losses (i.e., avoid overfitting), they result in lower training losses and higher test accuracies than the baseline. The results demonstrate the effectiveness of the improvements introduced

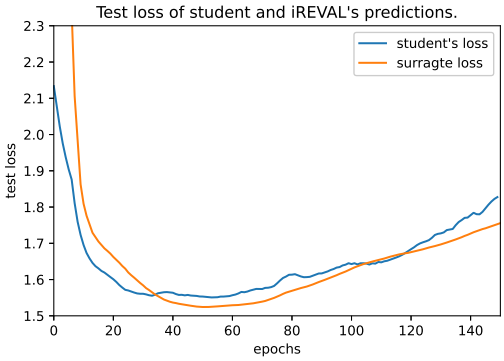


Figure 3.4: Supervisor's prediction compared to student's test loss on the CIFAR-10 dataset with the 4-layer MLP models.

into iREVAL over REVAL.

We summarize the test loss and accuracy for the other dataset (MNIST, CIFAR-100) in Table 3.1. When combined with learning rate decay and data augmentation, REVAL exhibits a complementary positive effect that improves the generalization performance of the model.

3.4.3.2 Effectiveness of Supervisor

Fig. 3.4 the accuracy of a supervisor's prediction of a student's test loss. This figure shows that the supervisor can predict the scale and time step of the overfitting from the training weights.

The training and test losses of the supervisor are shown in Fig. 3.5(a). Because we added new students to the training set of the supervisor, a spike appeared at the beginning of each iteration. The training curve continued to converge up until 200 epochs when the new students did not add further information about the loss surface to the supervisor. We consider this to be the reason why REVAL's training curve is similar to the baseline shown in Fig. 3.2.

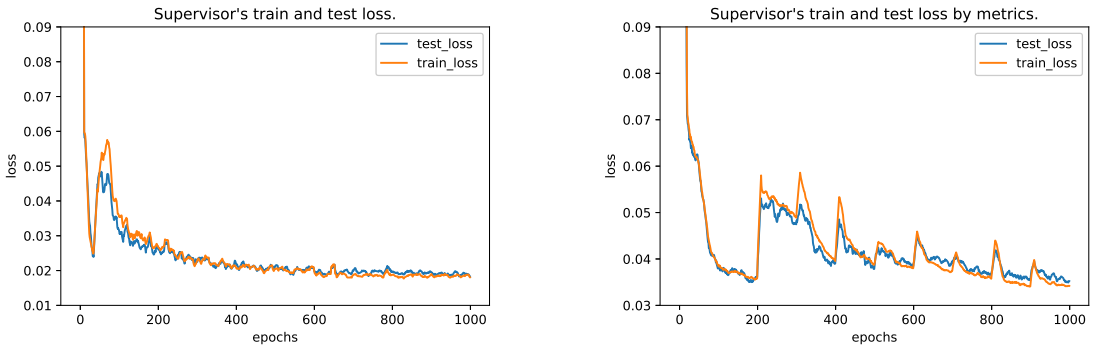


Figure 3.5: Supervisor's train and test MSE losses of the REVAL and iREVAL.

3.4.3.3 iREVAL: Improvements on REVAL

The iREVAL training uses the reduced sum of the weights as inputs. The reduced sum method decreased the dimension of weights but lost information about the loss landscape. This information loss problem is reflected in the later phases of supervisor training. In Fig. 3.5, iREVAL's supervisor (b) has a large training loss when meeting new students at the beginning of each iteration. However, after approximately 500 training epochs, iREVAL's supervisor also converged and the new students did not provide new information about the loss landscape. In Table 3.2, we show that the supervisor's parameters in iREVAL are a factor of 128x less than those in REVAL.

Table 3.2: Comparison of the REVAL and iREVAL supervisors' training on CIFAR-10 students.

Method	**Train loss**	**Test loss**	**Params**
REVAL	0.0196	0.0198	50.3M
iREVAL	0.0357	0.0324	393K

In Fig. 3.6, we show how an online update schema improves the supervisors' prediction. Although we have shown that the prediction is accurate when training without a supervisor,

the prediction accuracy drops quickly to a constant when used in the REVAL framework which has been reported as the gradient degradation problem in the meta hyperparameter optimization area.[7] However, after updating the supervisor online, Fig 3.6(b) shows that it helps update itself in time to capture the changes in position on the loss surface. Finally, online updating makes the surrogate loss curve closer to the student's test loss curve. We considered an accurate prediction of the loss surface to lead students to a more generalized solution.

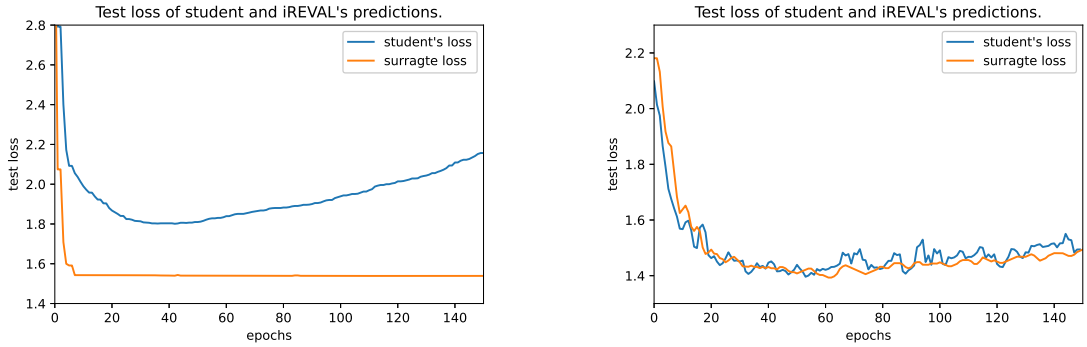


Figure 3.6: Comparison of how the offline and online training affects supervisor's prediction.

3.5 Analysis

3.5.1 Proof of the supervisor's consistency

We followed the proof mentioned in Li et al.[37] to prove the supervisor's consistency. The proof includes the following two steps. First, we define the supervisor's *empirical loss* and *expected loss*. Then, by applying Azuma's Inequality ,[46] we could give an upper bound of the gap between the *empirical loss* and *expected loss*.

Definition 3.5.1 (Empirical Loss)**.** The empirical loss of the supervisor is defined as:

$$L_N(s) = \frac{1}{N}\sum_{n=1}^{N} l(s(\theta_n;\phi), v_n) \tag{3.10}$$

where v_i is the label of the θ, ϕ is the trainable weights of the supervisor, N is the number of samples in the replay buffer.

Definition 3.5.2 (Expected Loss)**.** Let Θ be the space of trainable weights for the students, the expected loss of the supervisor on a dataset D is:

$$L(s) =_{(\theta,v)\sim D} [l(s(\theta), v)] \tag{3.11}$$

Using the above definitions, we can further define the gap between the *empirical loss* and *expected loss* as $|L_N(s) - L(s)|$. Therefore, our target is to show this gap is bounded by an arbitrarily small value ϵ:

$$P[|L_N(s) - L(s)| < \epsilon] > 1 - \delta \tag{3.12}$$

where δ is a small constant value.

Theorem 2 (Azuma's Inequality)**.** *Suppose* $\{X_n, n = 0, 1, 2, ...\}$ *is a martingale and* $|X_n - X_{n-1}| < c_n$ *almost surely. Then for all positive integers* N *and all positive real* ϵ*,*

$$P[|X_N - X_0| \geq \epsilon] \leq 2\exp(\frac{-\epsilon^2}{2\sum_{n=1}^{N} c_n^2}) \tag{3.13}$$

By applying Azuma's Inequality to determine the boundary of the $|L_N(s) - L(s)|$, we could prove the *lemma* of the supervisor's consistency in Theorem 3.

Theorem 3. *Let* S *be a hypothesis class containing all the possible hypotheses of supervisor s. For any* $\delta > 0$*, with probability at least* $1 - \delta$*,* $\forall s \in S$*:*

$$|L_N(s) - L(s)| < \sqrt{\frac{2(d + \ln \frac{2}{\delta})}{N}} \tag{3.14}$$

where d is Pollard's pseudo-dimension of S.

Proof. Because Azuma's Inequality requires the target sequence to be a martingale, we construct such random variables $M_1, M_2...M_N$ where $M_n = l(s(\theta_n), v_n) - L(s)$. Suppose $l(.) \in [0, 1]$, then we have $|M_i| \leq 1$. Let $O_n = \sum_{i=1}^{n} M_i$ with $O_0 = 0$, then for any $1 \leq n \leq N$:

$$\begin{aligned}
&E[O_n|O_{n-1}, ...O_0] \\
=&E[M_n + O_{n-1}|O_{n-1}, ...O_0] \\
=&E[l(s(\theta_n), v_n) - L(s) + O_{n-1}|O_{n-1}, ...O_0] \\
=&O_{n-1}
\end{aligned}$$

which shows that O_n is a martingale and note that $|O_n - O_{n-1}| = |M_n| \leq 1$.

Now we can apply Azuma's Inequality to O_N by replacing the X_N with O_N and c_n with 1. Then we have $|O_N - O_0| = |O_N| = |N(L_N(s) - L(s))|$ and for any $\epsilon > 0$, we have the boundary:

$$P[|L_N(s) - L(s)| \geq \frac{\epsilon}{N}] = 2\exp(\frac{-\epsilon^2}{2N}) \tag{3.15}$$

By setting the $\epsilon = \sqrt{N}\lambda$ and $\lambda = \sqrt{2(d + \ln \frac{2}{\delta})}$, we obtain get the results claimed in Theorem 3. □

3.5.2 Loss Landscape Based on the Training and Supervisor's Gradients Directions

To demonstrate why REVAL helps decrease overfitting, we applied the same visualization method discussed in.[30] In their original method, the loss landscape was plotted along two filter-wise normalized random directions based on a Gaussian distribution. Owing to the high dimensionality of the weight space, these two random directions can be regarded as nearly orthogonal with a high probability. In fact,[?] shows the expected cosine similarity between Gaussian random vectors in n-dimensions is rough $\sqrt{(2/\pi n)}$. The equation for obtaining the loss landscape is as follows:

$$f(\alpha, \beta) = L(\theta^* + \alpha d_1 + \beta d_2) \tag{3.16}$$

where d_1, and d_2 are the filter-wised directions and the α, and β are two scale factors with range $[-10, 10]$ for sampling on the loss landscape.

In contrast to the random directionsd_1, d_2, we replace them with the training gradients and the supervisor's gradients to show the advantages of the synthesis direction.

$$f(\alpha, \beta) = L(\theta^* + \alpha \nabla_\theta L_{st} + \beta \nabla_\theta L_{sp}) \tag{3.17}$$

The contour loss landscape is shown in Fig 3.7 where the x-axis is the training direction and the y-axis is the supervisor's provided direction. Firstly, Fig 3.7 suggests that the local minima are distributed on the diagonal of the loss landscape. We believe this is caused by the scale factor being more balanced on the diagonal, for example: $[100, 0], [90, 10], [50, 50], [10, 90], [100, 0]$. The second thing that interested us is, compared to the lower-right region, the upper-left region is smoother, and the minima tend to that region. This phenomenon explains why the supervisor's gradients increase generalization by smoothing the loss landscape.

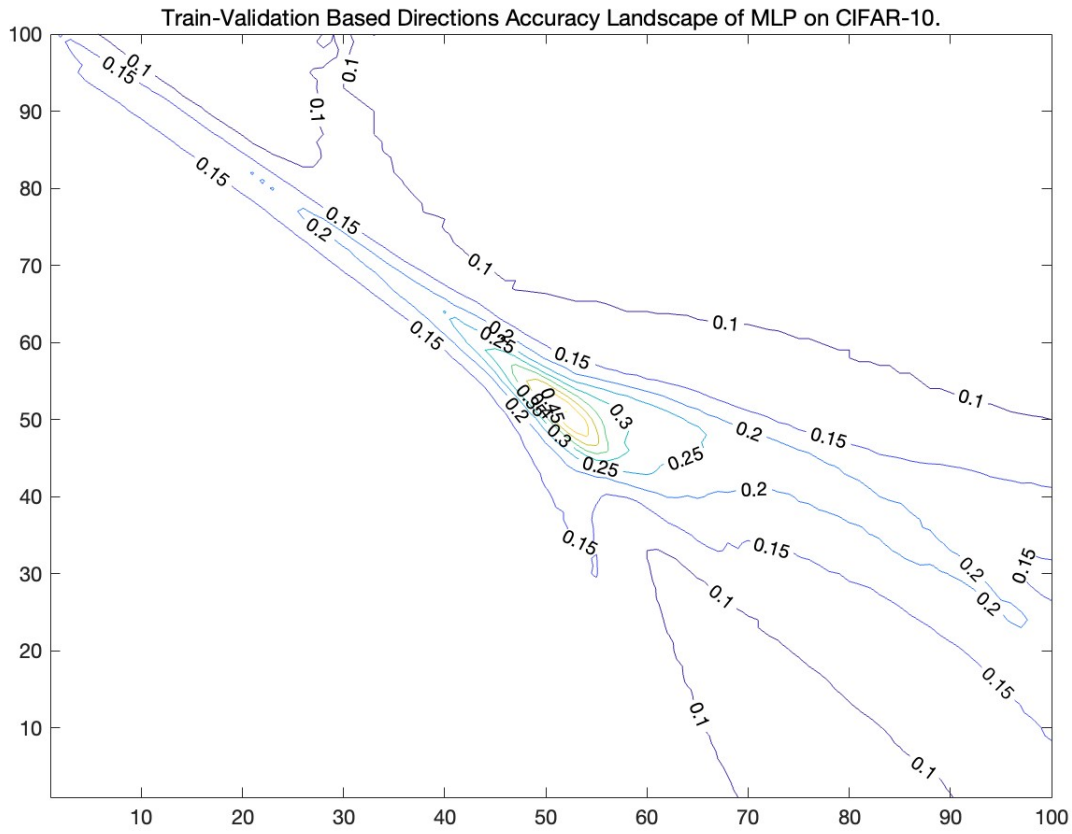


Figure 3.7: Validation Accuracy landscape based on the train gradients (x-axis) and supervisor's gradients (y-axis). Compared to the lower-right region, the upper-left region is smoother, and the minima tend to be in that region. This phenomenon suggests the supervisor's gradients increase generalization by smoothing the loss landscape.

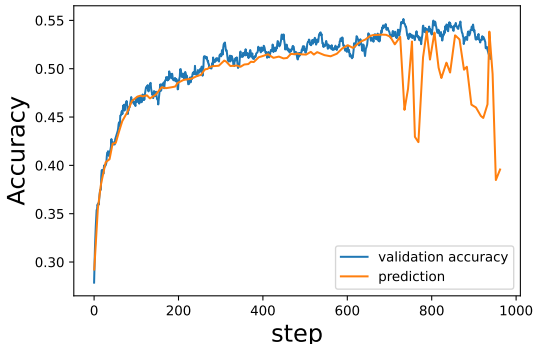


Figure 3.8: Predicted accuracy for each student during the iterations without the normalization for the weights.

3.5.3 Normalized weights

During our experiments, we found that directly training the supervisor model with raw student weights caused a high variance in the prediction at the end of the online student training, as shown in Fig 3.8. We consider that the variance of the accumulated gradients causes this during the student training process, therefore, we apply $l2$ normalization to reduce this effect, which we denote as $\theta = \frac{\theta}{||\theta||_2}$.

3.6 Conclusion and Future Work

This work is the first step toward utilizing the loss landscape information for further training of deep models, particularly the MLP model. First, our experiments revealed the fact that an MLP model can be used as an estimator of the validation loss of another MLP model. Second, using this surrogate MLP model, we proposed a supervisor-student training framework named REVAL to solve the overfitting problem. Finally, to handle the computation and space cost, and unstable gradients problem, we proposed several improvements to maintain the efficient training of REVAL.

This REVAL framework has many straightforward improvements:

- Metric encoding. Our approach utilizes the mapping between the weight space and

the validation loss directly. However, by applying dynamic programming, a more efficient knowledge encoding structure exists to guide student training.

- Weight sampling. Our approach takes the gradient from the supervisor and causes the vanished gradient and failure of the supervisor problems; by adjusting the gradient method to the sampling method may decrease such problems.

- Efficiency improvements. When applying REVAL, training MLP models on the CIFAR-10 dataset requires hundreds of GPU hours which means that it is unaffordable for other large-scale models.

Dataset	Model & Setup	Baseline		REVAL		iREVAL	
		Loss	Accuracy	Loss	Accuracy	Loss	Accuracy
MNIST	MLP	0.067	98.36%	0.064	98.46%	0.061	98.54%
	MLP&LRD	0.061	98.52%	0.060	98.56%	0.058	98.64%
	MLP&DA&LRD	0.052	98.70%	0.048	98.79%	0.037	98.98%
CIFAR-10	MLP	1.437	52.45%	1.429	52.87%	1.401	53.52%
	MLP&LRD	1.372	53.96%	1.368	54.34%	1.356	55.47%
	MLP&DA&LRD	1.164	59.32%	1.133	59.80%	1.058	61.78%
	ResNet-56	0.8188	78.42%	0.8142	79.73%	0.7933	81.13%
	ResNet-56&LRD	0.5927	81.94%	0.5339	83.55%	0.4479	86.42%
	ResNet-56&DA&LRD	0.5607	91.53%	0.5578	91.78%	0.5038	92.39%
CIFAR-100	MLP	3.316	22.45%	3.286	23.07%	3.264	24.18%
	MLP&LRD	3.304	23.39%	3.282	23.58%	3.257	24.33%
	MLP&DA&LRD	3.029	32.43%	3.029	33.17%	2.988	33.67%
	ResNet-56	1.239	60.07%	1.186	61.17%	1.162	62.72%
	ResNet-56&LRD	1.205	61.43%	1.196	61.88%	1.147	63.81%
	ResNet-56&DA&LRD	1.049	67.06%	1.001	68.44%	0.998	68.64%

Table 3.1: Comparisons of the Baseline, REVAL, and iREVAL (improved REVAL) on the MNIST, CIFAR-10, and CIFAR-100 datasets. DA: Data Augmentation, LRD: Learning Rate Decay.

Chapter 4

Meta Loss Landscape with Reinforcement Learning

4.1 Introduction

Detecting and avoiding overfitting in deep neural networks is of high importance. By analyzing training regimes, especially the validation progression during training, experts developed techniques to solve the overfitting problem while improving training. Those methods include, but not limited to, learning rate schedulers,[31,47,48] dropout,[14] adversarial training,[15,16] data augmentation,[17] and Adam optimizers.[49] Despite the experts' intuition, theoretical, and/or empirical insight of the high dimensionality and non-linearity of the deep models,[6] a natural question arises: *"Can we enable the gradient descent optimizer to decrease the validation loss, even if zero training loss indicates learning has stopped?"*. [1].

To address this problem, meta-learning[7] and especially meta-HPO(Hyper Parameter Optimization) methods define a bi-level objective framework. Typically, there are two common approaches for such bi-level optimization. In the first approach, we optimize

[1]This chapter was published as E. Zhang, R. Zhong, M. Munetomo and M. Wahib, Validation Loss Landscape Exploration with Deep Q-Learning, 2024 International Joint Conference on Neural Networks (IJCNN), 2024, pp. 1-9, doi: 10.1109/IJCNN60899.2024.10651182.

the outer target and let the controller search the parameter space. Then, we improve the controller using the meta-info from the inner optimization. This method is easy to implement, for example, grid search, yet is costly to compute due to the computationally-intense inner optimization.[8,9] In the second approach, we update the outer objective directly: a meta-gradient-based approach calculates the high-order gradient to the hyperparameters or settings.[10,11] However, despite the impractical cost of the high-order Hessian matrix, the degradation of the meta-gradient is also serious. Thus, decoupling the meta-gradient while training the inner model directly and efficiently is needed.

Based on the practical limitations discussed above, this chapter practically exploits the validation loss during the gradient descent process. In this context, "exploit" means using the past validation knowledge in the form of selecting the action with the maximum value under the given state. Naturally, this notion of exploitation lends itself conceptually to Reinforcement Learning[50] (RL). Baring similarity to the work by Li et al.,[51] we commonly treat the gradient-based optimization as the RL task or MDP (Markov Decision Process). In particular, by treating the weight and gradient spaces as the state and action spaces, respectively. However, the most significant difference is that in this chapter, we treat the validation loss as the reward while they treat the convergence rate and train loss as the reward. To be consistent with the RL terminology, we call this gradient action provider ***the controller***(trainer), while the model trained on the target dataset for exploring the validation loss landscape ***the worker(s)***(trainee). The test metrics are obtained from an isolated test dataset.

We call our framework Validation Knowledge Inheritance (VKI). In this chapter, "validation knowledge" refers to the weights and validation losses generated during the worker's supervised training process. "Inheritance" refers to exploiting the past validation knowledge through a reinforcement learning scheme that continuously accumulates information on the validation loss landscape. We give an overview of VKI in Fig 4.1. To achieve the target of VKI, we use a Q-network[39] as the controller that learns the mapping from the

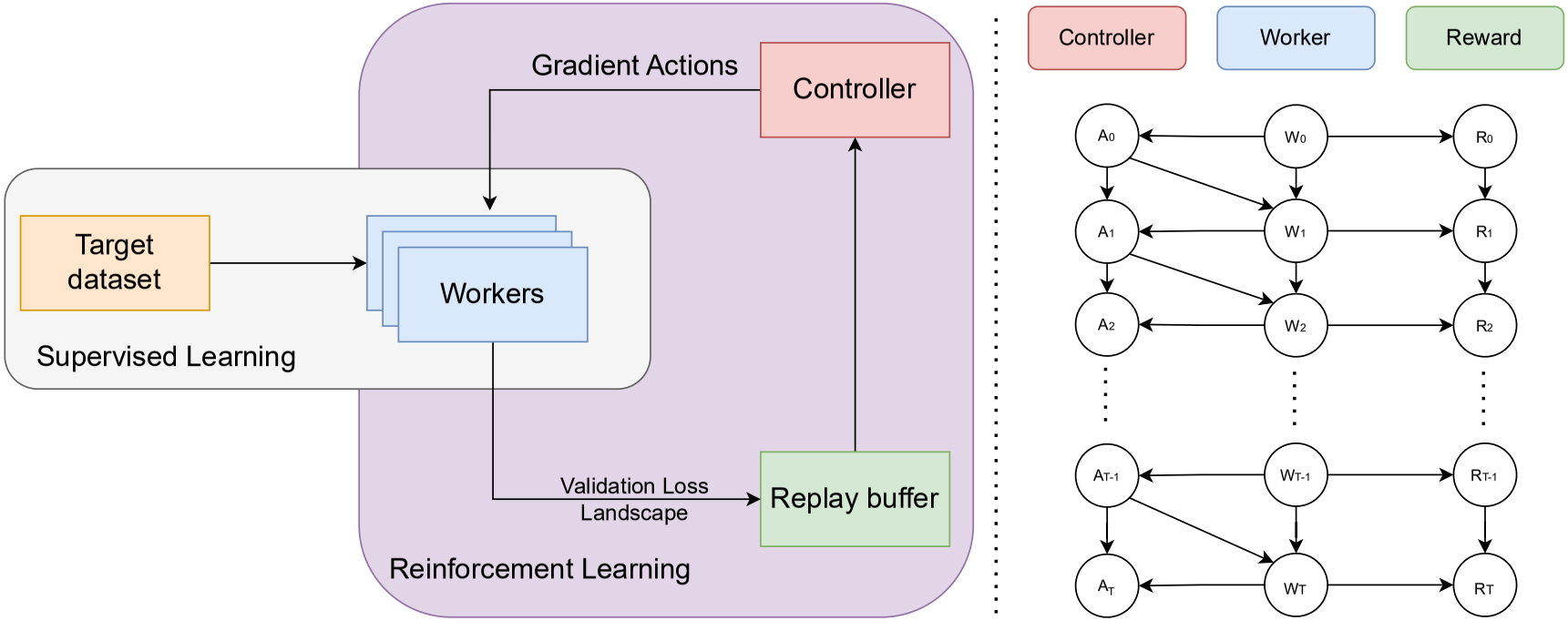


Figure 4.1: Validation Knowledge Inheritance (VKI). Top: an overview of how the workers, controller, and replay buffer interact. Bottom: an overview of how the controller provides action gradients to update workers' weights.

weight space to the future validation rewards of action. Then, during the workers' training, we apply this controller for selecting actions under a reduced action space. Under this scheme, the controller can potentially find scheduling rules similar to those found by expert humans.

4.2 Background

VKI reuses the training trajectories sampled from the validation loss landscape to guide the future training of workers. Li et al.[30] visualize the loss landscapes of ResNet and explain why the skip connections made the ResNet easier to optimize. Similarly, many studies looked into the properties of the loss landscape, e.g., sharpness,[31] smoothness,[30] and connectivity,[32] to understand the loss landscapes better. Another approach is to accelerate the training or try to find a more generalized solution by ensembling the model's weights,[32,33] or decreasing an expert sharpness-based target to find flatness near minima.[31,52]

VKI differs from the above works since it explores the validation loss landscape instead of the training loss. From this perspective, VKI can be regarded as a learning-to-learn method highly related to the meta-learning methods.[7] In this chapter, VKI improves workers by scheduling the hyperparameters, which may, at the surface, appear to be similar to the current methods used in hyperparameter optimization (HPO) or meta-HPO,[10] especially reinforcement learning-based solutions.[9] The difference is, however, that instead of treating hyperparameters as actions, we treat the weights space as the state space and the gradient space as the action space. Thus, a more accurate attribution of VKI would be the meta-optimizer.[51,53,54] Note that for the meta-gradients methods, updating the outer target of the bi-level optimization directly can lead to performance degradation and a high cost for computing the Hessian matrix.[7] On the contrary, the Q-Net plays a role in efficiently estimating the future reward of a given gradient based on the current weight. To further clarify the novelty and advantages of VKI, we provide an elaborate discussion in Sec 6.4.7.

Decaying the learning rate[47] is a standard method to avoid overfitting. However, for scaling the learning rate to accelerate training, Keskar et al.[31]y observed a linear scaling rule that tunes the learning rate according to the batch size to accelerate training. In one hour, they used this rule to train ResNet50 on ImageNet.[48] You et al. proposed a layer-wise learning rate adaptive optimizer (LARS) to accelerate training[55] further. In contrast to these empirical methods that human experts derive, the controller in VKI can be regarded as an automated validation loss landscape-awareness learning rate scheduler for deciding when to scale the learning rate or decay it.

Finally, our method VKI is also inspired by the recent advances in Network Architecture Search (NAS).[37,56] In these works, reinforcement learning is used to control the generation of high-value network structures. However, in contrast, our work treats the neural network parameter optimization as a reinforcement learning task, inherits the validation knowledge hidden in the loss landscape, and uses it in later workers' training.

4.3 Meta Loss landscape with Reinforcement Learning

Since the core method of VKI is to use deep Q-learning to decrease the validation loss in gradient descent-based supervised learning, we briefly introduce the relevant background of reinforcement learning and supervised learning(image classification), respectively.

4.3.1 Reinforcement Learning

In reinforcement learning, the agent interacts with an environment E, and a series of actions reward this interaction. At the state s_t, the agent must select a corresponding action a_t from the action space A. For the state of this action (s_t, a_t), the agent will receive a reward r_t, and the reward r_t depends on the reward function $R(s_t, a_t)$. For a certain dataset, we can obtain multiple complete state-action-rewards trajectories denoted as $T = (s_1, a_1, r_1 \dots s_t, a_t, r_t)$, known as Markov Decision Process (MDP), in particular when the number of states and

actions is finite. The state transitions are deterministic, and MDP converges to a stable state.

Based on the above definitions, many reinforcement learning algorithms aim to find an agent with the maximum expected reward function in a limited number of iterations. Here, we define the optimal action-value function $Q^*(s, a)$ as follows:

$$Q^*(s, a) = \max_{\pi} E[R_t | s_t, a_t, \pi]$$

where $R_t = \sum_{t'=t}^{T} \gamma^{t'-t} r_{t'}$ is the discounted future rewards from time step t, γ is this discount factor, and π is the policy mapping sequences to actions. It is worth mentioning that the optimal action-value function $Q^*(s, a)$ follows the *Bellman Equation* which suggests that if the rewards of each action a are known at state s, then the optimal policy is to select the action a^* with a maximum expected value of $r + \gamma Q^*(a^*, s)$. By using the *Bellman Equation* as the update iteration, $Q_{i+1}(s, a) = E_{s'}[r + \gamma \max_{a'} Q^*(s', a') | s, a]$, then $Q_{i+1} \to Q^*$ if $i \to \infty$.[50]

However, simply applying this update rule is impractical. To obtain acceptable generalization for different datasets, the deep Q-learning algorithm[57] uses a non-linear network (MLP, CNN, and ResNet) called Q-network to approximate the optimal value function $Q^*(s, a)$. Additionally, a replay buffer B [39] is also commonly used to remove the correlation between observations and to smooth the changes between data distributions. The objective function of the Q-network is:

$$L_i(\phi_i) = E_{(s,a,r,s') \sim B}[(r + \gamma \max_{a'} Q(s', a'; \phi_{i-1}) - Q(s, a; \phi_i))] \tag{4.1}$$

4.3.2 Image Classification

Although the VKI framework is not restricted to a specific class of supervised learning tasks, here we discuss VKI in the context of training models for image classification tasks.

Consider an image dataset D consisting of input images $x \in R^d$ and the one-shot label $y \in R^K$ where K is the number of classes. We denote the model as $f : R^d \rightarrow R^K$ and parameterize it by the trainable weights θ. For each sample $(x, y) \in D$, the output prediction is given by $\tilde{y} = f(x; \theta)$. Let $l : R^K \times R^K \rightarrow R$ denote a loss function (the commonly used softmax cross-entropy loss l_{CE} for the classification tasks), then the optimization process can be written as:

$$\min_{\theta} \quad L(x, y) = \frac{1}{N} \sum_{n}^{N} l_{CE}(f(x_i; \theta), y_i) \tag{4.2}$$

where N is the size of the training dataset D, and the loss L is the training loss. Many useful gradient-based optimization methods can be applied to optimize this objective function Eq.5.1. Here, for simplicity, we consider the widely used stochastic gradient descent (SGD) to update the trainable weights θ:

$$\theta \leftarrow \theta - \alpha \nabla_{\theta} L(x, y; \theta) - \lambda \nabla_{\theta} ||\theta||_{l_2} \tag{4.3}$$

where α is the learning rate, λ is the penalty factor of the regularization, ∇_{θ} is the differentiation operator w.r.t θ.

4.3.3 Validation Knowledge Inheritance

Based on the above notation, if we replace the state space S with the weights space Θ, the cost function C with validation loss function V, and the action space A with gradient space G, then we could transfer the context of supervised learning to become reinforcement learning. Here we can rewrite the optimal action-value function $Q^*(\theta, g)$ as follows:

$$Q^*(\theta, g; \phi) = \min_{\pi} E[V_t | \theta_t, g_t, \pi]$$

where ϕ is the trainable weights of Q-network, $V_t = \sum_{t'=t}^{T} \gamma^{t'-t} v_{t'}$ is the discounted future costs and π is the policy, they are the same notions as before. The loss function of this Q-network is also the same as Eq.4.1 and Q-network is updated by the following gradients:

$$\nabla_{\phi_i} L_i(\phi_i) = E_{(g,v,\theta')\sim B}[(v + \gamma \min_{g' \in G} Q(\theta', g'; \phi_{i-1}) - Q(\theta, g; \phi_i))\nabla_{\phi_i} Q(\theta, g; \phi_i)]$$

4.3.4 Regularization by Validation

A model is said to be *overfitting* when the model memorizes the samples in the training set by some specific features. Because when a model overfits at some point, it starts to exhibit an increase in loss. Continuing to train the model after loss starts to increase would make overfitting worse. That's why early stopping[13] is one of the methods used to solve the overfitting problem. Other solutions like dropout,[14] data augmentation,[17] and adversarial training[15,16] also help decrease overfitting when keep training the deep models.

Here we propose a new idea to avoid overfitting. Overfitting is analogous to a student trying to get a higher score on a test by memorizing the answer. In this analogy, our method is to have a supervisor who only tells the student the score in the exam but hides the answers. The supervisor would instead give the direction to the right state (the right way to answer the questions) based on the experience of the past students.

We named the model *student* who learns the target task and denote it as $f(x; \theta)$. Then, we can denote the loss function as $L_{st}(x, y; \theta)$ and with it, we can use the previously defined objective function Eq. 5.1 and the update of student's weights is also similar to the general optimization framework by Eq. 4.3.

Next, we proposed another network named *supervisor* and denoted as $s(\theta; \phi) : R^{d_\theta} \rightarrow R$, here d_θ is the dimension of student's weights. The supervisor is expected to give the surrogate validation loss and gradients that help the student get rid of the direction of over-

fitting and lead to the empirical optimal minima. The training of the supervisor can be regarded as a fitting problem with the Mean Square Error (MSE) loss function $L_{sp}(\theta;\phi)$:

$$\min_{\phi} \quad L_{sp}(\theta;\phi) = \frac{1}{M}\sum_{m=1}^{M}(s(\theta_m;\phi) - v_m)^2 \tag{4.4}$$

where $s(\theta;\phi)$ is the surrogate loss, θ is the weights of students, ϕ is the trainable weights of the supervisor, and M is the total number of the student training trajectories. During the training of students, validation loss v and their weights θ can be collected to form an experience dataset D_e for the supervisor's training. The update of the supervisor's weights is also similar to the general optimization framework by Eq 4.3.

Combined with the loss function of students $L_{st}(x;\theta)$ and supervisor $L_{sp}(\theta;\phi)$, we can formulate the REVAL algorithm as a bi-level optimization problem:

$$\begin{aligned} &\min_{\phi} \quad L_{sp}(\theta;\phi) \\ &\min_{\theta} \quad L_{st}(x;\theta) + \lambda L_{sp}(\theta;\phi) \end{aligned} \tag{4.5}$$

where λ is the penalty factor, by adjusting it, we could control the scale of the gradient from the supervisor. In other words, λ makes a similar difference in the regularization method which controls the scale of the normed value.

In comparison to the general objective function of training, instead of the norm-based regularization method, if we denote the L_{sp} as a distance operator $\|.\|_{L_{sp}}$, then REVAL can be regarded as an adaptable regularization method that could be updated after each iteration of students.

In other words, if we use the supervisor model instead of the norm function, for example, $s(\theta) = \|\theta\|_2$, it is exactly the same as the general objective function with l_2 norm penalty or weights decay.[12] Based on this, we view the general norm-based regularization method as a special case of the REVAL framework formulation.

Now with the supervisor's guidance, the updated equation of student's weights can be

rewritten as:

$$\theta \leftarrow \theta - \alpha \nabla_\theta L_{st}(x;\theta) - \alpha\lambda \nabla_\theta L_{sp}(\theta;\phi) \tag{4.6}$$

In comparison with Eq. 4.3, in the above-updated equations the training of students has another component of the gradient $\nabla_\theta L_{sp}(\theta;\phi)$ which comes from the supervisor.

It is important to note that while the first component of the gradient $\nabla_\theta L_{st}(x;\theta)$ leads the weights under the current train data, the second component of the gradient $\nabla_\theta L_{sp}(\theta;\phi)$ would provide the direction that leads to the empirical local minima among the past training trajectories of students.

Then, we could use this Q-network to exploit the loss landscape by the following update rule for the workers:

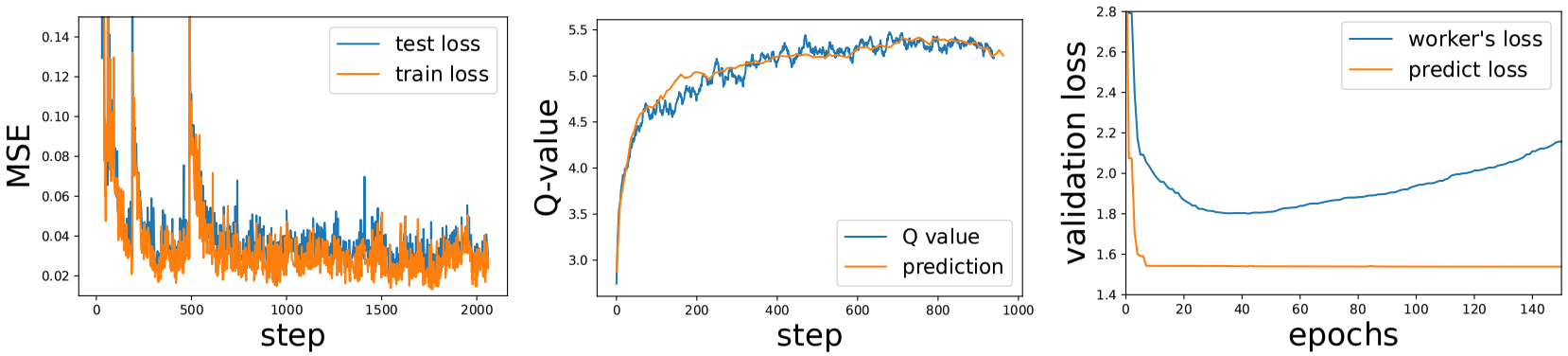
$$\theta \leftarrow \theta - \arg\min_{g \in G} Q(\theta, g; \phi) \tag{4.7}$$

In practice, we designed a reduced gradient space to replace G and the core reason for the gradient space replacement is the difficulty of representing or generating high-dimensional robust gradients without backpropagation. Thus, our reduced gradient space could be regarded as a neighborhood of the current batch gradient $\nabla_\theta L(x;\theta)$. Other reasons for the gradient space replacement include, for example, the expense of input dimensions for the Q-network (future work outside the scope of this chapter).

4.4 Experiments for Meta Loss landscape with Reinforcement Learning

4.4.1 Datasets

MNIST[41] is a dataset of handwritten digits containing $60,000$ training examples and $10,000$ test examples. CIFAR-10[42] and CIFAR-100[43] datasets contain $50,000$ training images and



(a) Controller's training loss

(b) Predicted and true Q-values

(c) Taken direct gradients

Figure 4.2: Controller's training/test losses, predicted Q-values, and gradient degradation

$10,000$ testing images with 10 and 100 classes, respectively. These datasets are commonly used across many classification tasks. When training the workers, we shuffled the training and test dataset. We split the dataset into 20% for the validation set and 80% training set. When training the workers we also collect the validation metrics and weights to train the controller.

4.4.2 Settings of Workers and Controller

4.4.2.1 Workers

For the four-layer MLP(Multilayer Perceptron) models, the units are $[128, 64, 32, classes]$ and the activation functions are $[ReLU, ReLU, ReLU, Softmax]$. For ResNet-56 we used the settings reported in.[44] All workers are initialized to the same weights by using the $glorot$ initializer.[45] The loss function is *cross entropy* loss and the optimizer is the naive SGD optimizer with a learning rate of 0.1. For the experiments with the Learning Rate Decaying (LRD), it starts at $(50\%, 75\%)$ steps of the total epochs with a factor of 0.1. The batch size is set to be 256 and all the workers are trained for 120 epochs (MLP) and 200 epochs (ResNet-56). For the experiments with Data Augmentation (DA), we process the images as recommended in.[44]

In order to process the samples efficiently, instead of validating each epoch, we collect the validation metrics after a certain training step, after what we call the validation gap. By adjusting the validation gap, we can adjust the sampling frequency on the metrics landscape. Here we set it to be every 30 in the reported experiments.

4.4.2.2 Controller

First, in Eq.4.7, $\arg\min_g Q(\theta, g; \phi)$ is an ideal case for using the Q-Network to generate a gradient g suitable for training the worker models directly. Second, we are more concerned about the validation accuracy rather than the validation loss. Therefore, in practice, we first

replace the validation loss with the validation accuracy. Then, replace the total gradient space with a set of Action A expanded by a group of learning rates αs or regularization penalty factors λs, where the number of the total actions is $||A|| = ||\alpha|| \times ||\lambda||$. The pair (α, λ) is selected at the beginning of the training step to update the workers as follows:

$$(\alpha, \lambda) \leftarrow \underset{(\alpha,\lambda)\in A}{\arg\max} \quad Q(\theta, (\alpha, \lambda); \phi) \tag{4.8}$$

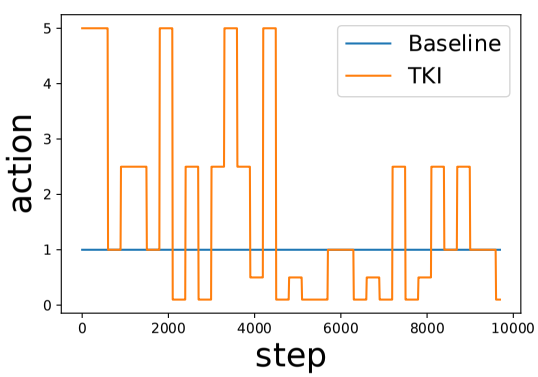
$$\theta \quad \leftarrow \theta - \alpha \nabla_\theta L(x; \theta) - \lambda \nabla_\theta ||\theta||_{l_2} \tag{4.9}$$

The controller is mainly a four-layer MLP model with a downsampling layer to reduce input size. For each layer in the controller, the number of units is $[128, 64, 32, ||A||]$ where $\alpha = [0.01, 0.1, 1.0, 2.5, 5.0]$ or $\lambda = [0.0, 1.0, 2.0, 4.0, 8.0]$ for learning rate or penalty factor for the discrete action space. For the continuous action space, $A = [-0.0001, 0.0, 0.0001]$ plus the current factors. Worth mentioning that, to avoid numerically unstable, the range of the continuous factors is limited in $[1e-4, 4.0]$. The activation functions are $ReLU$. The weights are initialized by $glorot$ initializer. The loss function we used is the MSE loss. The optimizer is an SGD optimizer with a learning rate of 0.01 and a batch size of 128.

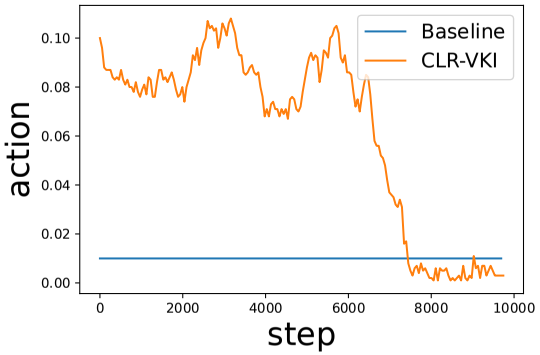
The ϵ-greedy policy has been applied with an annealing strategy during the iteration. Note that each iteration generates N new samples. To keep the balance between the number of new and old samples, we train the controller with N_n new samples and N_o old samples randomly selected from the replay buffer B.

4.4.3 Results

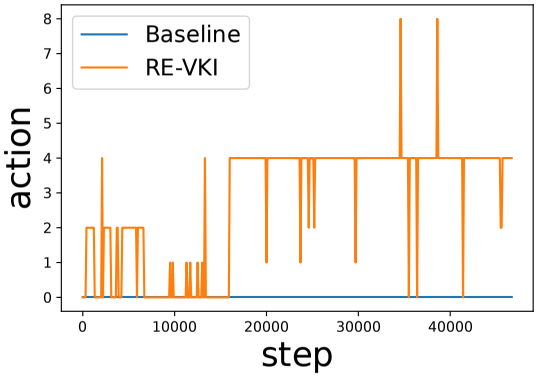
In this section, we discuss the results of our experiments, including the training of the workers and controller, the prediction of the controller, and the learning rate and penalty factor strategy found by the controller.



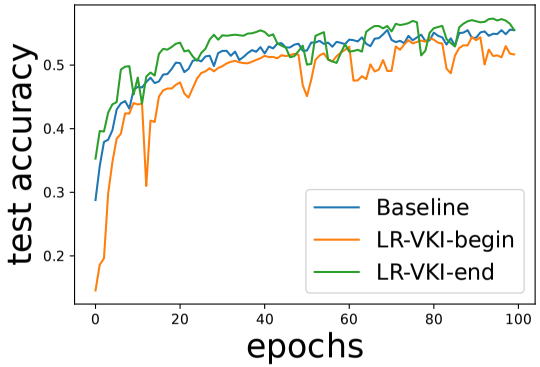
(a) LR-VKI learning rate



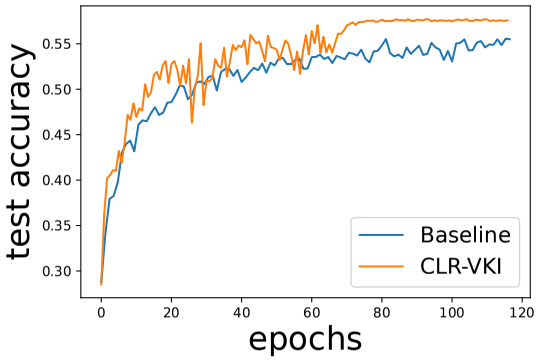
(b) CLR-VKI learning rate



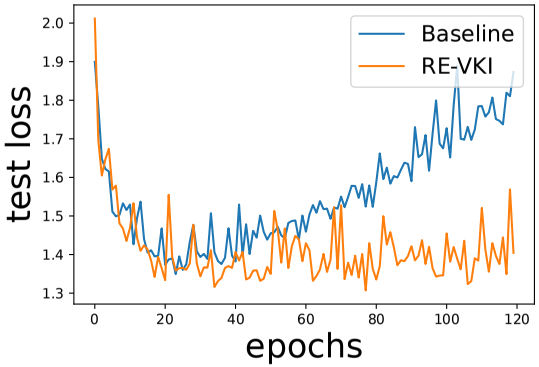
(c) RE-VKI penalty factor



(d) LR-VKI test accuracy



(e) CLR-VKI test accuracy



(f) RE-VKI test loss

Figure 4.3: "Discovered" discrete/continuous(C) learning rate(LR) and l_2-penalty factor(RE) schedulers on the CIFAR-10 dataset for the MLP models. The baseline is the naive training without the meta-HPO methods.

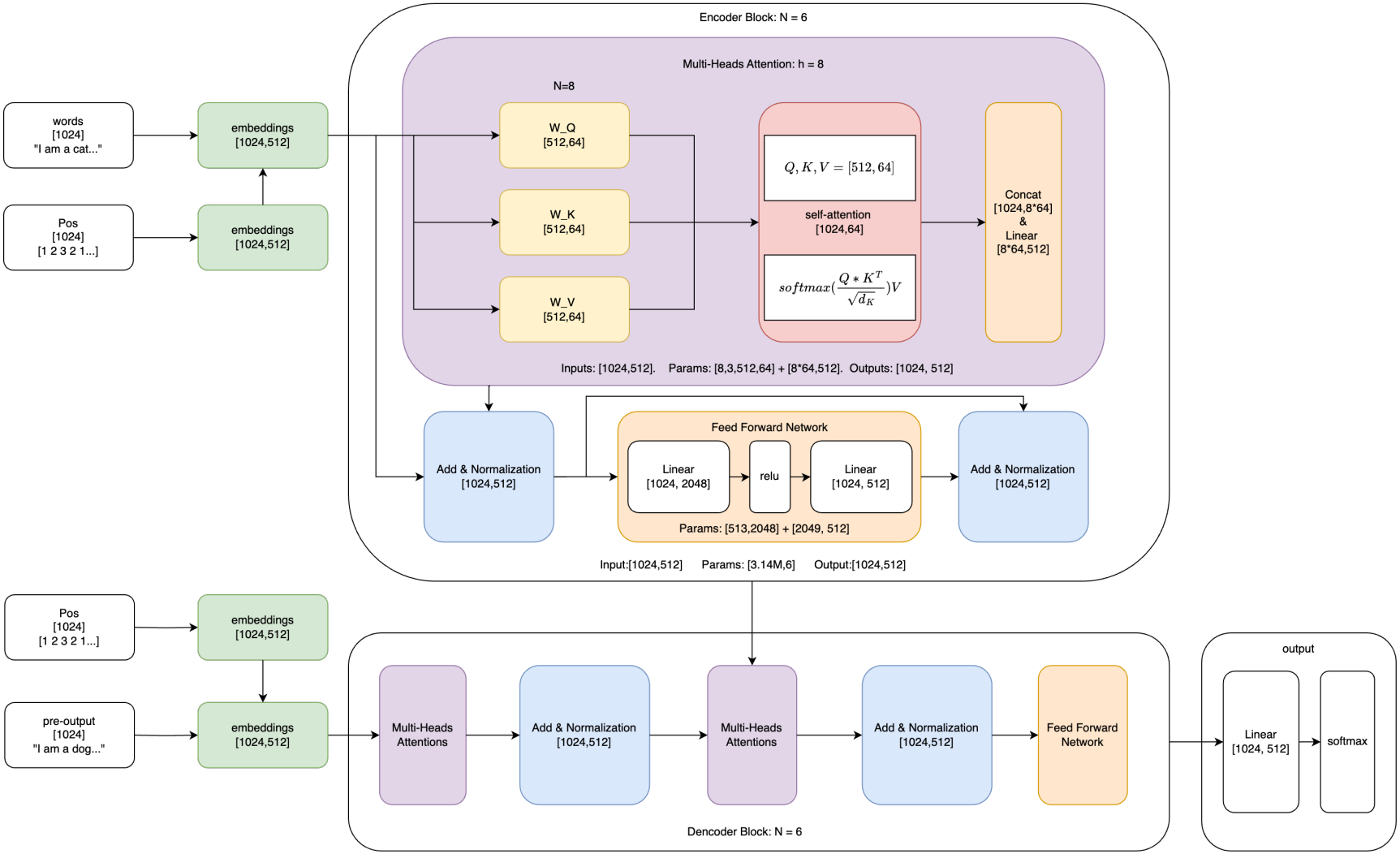
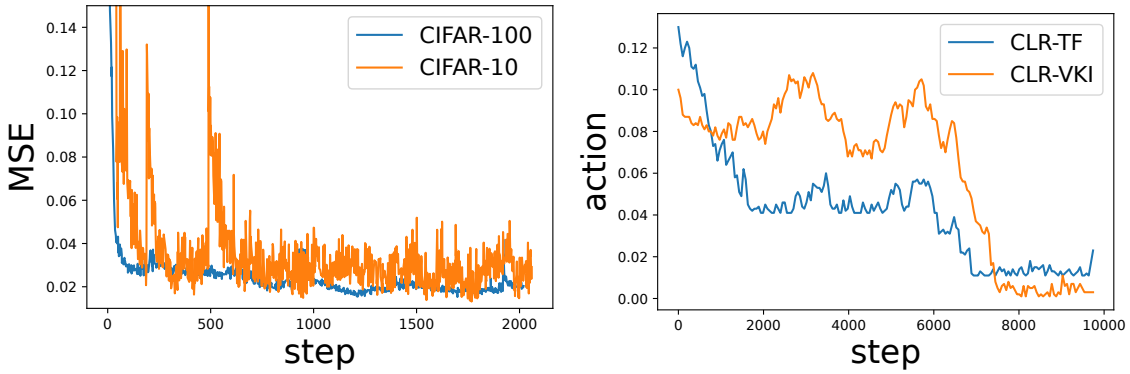


Figure 4.4: Structure of Transformer.



(a) Controller's training loss

(b) Source LR(CLR-VKI); Target LR (CLR-TF)

Figure 4.5: Transferability (TF) of VKI: pretrained Q-Net on CIFAR-10 converged faster when finetuned on CIFAR-100 dataset. Note that the LR found on the CIFAR-10 is similar to the fine-tuned result on CIFAR-100.

4.4.3.1 Training and Prediction of the controller

We show the training loss and test loss of the controller in Fig. 4.2(a). Because we add new samples to the replay buffer of the controller, there exits a spike at the beginning of each iteration. The training curve converges at about 1500 steps when new workers would not provide new validation knowledge to the controller. Fig. 4.2(b) shows how accurate the controller's prediction of a worker's Q value is. This figure shows us the controller could predict the worker's Q value from the worker training weights. Then, by choosing the actions with the max Q value, the training could be accelerated or stopped before overfitting.

4.4.3.2 Training of workers

In Fig. 4.3, we show how the controller guides the training of the worker through scheduling the learning rates and penalty factor. For the learning rates, initially, we considered that increasing[34] and decaying the learning rate may be discovered by the controller. However, not only could the acceleration of training and decaying be found, the results in Fig. 4.3a and Fig. 4.3e, surprisingly, show the discovered schedule is more like a combination of cyclical scheduling[58] with cosine decaying[47] which help the worker jump out of the local

minima and found more generalized solutions. For the penalty factors shown in Fig 4.3c, we found it to decay at the beginning and to increase when overfitting, which is what we expected. We show more details (*accuracy*, *variance*) of VKI in comparison to the baseline in the Table 4.1, it is worth mentioning that the increase of the variance by combining the manually set LRD method and LR-VKI shows a negative effect which we consider the conflict of the two approaches.

4.4.3.3 Transferring pre-trained Q-Net to other datasets.

Since the input of the Q-Net are the weights of deep models, the pre-trained weights can be transferred to the target dataset after fine-tuning. That is in contrast to meta optimizer and meta-HPO methods, in which the strategies applied on a model and dataset combination do not transfer to other datasets. Inspired by this idea, we consider the potential power for transferring the pre-trained Q-Net to target dataset. First, we reuse the previously obtained Q-Net from the CIFAR-10 dataset as the pre-trained model. Then, we conduct the same experiments on a subset of the CIFAR-100 dataset with 10 classes. The training of the Q-Net is shown in Fig 4.5. In comparison to the training on the CIFAR-10, first the finetuning process converges much faster and smoother. Second, in Fig 4.5b, we applied the Q-Net only trained for 200 steps on the target dataset and the LR schedule rule that is discovered is also similar to the source dataset.

4.5 Discussions and Improvements

4.5.1 Comparison to the Meta-HPO methods

Many Meta-HPO or learning-to-learning methods conduct a bi-level optimization to fit their motivation. Here we refer to the same notation in[10] of the bi-level objective function:

$$\lambda^* = \arg\min_{\lambda} L_V(\lambda, \theta^*) \text{ subject to } \theta^* = \arg\min_{\theta} L_T(\lambda^*, \theta)$$

where L_T is the training loss, θ is the model's weights, L_V is the validation loss, and λ is the hyperparameters.

The gradient-based method solves the above problem by calculating the hyper gradient $\frac{\partial L_v}{\partial \lambda}$ directly or indirectly through $\frac{\partial L_v}{\partial \theta}\frac{\partial \theta}{\partial \lambda}$. The direct gradient is easy to calculate but led to degradation in the prediction accuracy, mainly due to the gradient degradation problem (we show the inaccurate prediction in Fig 4.2c). For the indirect hyper-gradient, the calculation of $\frac{\partial \theta}{\partial \lambda}$ is difficult and needs approximation, for example,[10] applies Implicit Function Theorem (IFT) and Neumann series for further approximate the inverse Hessian matrix. However, first, in our proposed method VKI the action is selected from the prediction of Q-Net, and the Q-Net is trained online to save the high cost for the inner optimization. Thus, training such Q-Net avoids the calculation of both the $\frac{\partial L_v}{\partial \theta}$ and $\frac{\partial \theta}{\partial \lambda}$. The former used the gradient from the validation samples and the latter involves unacceptable calculation. Second, the prediction of the Q-Net is dependent on the weights and due to the transferability of weights, unlike other meta-require that need re-trained on the transfer dataset, the Q-Net can be reused on the transfer dataset with fine-tuning.

We summarize the main differences between the meta-learning methods in Appendix 2 and compare the transfer cost of VKI to the meta-HPO methods in Table 4.2 to demonstrate the reduction of the transfer cost where P is the number of hyperparameters, T is the adjusting steps, N is the number of samples, M is the cost of the training the model, W is

the number of weights, H is the cost for the hyper-gradient, Q is the cost for training the Q-Net, and $\lambda \leq 1$ is the transfer coefficient related to the past transferred samples N_e and mix parameter r.

4.5.2 Consistency of the controller

In the work of Li et al.,[37] the validation accuracy estimator s has been shown to learn the performance of different network structures sampled in the structure search space. On the other hand, VKI is using s to learn the discounted validation rewards from weights. In other words, the estimators s of their method can be represented as $s : R^{d_a} \rightarrow R$, while in our method the estimator can be represented as $s : R^{d_w} \rightarrow R^A$. Here d_a is the dimension of structure space, d_w is the dimension of the weight space, and R^A is the dimension of the actions' space. Accordingly, we reformulate the learning capability of the estimator in[37] as shown in Lemma.4 (proof in Appendix A).

Lemma 4. Let S be a hypothesis class, L_N be the empirical reward, and L be the expected reward. For any $\delta > 0$, with probability at least $1 - \delta$, $\forall s \in S$:

$$|L_N(s) - L(s)| < \sqrt{\frac{2(d + \ln \frac{2}{\delta})}{N}}$$

where d is Pollard's pseudo-dimension of S, and N is the number of total samples.

Lemma.4 shows us the convergence of the error bound is decided by Pollard's pseudo-dimension d and the number of total samples N. With a fixed d and infinite samples N, the gap between empirical reward and expected reward asymptote is less than a constant. Although using an infinite number of samples is impossible in experiments, this lemma suggests that we can move faster toward the error bound by reducing the input dimension of s and accelerating the training of workers.

4.5.3 Generalization bound of the transferability of VKI to new datasets

Lemma 5. Let $\delta^{'} = \delta - N_e^{\frac{1}{2(r+1)} - \frac{1}{4}}$ where $r > 1$ is required. Then for any sample S of size, N_e drawn according to an algebraic β-mixing stationary distribution. For any $\delta > 0$, with probability at least $1 - \delta$:

$$|R(L(s)) - R_{N_e}(L(s))| < O((N_e)^{\frac{1}{2(r+1)} - \frac{1}{4}} \sqrt{\log \frac{1}{\delta}})$$

where $R(L(s))$ and $R_{N_e}(L(s))$ denote the expected risk, the empirical risk of $L(s)$, N_e is the number of the transfer samples, and r is the mixing parameter.

Lemma 5. shows that the generalization bound became tighter as the increase in the mix parameter r denotes the independence and the number of transfer samples N_e. This lemma tells us the Q-Net can be continuously improved during training on the other transfer datasets.

4.6 Related Work

VKI reuses the training trajectories sampled from the validation loss landscape to guide the future training of workers. Li et al.[30] visualize the loss landscapes of ResNet and explain why the skip connections made the ResNet easier to optimize. Similarly, many studies looked into the properties of the loss landscape, e.g., sharpness,[31] smoothness,[30] and connectivity,[32] to understand the loss landscapes better. Another approach is to accelerate the training or try to find a more generalized solution by ensembling the model's weights,[32,33] or decreasing an expert sharpness-based target to find flatness near minima.[31,52]

VKI differs from the above works since it explores the validation loss landscape instead of the training loss. From this perspective, VKI can be regarded as a learning-to-learn method highly related to the meta-learning methods.[7] In this chapter, VKI improves workers by scheduling the hyperparameters, which may, at the surface, appear to be similar

to the current methods used in hyperparameter optimization (HPO) or meta-HPO,[10] especially reinforcement learning-based solutions.[9] The difference is, however, that instead of treating hyperparameters as actions, we treat the weights space as the state space and the gradient space as the action space. Thus, a more accurate attribution of VKI would be the meta-optimizer.[51,53,54] Note that for the meta-gradients methods, updating the outer target of the bi-level optimization directly can lead to performance degradation and a high cost for computing the Hessian matrix.[7] On the contrary, the Q-Net plays a role in efficiently estimating the future reward of a given gradient based on the current weight. To further clarify the novelty and advantages of VKI, we provide an elaborate discussion in Sec 6.4.7.

Decaying the learning rate[47] is a standard method to avoid overfitting. However, for scaling the learning rate to accelerate training, Keskar et al.[31]y observed a linear scaling rule that tunes the learning rate according to the batch size to accelerate training. In one hour, they used this rule to train ResNet50 on ImageNet.[48] You et al. proposed a layer-wise learning rate adaptive optimizer (LARS) to accelerate training[55] further. In contrast to these empirical methods that human experts derive, the controller in VKI can be regarded as an automated validation loss landscape-awareness learning rate scheduler for deciding when to scale the learning rate or decay it.

Finally, our method VKI is also inspired by the recent advances in Network Architecture Search (NAS).[37,56] In these works, reinforcement learning is used to control the generation of high-value network structures. However, in contrast, our work treats the neural network parameter optimization as a reinforcement learning task, inherits the validation knowledge hidden in the loss landscape, and uses it in later workers' training.

4.7 Conclusion

This work is the first step toward exploiting and exploring the validation loss landscape. First, our experiments reveal that deep Q-Network could be used to estimate workers' Q

value w.r.t. learning rates. Second, by applying the controller as a learning rate scheduler, we proposed a worker-controller training framework named VKI. Finally, through our framework VKI, we could ”rediscover” the previously found results for learning rate schedules. For future work, we plan to improve the training efficiency of the Q-Net for large-scale models and further explore the transfer ability among more datasets.

Table 4.1: Comparisons using the MNIST, CIFAR-10, and CIFAR-100 datasets with the Baseline (naive training without meta-method). Here, LR-VKI and RE-VKI mean the actions spaces for VKI are Learning Rate (LR) and REgularization (RE). MLP = Multilayer Perceptron; LRD = Learning Rate Decay; DA = Data Augmentation

Dataset	Model [Setup]	Baseline	RE-VKI	LR-VKI
MNIST	MLP	98.36% ±0.002	98.43% ±0.007	98.54% ±0.003
	MLP [LRD]	98.52% ±0.001	98.75% ±0.011	98.30% ±0.014
	MLP [DA+LRD]	98.70% ±0.007	98.97% ±0.013	99.37% ±0.021
CIFAR-10	MLP	52.45% ±0.007	52.87% ±0.007	53.52% ±0.001
	MLP [LRD]	53.45% ±0.012	54.14% ±0.008	54.87% ±0.027
	MLP [DA+LRD]	59.32% ±0.017	59.80% ±0.013	61.78% ±0.021
	ResNet-56	81.42% ±0.012	82.73% ±0.002	85.13% ±0.015
	ResNet-56 [LRD]	82.84% ±0.009	83.55% ±0.013	84.72% ±0.023
	ResNet-56 [DA+LRD]	93.53% ±0.007	93.78% ±0.005	94.39% ±0.022
CIFAR-100	MLP	22.45% ±0.005	23.07% ±0.012	24.18% ±0.011
	MLP [LRD]	23.39% ±0.003	23.58% ±0.014	23.33% ±0.026
	MLP [DA+LRD]	32.43% ±0.006	33.17% ±0.011	33.67% ±0.021
	ResNet-56	60.07% ±0.011	61.17% ±0.012	62.72% ±0.016
	ResNet-56 [LRD]	61.43% ±0.002	61.88% ±0.014	63.81% ±0.021
	ResNet-56 [DA+LRD]	67.06% ±0.009	68.44% ±0.012	68.64% ±0.027

Table 4.2: Test accuracy (loss), time, and transfer cost of the Meta-HPO methods. Other meta-HPO methods need to be re-trained on the new/transfer dataset, but VKI can be continually improved from the transfer samples. P: number of hyperparameters; T: adjusting steps; N: number of samples; M: cost of training the model; W: number of weights; H: cost for the hyper-gradient; Q: cost for training the Q-Net; $\lambda \leq 1$: transfer coefficient related to the past transferred samples N_e and mix parameter r.

Approach	Method	**CIFAR-10/100**	**Time/Transfer(s)**	**Complexity**
Meta Methods	Grid Search	87.42%(0.809) / 65.07%(1.239)	100K/100K	$O(P^T M)$
	Random Search	89.73%(0.752) / 65.06%(1.205)	100K/100K	$O(NM)$
	Bayesian Opt [59]	91.94%(0.651) / 66.88%(1.196)	100K/100K	$O(N^3 M)$
	RTHO [60]	91.62%(0.603) / 67.11%(1.147)	50K/50K	$O(WM)$
	STN [61]	93.55%(0.561) / -	25K/25K	$O(cM)$
	MH-IFT [10]	94.62%(0.554) / -	18.5K/18.5K	$O(H+M)$
Ours	VKI	95.67%(0.504) / 68.64% (0.998)	50K/10K	$O(\lambda Q + M)$

Chapter 5

Meta Generative Data Augmentation Optimization

5.1 Introduction

Over the past few years, large-scale supervised deep-learning models have achieved impressive performance gains, thanks to the availability of massive amounts of data.[62,63] However, overfitting on the training set can lead to poor generalization on the test set, undermining the utility of these models. To address this issue, data augmentation has emerged as a powerful technique for improving the generalization of models. Traditional image data augmentation techniques include random cropping, flipping, and color modification.[29,44] However, developing effective data augmentation strategies relies on experts' intuition and may require significant theoretical or empirical insight to reduce the validation loss.[1].

Recent research has focused on automatically designing data augmentation techniques tailored to specific datasets and tasks to overcome this limitation. One approach is to efficiently select effective combinations of image transformations from a large pool of can-

[1]This chapter was published as Zhang E, Dong B, Wahib M, et al. Meta generative image and text data augmentation optimization[J]. The Journal of Supercomputing, 2024: 1-19. DOI10.1007/s11227-024-05912-5.

didate operations. For instance, AutoAugment[21] and related methods[22–25] optimize the selection of augmentation operations to improve validation performance, achieving state-of-the-art results on several benchmarks. Another approach involves using conditional generative models[26,27] to improve model performance in low-data settings.

However, these methods have limitations. For example, conditional adversarial generative models can be challenging to train due to the generator's and discriminator's adversarial nature.[64,65] Meanwhile, AutoAugment approaches have a limited number of handcraft augmentation operations, and the cost of training the large-target model on the large-scale dataset can be prohibitively high.[21,22,24]

To overcome these challenges, our motivation is to replace the discriminator in the conditional generative adversarial model with a "fixed" validation loss of the target model. Here, "fixed" refers to the validation loss evaluated on the test dataset, which is fixed for each target model. In the meantime, we replace the limited data augmentation operation set in the AutoAugment approach. By doing so, we hope to solve the hard training problem in the GAN approach and increase the augmentation diversity in the AutoAugment method.

In this chapter, we propose a new method named Meta Generative Data Augmentation Optimization (MGDAO) to solve the limited operations in the AutoAugment approach and the hard training in the generative adversarial model. More specifically, we replace the operation action space in the AutoAugment course with a deep-style generator and return the discriminator with the validation loss from the target model's evaluation process.

We give an overview of MGDAO in Fig 5.3. To achieve the target of MGDAO, we use a generator as an agent that learns the changes from the train sample's domain to the validation sample's domain. Then, while training the target model, we apply this generator to generate fake test samples as data-augmented samples to train the target model and reduce the validation loss. Under this scheme, the generator can find different data augmentation techniques that reduce the validation loss.

The contributions in this chapter are listed as follows:

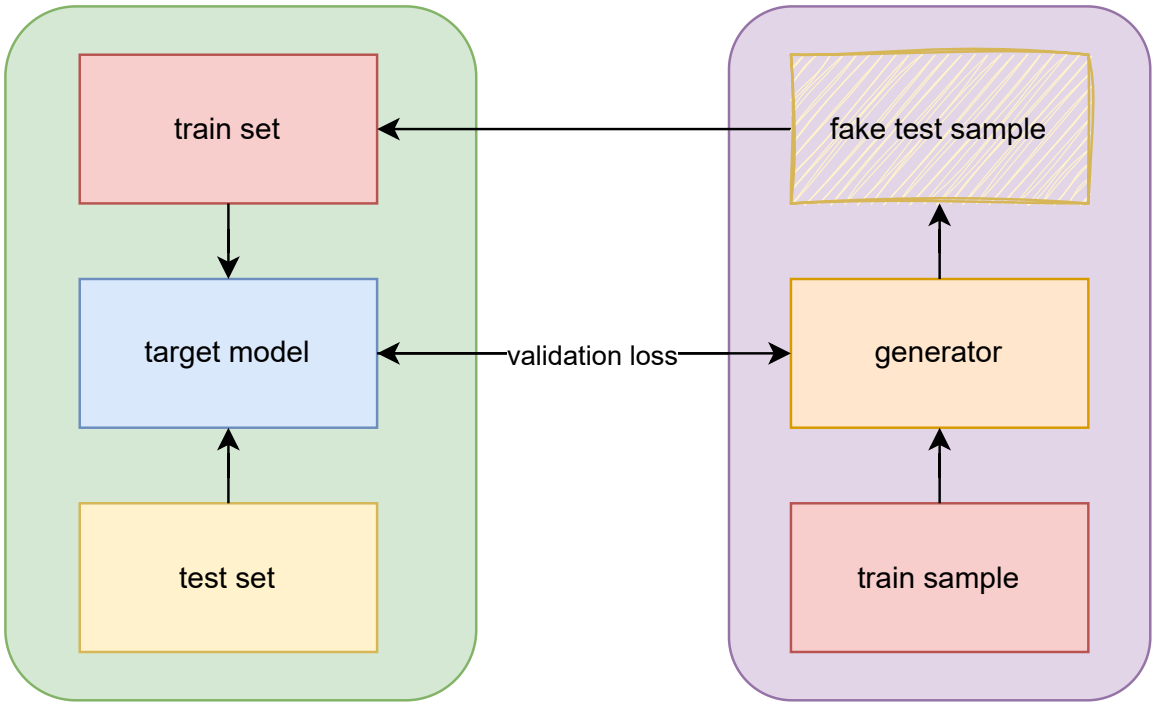


Figure 5.1: Meta Generative Data Augmentation Optimization.

- **A meta automatic data-augmentation method:** we show how our method can generate the validation loss-oriented augmentation training samples for few-shot. To illustrate the generalization for different types of input samples, we conduct experiments on both image classification tasks and system intrusion classification tasks.

- **A validation loss aware sample generator:** Compared to another automatic data-augmentation method, our scheduler generates samples that fit the validation loss. That is the reason we call the augmented samples fake test samples. As two approaches of MGDAO, first by pixel-wise similarity, we can ”rediscover” image data augmentation rules, for example, adding noise or masking, designed by human experts. Then, by the feature-level similarity, we can even change the feature of samples which is hard to achieve by engineering.

- **A few-shot training method:** MGDAO achieves notable improvements for few-shot training on common architectures over several image-based and text-based datasets

and models. Compared to the baseline, we find that the target model trained using MGDAO improves 1.4% test accuracy on the CIFAR-10 dataset, 2.46% on the MNIST dataset, and 1.56% on the CIFAR-100 dataset.

5.2 Related work

Data augmentation is a common technique used to improve the performance of deep learning models, particularly in classification tasks. It is often difficult to encode known invariances in a model. Instead, data augmentation generates additional data items through transformations from existing data items. For example, in handwritten character recognition, the labels should be invariant to small shifts in location, rotations, or changes in intensity, stroke thickness, and size.[29,44] Most data augmentation methods are designed based on a priori knowledge of the invariances.

In recent years, Generative Adversarial Networks (GAN)[66] and their variants have been successfully used to generate realistic images. The discriminator of a GAN is trained to distinguish between real and generated images, while the generator is trained to produce images that can fool the discriminator. Through this process, GANs can learn complex joint densities and generate diverse samples not found in the original dataset.[26] employed a generative adversarial network to generate images that augment datasets. Bayesian DA combined the Monte Carlo expectation maximization algorithm with GAN to generate data by treating augmented data as missing data points on the distribution of the training set.

AutoAugment[21] treats the data augmentation problem as an RL-base problem. In particular, they treat the accuracy of the validation dataset as the reward for training an RNN controller. Then, use it to train a child model to show the state-of-the-art performances on various datasets with different models. However, training such controllers is expensive, and they used thousands of GPU hours in their experiments. To address such problems,

Population-Based Augmentation(PBA)[22] is proposed, which was formerly used for hyper-parameters optimization. Based on the Bayesian optimization, Fast/Faster AutoAugmentation[23,24] is proposed to reduce the training cost, and they used several GPU hours on the reduced CIFAR-10 dataset.

However, the inefficiency of the AutoAugment-based methods is due to the non differentiable selection of the data augmentation operations.[25,67,68] Inspired by the differentiable architecture search,[69] they[67] relax the optimization problem to a differentiable bi-level problem. Then, applying the Implicit Function Theorem (IFT) and Neumann series, MGDAO directly optimized this bi-level problem by approximating the gradient of optimal training weights w.r.t the controller's parameter. Through their efforts, the efficiency of the AutoAugment-based methods is further improved.

5.3 Methodology

This section introduces the preliminary settings of the automatic and generative adversarial data augmentation methods. Then, we propose a new method based on the previous approaches of two data augmentation.

5.3.1 Generalizing AutoAugment Family

Consider the training dataset D_T and D_V consisting of input images $x \in R^M$ and the one-hot label $y \in R^C$ where C is the number of classes. We denote the network model as $f : R^M \to R^C$ and parameterized by the trainable weights θ. The policy parameters for the data augmentation are denoted as $\phi \in R^N$. For each sample $(x, y) \in D_T$, the output prediction is given by $\tilde{y} = f(x; \theta, \phi)$. Let $l : R^C \times R^C \to R$ denote a loss function (commonly use the softmax cross-entropy loss l_{CE} for the classification tasks), and then the optimization process can be written as:

$$\min_{\theta} \quad L_T(D_T; \theta, \phi) = \frac{1}{N_T} \sum_{(x_i, y_i) \in D_T}^{N_T} l_{CE}(f(\pi(x_i; \phi); \theta), y_i) \tag{5.1}$$

Where N_T is the size of the training dataset D_T, the loss L_T is the training loss, and π is the data augmentation policy. Then, the objective function of data augmentation policy in AutoAugment family methods can be generalized as:

$$\min_{\phi} \quad L_V(D_V; \arg\min_{\theta} L_T(D_T; \theta, \phi), \phi) \tag{5.2}$$

Where L_V is the validation loss, the above objective function means minimizing the training loss with θ and minimizing the validation loss with ϕ.

5.3.2 Generative Adversarial Data Augmentation

Generative Adversarial Methods are one approach for learning to generate samples by minimizing the distance between the distribution of the generated sample $\tilde{x}$ and the true sample x. The generative model G takes the latent Gaussian variables $\epsilon = N(0, I)$ as inputs, and the discriminator takes the generated sample $\tilde{x}$ and the true sample x as inputs to decide whether they came from fake/true distribution. The objective function defined for the generative adversarial model is:

$$\min_{G} \max_{D} V(G, D) = E_X[logD(x)] + E_Z[log(1 - D(z))] \tag{5.3}$$

The generative adversarial data augmentation methods treat the generative model G as the data augmentation operation for training the target model. Thus, the generator $\tilde{x} = G(x_e, \epsilon)$ takes an encoded representation of training sample x and Gaussian noise ϵ, adding the variance as inputs.

5.3.3 Meta Generative Data Augmentation Optimization

The objective function for MGDAO is similar to the AutoAugment approach; the difference is we use a generator to replace the operation set. The objective function for the generator g can be written as :

$$\min_{\phi} \quad L_g(D_T; \theta^*, \phi) = L_S + L_V \tag{5.4}$$

$$= \frac{1}{N_T} \sum_{(x_i, y_i) \in D_T}^{N_T} ||g([x_i, \hat{y_i}]; \phi) - \hat{x_i}||_{l_1} \tag{5.5}$$

$$+ L_V \tag{5.6}$$

Where L_S is used to control the l_1-similarity between the generated sample to the input x_i, and L_V is used for decreasing the validation loss.

The optimization for the L_S can take the gradient w.r.t ϕ directly. However, optimizing the L_V w.r.t ϕ needs to solve the bi-level problem defined in Eq. 5.2, which is difficult for two reasons. The first one is due to training the deep model θ in the inner optimization is expensive, and the second one is the search space of the ϕ growth exponential according to the numbers of the data augmentation policies. The approaches for solving these two problems are reducing the dimension of the parameter space[70,71] for ϕ or trying to reduce the training time[21,22,24] In the inner optimization for the network model.

In this chapter, we intend to solve this bi-level problem. Eq. 5.2 by directly taking the gradient w.r.t ϕ denoted as $\nabla_\phi L_V = \frac{\partial L_V}{\partial \theta} \frac{\partial \theta}{\partial \phi}$ which could be used to update the generator's parameters directly. However, cause of the high dimension of ϕ and θ, the time and space cost of calculating $\frac{\partial \theta}{\partial \phi}$ is expensive. Therefore, we apply the commonly used implicit function theorem (IFT) as an approximation to $\frac{\partial \theta}{\partial \phi}$. The definition of the IFT is as follows:

Theorem 6 (Cauchy's Implicit Function Theorem)**.** *Suppose for some* (θ, ϕ), $\frac{\partial L_T}{\partial \theta}|_{\theta', \phi'} = 0$ *and the regularity conditions are satisfied, then the surrounding the* (θ', ϕ')*, there is a function* $\theta^*(\phi)$ *s.t.* $\frac{\partial L_T}{\partial \theta}|_{\theta(\phi), \phi} = 0$ *and we have:*

$$\frac{\partial \theta^*}{\partial \phi} = -\left[\frac{\partial^2 L_T}{\partial \phi \partial \phi^T}\right]^{-1} \times \frac{\partial^2 L_T}{\partial \theta \partial \phi^T}\Big|_{\phi', \theta^*(\phi')} \tag{5.7}$$

While such a direct approach involves of calculation the inverse of the Hessian Matrix $\left[\frac{\partial^2 L_T}{\partial \phi \partial \phi^T}\right]^{-1}$ which is $O(M^3)$ expensive, M denote the dimension of the model's parameters θ, we approximate it by applying the Neumann series that is similar to the previous work [10,25].

$$\left[\frac{\partial^2 L_T}{\partial \phi \partial \phi^T}\right]^{-1} = \lim_{i \to \infty} \sum_{j=0}^{i} \left[I - \frac{\partial^2 L_T}{\partial \theta \partial \theta^T}\right]^j \tag{5.8}$$

Together with Eq 5.7, the approximate of $\frac{\partial \theta}{\partial \phi}$ can be write with a constant $J = 5$ here:

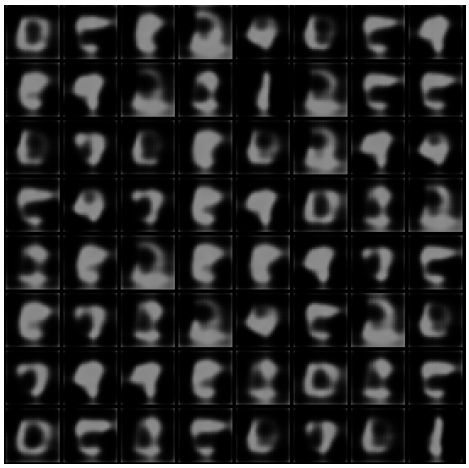
$$\frac{\partial \theta^*}{\partial \phi} = -\frac{\partial L_V}{\partial \theta} \sum_{j=0}^{J} \left[I - \frac{\partial^2 L_T}{\partial \theta \partial \theta^T}\right]^j \frac{\partial^2 L_T}{\partial \theta \partial \phi^T} \tag{5.9}$$

5.4 Experiments and Results

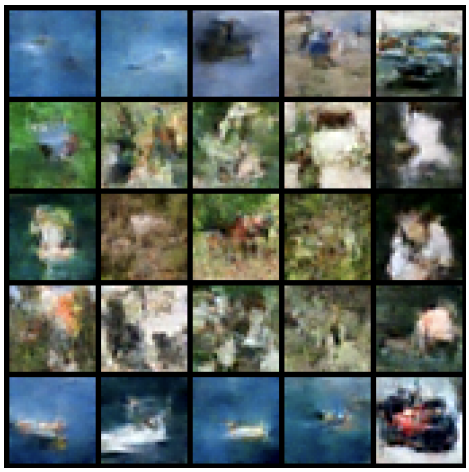
5.4.1 Dataset

We conducted experiments on the well-used MNIST, CIFAR-10, and CIFAR-100 datasets for image processing tasks. MNIST[41] is a dataset of handwritten digits containing $60,000$ training examples and $10,000$ test examples. CIFAR-10[42] and CIFAR-100[43] datasets contain $50,000$ training images and $10,000$ testing images with 10 and 100 classes, respectively. These datasets are commonly used across many classification tasks. When training the workers, we shuffled the training and test dataset. We split the dataset into 20% for the validation and 80% training sets. We also collect the validation loss to train the generator.

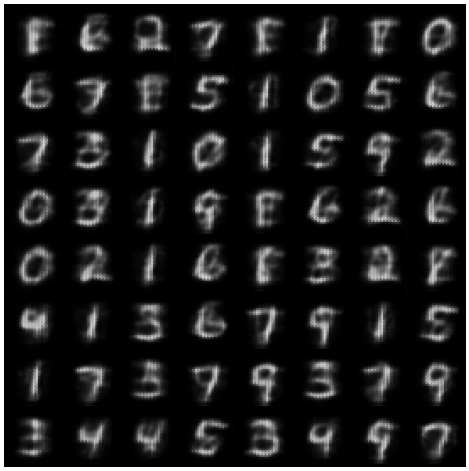
We also conduct experiments on the ADFA-LD[72] and ADFA-WD datasets for the text



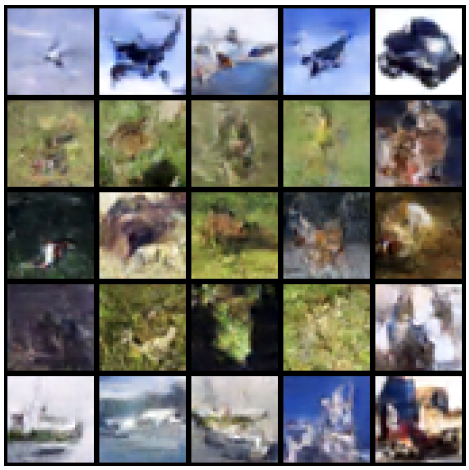
(a) Epoch 5



(b) Epoch 5



(c) Epoch 10



(d) Epoch 15

(e) Epoch 20

(f) Epoch 35

Figure 5.2: Generator's training for the Augmented MNIST and CIFAR-10 samples during epochs. The augmented samples gradually become explainable through interactive training with the generator and the target model.

Dataset	Target Model & Setup	Accuracy	
		baseline	MGDAO
ADFA-WD	MLP-4L/25-shots	72.21%	73.36%
	LSTM-3L/35-shots	84.71%	-
	LSTM-3L/Fullsize	93.52%	95.78%
ADFA-LD	MLP-4L/15-shots	74.16%	-
	MLP-4L/25-shots	79.28%	-
	MLP-4L/35-shots	81.12%	-
	Trans-6H8L/Fullsize	94.17%	95.53%

Table 5.1: Comparisons of the Baseline and MGDAO on the ADFA-WD and ADFA-LD datasets with several target models.

or sequential tasks. The ADFA is a set of system invasion attack datasets published by the Australian DeFence Academy. ADFA datasets include two systems datasets: Linux dataset(LD) and Windows Dataset(WD). For the ADFA-LD dataset, the samples of the intrusion trace are recorded by a sequence of the system calls under a given period. The ADFA-LD contains three parts: training set, validation set, and attack set. The training and validation are normal type traces, while the attack set contains six kinds of Intrusion. The range of the trace sequence is 300B to 10KB.

For the ADFA-WD system call trace, it also contains nine DLL calls, including ntdll.dll, kernel32.dll, user32.dll, comctl32.dll, ws2 32.dll, mswsock.dll, msvcrt.dll, msvcpp.dll, and ntoskrnl.dll.

5.4.2 Models

5.4.2.1 Generator

In the experiments, the generator we used was a conventional unit. The UNet generator has eight blocks, each with four convolutional layers with leaky ReLU activations and batch normalization followed by one downsampling or upsampling layer. Downsampling layers are convolution layers with stride two followed by leaky ReLu, batch normalization. Upsampling layers are stride 1/2 replicators, followed by a convolution, leaky ReLu, and

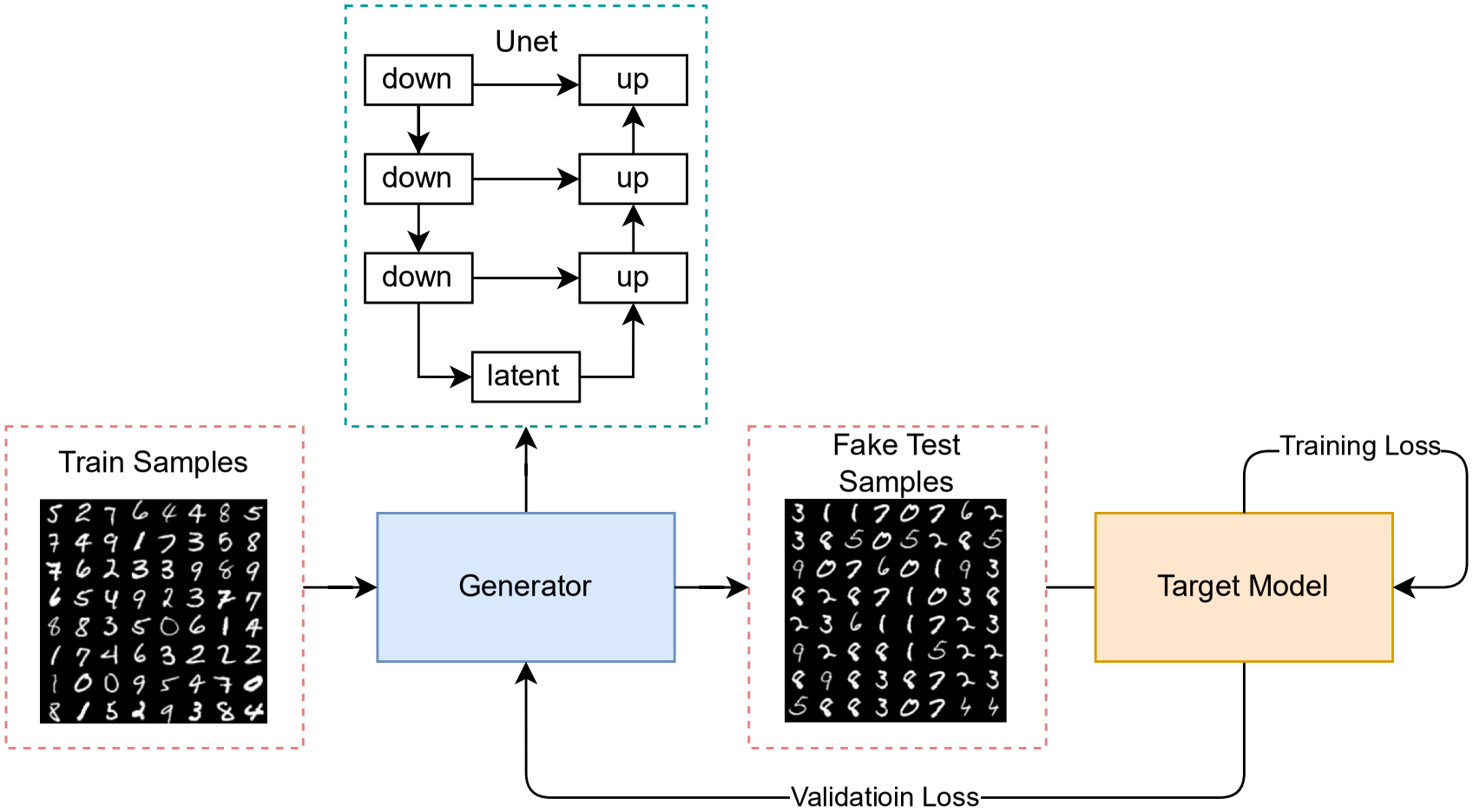


Figure 5.3: Meta Generative Data Augmentation Optimization Training Framework.

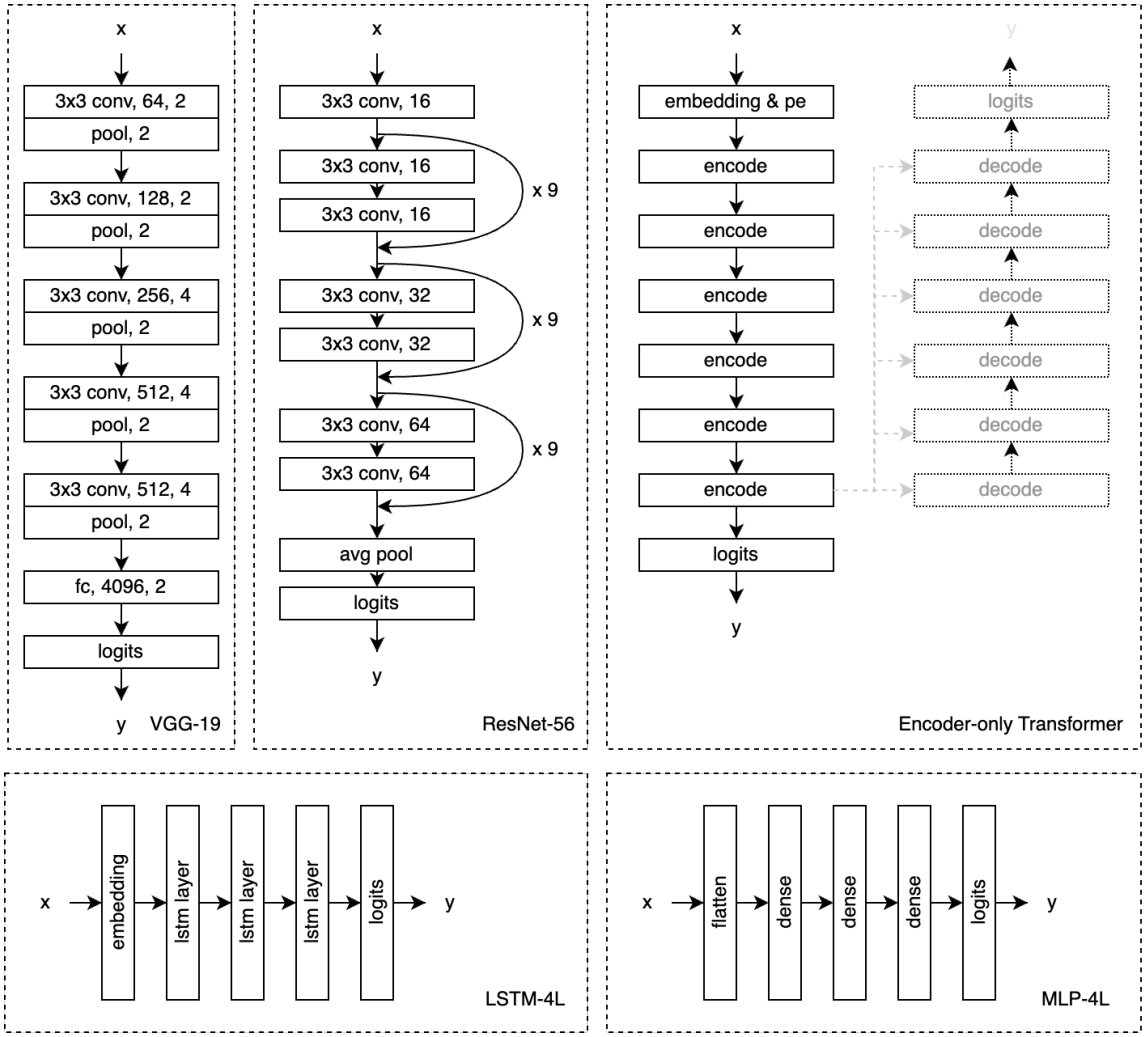


Figure 5.4: Meta Generative Data Augmentation Optimization Models for Training, includes MLP-4L, ResNet-56 for image classifation, and Transformer for text classification. Worth mentioning that we are only using the encoder part of the transformer, which is the same usage of BERT[1]

batch normalization.

5.4.2.2 Target Model

For the four-layer MLP models, the units are $[128, 64, 32, classes]$, and the activation functions are $[ReLU, ReLU, ReLU, Softmax]$. For VGG-19 and ResNet-56, we used the settings reported in.[44,73] All workers are initialized to the same weights using the $glorot$ initializer.[45] We applied a four-layer LSTM and BERT-like transformer model for the intrusion classification tasks.

The loss function is *cross entropy* loss, and the optimizer is the naive SGD optimizer with a learning rate of 0.1. For the experiments with the Learning Rate Decaying (LRD), it starts at $(50\%, 75\%)$ steps of the total epochs with a factor of 0.1. The batch size is set to be 256, and all the workers are trained for 100 epochs (MLP) and 300 epochs (ResNet-56). For the experiments with the Data Augmentation (DA), we process the images as recommended in.[44]

5.4.3 Results

(a) source domain (b) middle transform (c) target domain

Figure 5.5: MGDAO can be regarded as the transfer learning task that transformed the samples from the source domain(train samples) to the target domain(test sample).

5.4.3.1 Training Of The Generator

We first show the training for the generator in Fig 5.2. These figures are the augmented samples by the generator during the experiment. Initially, the augmented samples are unclear, however, the augmented samples gradually become explainable through interactive training with the generator and the target model.

The L_S and L_V are defined in the objective function Eq 5.4. for two reasons. The L_S controlled the shape of the augmented sample should match the train samples. The L_V controlled the evaluation behavior of the augmented samples, which means the behavior on the target model should match the true test samples. We hope the generated samples are close to the "validation" sample when evaluated by the target model.

Not only does the shape match by the l_1 loss, but we also consider matching the high-level meaning of the sample. Since we do not want the augmented shape to be exactly the same as the input, we want the augmented sample to still be explainable after the decoder. Thus, we also tested to decrease the distance of the samples in the encoded latent space in the Sec 5.5.2.

5.4.3.2 Controlling the Augmented Samples

For the generator $g(x, \hat{y}_i)$, we give it the target label $\hat{y}_i$ and a batch of train samples x, and we hope it can learn a transformation from x to the validation domain $\hat{x}$. The generator g is trained by the feature loss L_S and validation loss L_V introduced in 5.1.

We show the result of controlling for the generator to generate target samples in Fig 5.5. Where Fig 5.5(a) is the sample batched from the train set, which can be considered sampled from the source domain. Fig 5.5(c) is the augmented batch from the input batch, which can be regarded as a transfer to the target domain. Fig 5.5(b) is the middle interpolation, which denotes the transformation during the training.

Our results show that only the train samples and the target labels, the generator $g(x, \hat{y}_i)$, could generate fake samples with the same validation loss. Therefore, an intuitive idea is

whether we can use such fake validation/test samples to train the target model. We show the few-shot results in the next section.

5.4.3.3 Few Shots Learning

Since the generator G is powerful for generating a variety of "validation" samples, we want to use such fake validation samples to train the target models. Especially, we expect with such a generator, we can use a few shot settings for training the target model.

We show the train and test accuracy curves for VGG19 on the MNIST in Fig 5.6. From these figures, the "augmentation" curve means training with the generator, and the "original" curve is the target model in a normal setting. These figures suggest that results that with the generator, the target model training is faster, and the generalization is better compared to the baseline. For more details on other datasets and models, we summarized the results in Tab 5.3.

5.5 Discussion and Improvements

5.5.1 Generating from Pixel-wise Sample Space

The objective function of MGDAO Eq 5.4. is consisted of two parts, L_S and L_V

Here, we conduct pixel-wise similarity to the input train samples to the L_S to make the generated sample match the same distance to the train samples. The calculation for L_S as follow:

$$L_S = \frac{1}{N_T} \sum_{(x_i, y_i) \in D_T}^{N_T} ||g([x_i, \hat{y_i}]; \phi) - \hat{x_i}||_{l_1} \tag{5.10}$$

Where $\hat{x_i}$ is the training sample randomly obtained from the train set and $\hat{y_i}$ is its label. By applying the pixel-wise similarity, the generated sample is close to the input train samples

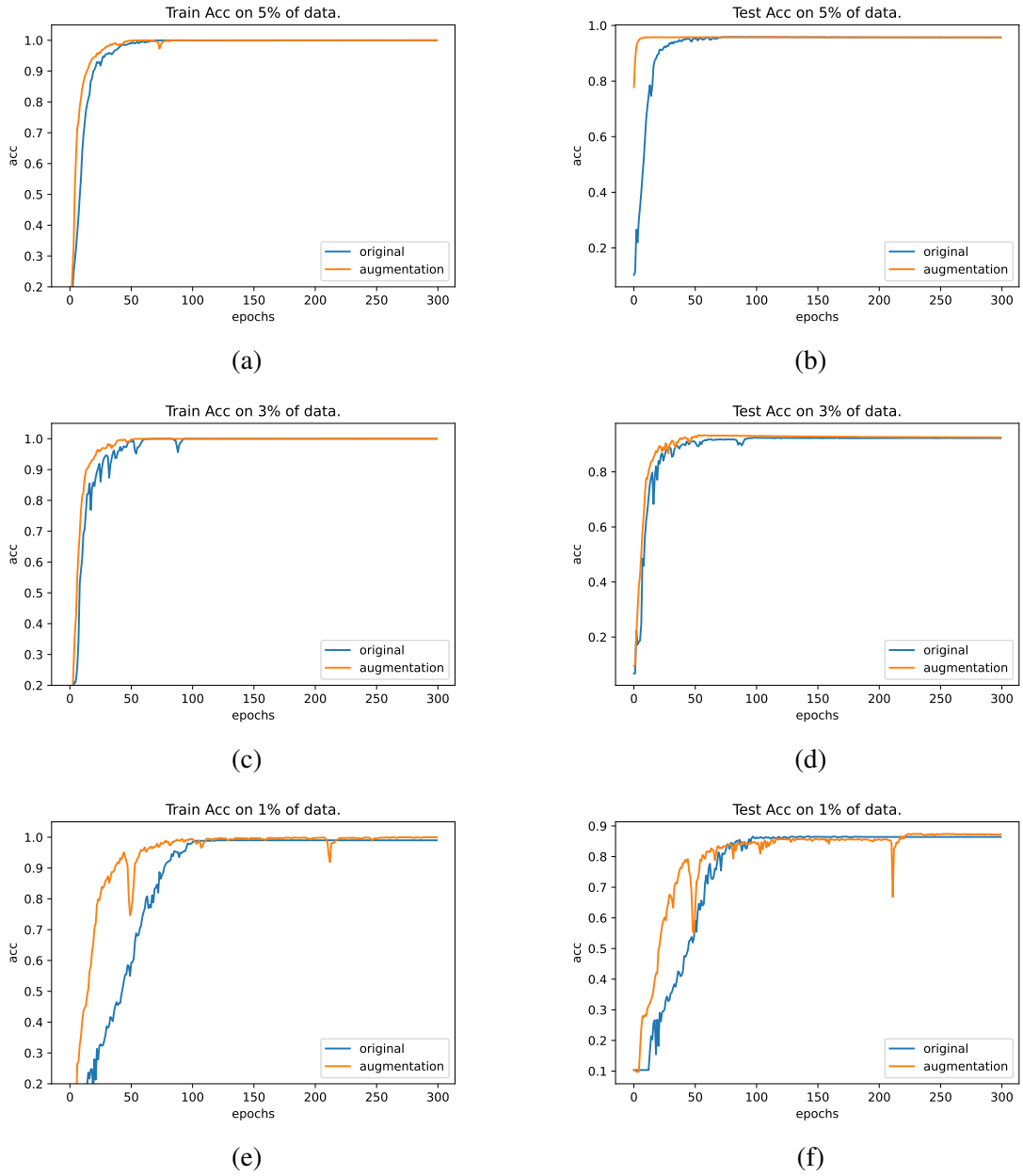
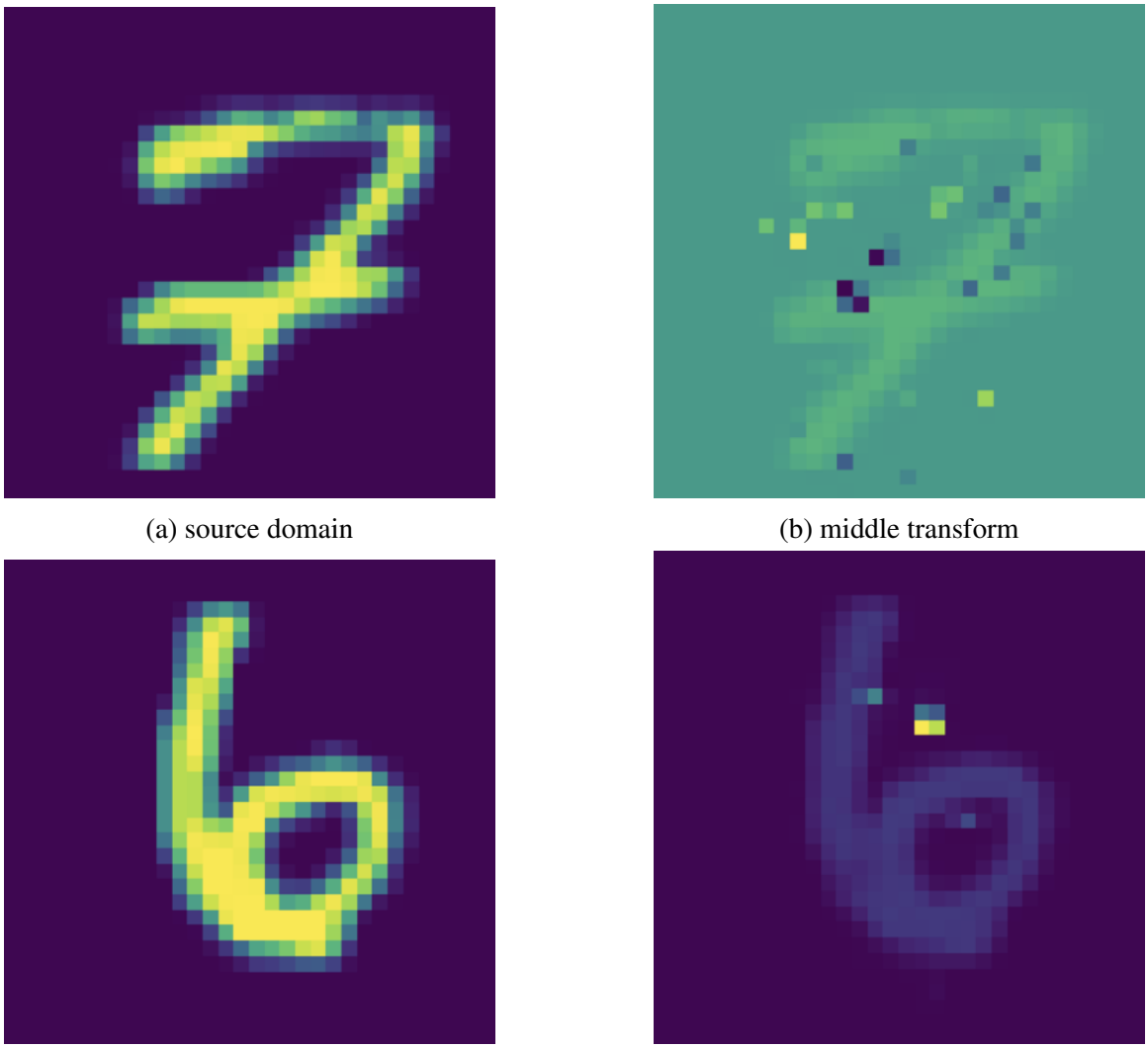


Figure 5.6: Target model(VGG19)'s training accuracy, test accuracy, and for 1%, 3%, 5% data on MNIST.

shown in Fig 5.7.



(a) source domain

(b) middle transform

(c) target domain

(d) target domain

Figure 5.7: Pixel-wise similarity to the train samples. We ”discovered” augmentation operations like random noise, color change, etc.

Despite the similarity to the input train samples, the generator could still find data augmentation operations like random noise, masking, color change, etc., to decrease the validation in L_V.

5.5.2 Image Generating from Encode Latent Space

The pixel-wise similarity helps the generated sample away from being random noise to suit the target models for the same validation loss. We hope the generated and trained samples should be closed in the latent space to release the constraint in the pixel-wise feature space. Thus, we hope the encoded generated samples and encoded train samples are close to each other. Thus, the loss in the encoded latent space is as follows:

$$L_S = \frac{1}{N_T} \sum_{(x_i, y_i) \in D_T}^{N_T} ||E(g([x_i, \hat{y_i}]; \phi)) - E(\hat{x_i})||_{l_1} \tag{5.11}$$

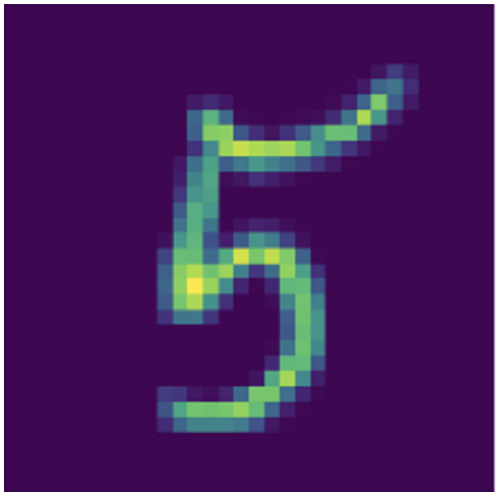
We show the result in Fig 5.8. The sample in the source domain is randomly drawn from the training dataset. The samples in the target domain are the generated samples close to the train samples in the encoded latent space and have the same validation loss as the validation samples.

5.5.3 Computation Costs

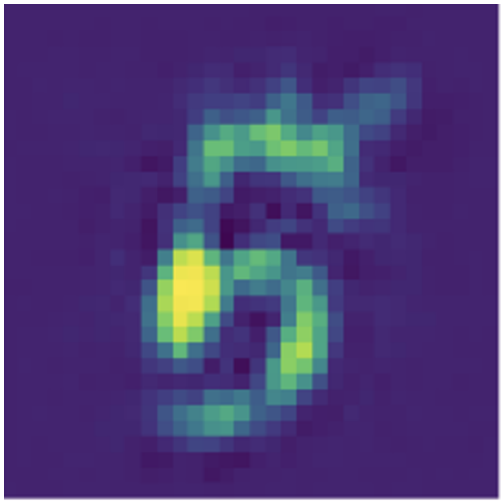
For training the target model, the extra memory cost and computation of MGDAO are mainly due to the generator, which, in our case, is the UNet. However, the generator is trained only when a new validation loss is generated. This training scheme suggests that the extra computation is not too expensive compared to the cost of training the target model.

However, the implicit cost of applying the generator as the data augmentation is expensive compared to training the generator. The batch samples need to be augmented by the generator, and it caused the training of the target model to be delayed, which is even more expensive than training the generator. To address this problem, we could use the generator to augment the total samples before the training of the target model.

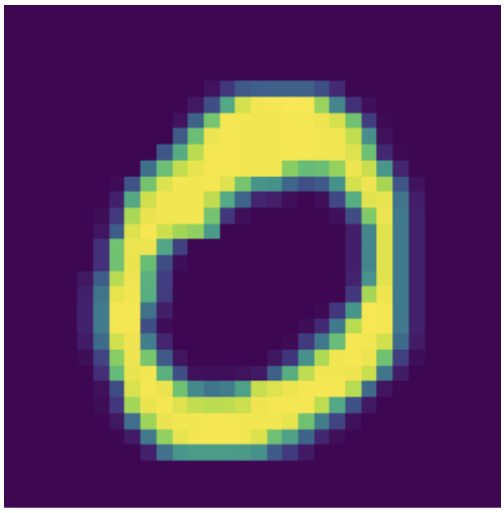
When training the ResNet-56 on CIFAR-10, we found that the time cost of baseline on 8 GPUs V100 is about 2 GPU hours and 3 GPU hours with MGDAO.



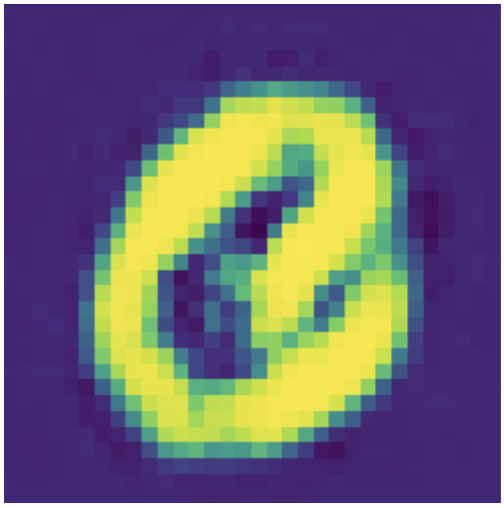
(a) source domain



(b) middle transform



(c) target domain



(d) target domain

Figure 5.8: Maintaining the similarity in the latent space, we ”discovered” shape-changing augmentation operations and have the same validation loss.

5.6 Conclusion

MGDAO replaces the limited operations space in the AutoAugment series with a deep-style generator and replaces the discriminator in a generative adversarial model with the validation loss of the target model. The generator learns to transform the data from the training domain to the validation data domain. It is further used to generate data-augmented samples to train the target model and reduce the validation loss. Experiments on few-shot image classification benchmarks show that MGDAO achieves competitive results compared to data auto-augmentation methods. MGDAO opens different directions for future inquiries:

- Invariant Feature Extraction: We discussed the pixel-wise and the encoder latent space similarity in the previous sections. We are considering a new way to extract high-level features and combine them properly to fit the validation loss.

- Fake Test Sample Reconstruct: When applying MGDAO, we could generate sample samples similar to the training samples with the same validation loss as the validation samples. However, we hope to find a way to generate the same validation sample through the target model directly.

Dataset	Target Model & Setup	Accuracy	
		baseline	MGDAO
MNIST	MLP-4L/25-shots	84.42%	85.36%
	MLP-4L/50-shots	87.77%	89.23%
	MLP-4L/100-shots	93.34%	95.18%
	MLP-4L/Fullsize	98.31%	99.72%
	VGG-19/15-shots	83.77%	85.66%
	VGG-19/25-shots	87.33%	88.21%
	VGG-19/35-shots	89.71%	91.18%
	VGG-19/Fullsize	98.52%	99.78%
CIFAR-10	MLP-4L/15-shots	48.45%	52.45%
	MLP-4L/25-shots	51.47%	53.45%
	MLP-4L/35-shots	54.71%	55.94%
	MLP-4L/Fullsize	59.32%	59.32%
	VGG-19/15-shots	50.67%	53.96%
	VGG-19/25-shots	55.77%	57.06%
	VGG-19/35-shots	58.33%	67.77%
	VGG-19/Fullsize	81.93%	82.77%
	ResNet-56/15-shots	53.58%	55.88%
	ResNet-56/25-shots	59.97%	61.32%
	ResNet-56/35-shots	67.79%	69.87%
	ResNet-56/Fullsize	93.17%	94.23%
CIFAR-100	VGG-19/15-shots	39.30%	41.42%
	VGG-19/25-shots	42.72%	43.96%
	VGG-19/35-shots	46.37%	47.18%
	VGG-19/Fullsize	51.01%	53.96%
	ResNet-56/15-shots	48.32%	52.12%
	ResNet-56/25-shots	53.07%	56.71%
	ResNet-56/35-shots	55.91%	59.32%
	ResNet-56/Fullsize	67.06%	68.64%

Table 5.2: Comparisons of the Baseline and MGDAO on the MNIST, CIFAR-10, and CIFAR-100 datasets with several target models.

Dataset	Target Model & Setup	Accuracy	
		baseline	MGDAO
ADFA-WD	MLP-4L/25-shots	72.21%	73.36%
	MLP-4L/50-shots	81.37%	82.23%
	MLP-4L/100-shots	83.34%	85.18%
	MLP-4L/Fullsize	87.31%	88.72%
	LSTM-3L/15-shots	81.77%	-
	LSTM-3L/25-shots	82.33%	-
	LSTM-3L/35-shots	84.71%	-
	LSTM-3L/Fullsize	93.52%	95.78%
ADFA-LD	MLP-4L/15-shots	74.16%	-
	MLP-4L/25-shots	79.28%	-
	MLP-4L/35-shots	81.12%	-
	MLP-4L/Fullsize	86.32%	87.78%
	LSTM-3L/15-shots	-	-
	LSTM-3L/25-shots	-	-
	LSTM-3L/35-shots	-	-
	LSTM-3L/Fullsize	-	-
	Trans-6H8L/15-shots	-	-
	Trans-6H8L/25-shots	-	-
	Trans-6H8L/35-shots	-	-
	Trans-6H8L/Fullsize	94.17%	95.53%

Table 5.3: Comparisons of the Baseline and MGDAO on the ADFA-WD and ADFA-LD datasets with several target models.

Chapter 6

Adaptive Patching for High-resolution Image Segmentation with Transformers

Attention-based models are advancing rapidly in image analytics, particularly for segmentation tasks. A typical approach involves splitting images into patches and processing these as a linear sequence of tokens through transformer encoders. However, high-resolution images, such as microscopic pathology scans, face challenges due to the quadratic growth of computational and memory requirements, especially with smaller patch sizes optimal for segmentation. One solution is employing complex multi-resolution models or approximating attention mechanisms. Inspired by Adaptive Mesh Refinement (AMR) methods in high-performance computing, we introduce adaptive image patching as a pre-processing step, reducing the number of patches by orders of magnitude based on image detail. This approach incurs minimal overhead and integrates seamlessly with any attention-based model. Our method achieves superior segmentation performance compared to state-of-the-art models on real-world pathology datasets, while delivering a geometric mean speedup of $6.9\times$ for resolutions up to $64K^2$ across $2,048$ GPUs. [1]

[1]This chapter was published as E. Zhang, I. Lyngaas, P. Chen, et al. Adaptive Patching for High-resolution Image Segmentation with Transformers, 2024 The International Conference for High Performance Computing, Networking, Storage, and Analysis, ATLANTA, GA.

6.1 Introduction

Vision Transformers (ViTs) have recently revolutionized computer vision by achieving exceptional results in image classification.[74–79] To address dense prediction tasks, such as segmentation, numerous adaptations of ViTs have been proposed,[80–83] while some approaches combine transformers with U-Net like architectures.[3,81,84,85] Processing high resolution images using ViTs introduces scalability constraints, especially when handling small patches arranged in long sequences of intricate medical imaging data.[3,86] The complexity of handling extensive sequences arises from the quadratic cost of self-attention, imposing significant computational overhead.[87] As a result, conventional ViTs struggle to process high-resolution images efficiently, where detailed content spans wide spatial contexts.

Two main strategies attempt to address this scalability issue, though neither reduces the total computational burden. Sequence parallelism distributes long sequences into smaller segments across multiple processing units (e.g., GPUs).[88–91] Alternatively, blocking/tiling techniques partition the attention matrix into smaller sub-matrices that fit within user-managed cache memory. Examples include FlashAttention[92,93] and Swin Transformer,[81] which processes images through overlapping windowing techniques. While blocking/tiling enables memory-efficient scaling for longer sequences, it does not reduce overall computational effort.

In contrast, there are two other approaches that address the long-sequence scaling problem by reducing the amount of work (not necessarily specific to vision transformers): attention approximation and hierarchical training.

Approximation attention approaches approximate the self-attention mechanism through spectral attention,[94–96] low-rank approximation,[97,98] sparse attention matrix sampling,[99–103] infrequent self-attention updates,[104,105] or their combinations.[106] Approximation methods greatly reduce the memory and computation cost, some of which reduce the quadratic complexity of self-attention to be linear. Yet, loss of information due to approximating the self-

attention could have negative impact on accuracy, especially for long-range sequences. Experiments show a notable drop in accuracy when the compression ratio surpasses 70%.[107] Finally, implementing approximation approaches is complex and often requires custom operators and sparse formats.

Hierarchical training of ViTs comprises multiple transformers being trained at different levels of resolution.[5,83,108,109] Training begins with the lowest-level transformer processing short sequence segments. Higher-level transformers iterate on using outputs from lower levels to process longer segments. However, employing multiple transformers increases the training time and memory usage. Moreover, managing multiple interacting transformers is complex, demanding hyperparameter tuning for the model at each resolution level.

In summary, to scale long sequences for high-resolution image segmentation trained on ViT models or U-Net models that use transformers to ingest the images, we need the following: a) be able to use smaller patch sizes that are favorable in segmentation,[86] b) avoid the potential loss in performance that comes with self-attention altering mechanisms, c) avoid the high aggregate compute cost of sequence parallelism and tiling/blocking methods, and d) have a general solution that can work with transformer model, and not custom built models.

To address these challenges, we draw inspiration from tree-based Adaptive Mesh Refinement (AMR) techniques[110] widely utilized[111,112] for decades in high-performance computing to reduce computational costs for structured mesh-based solvers. We propose an Adaptive Patch (AP) method, designed to integrate seamlessly with vision transformers. AP serves as a pre-processing solution employing a quadtree structure to partition images into mixed-scale patches, adapting to the varying detail levels across regions. Larger, less detailed patches are downscaled, ensuring all patches are resized uniformly for the model while preserving the core vision transformer architecture and attention mechanisms.

To demonstrate scalability, we trained transformer-based models extensively, leveraging small patch sizes for lengthy sequences of high-resolution images. The primary contri-

butions of this work are summarized below:

Adaptive Patching: A strategy to minimize the number of patches extracted from images, significantly reducing training cost. This approach lowers memory usage and computation demands while allowing for smaller patches (e.g., 4×4 or 2×2), which enhances segmentation accuracy.[86] Results show that at resolutions $[512, 1024, 4096, 8192, 16384]$, models using AP achieve nearly $8\times$ smaller patches or $64\times$ longer sequence lengths without increasing costs compared to uniform patching.

High-quality segmentation for real-world datasets: Using the Frontier supercomputer with up to $2,048$ MI250X GPUs, we conducted experiments on high-resolution pathology datasets. With a fixed compute budget and up to 13 resolution levels, we scaled to $16K^2$ image resolutions, reducing patch sizes from 16×16 to 2×2 on vision transformers. These smaller patches improved segmentation quality by 5.5% compared to widely-used models. Alternatively, achieving the same segmentation quality resulted in speedups ranging from $12.7\times$ to $3.9\times$. AP also boosted classification accuracy by more than 7% compared to the most advanced model for high-resolution microscopic pathology classification.

Simplicity and efficiency: Unlike methods requiring attention mechanism modifications, AP preserves the original architecture, enabling straightforward integration with any vision transformer. This low-overhead pre-processing solution becomes negligible as training progresses, making it highly efficient.

In conclusion, AP offers an innovative, generalizable solution for managing long sequences in vision transformers. It retains dense self-attention benefits while significantly reducing sequence length, enhancing segmentation efficiency. This advancement broadens the potential applications of vision transformers in high-resolution scientific imaging.

6.2 Background and Motivation

6.2.1 Adaptive Mesh Refinement and Quadtrees in Imaging

Structured AMR[110] uses a hierarchical spatial representation of mesh spacing. In the 2D tree-based scheme, the mesh is organized into a hierarchy of refinement levels in a tree that represents the hierarchy of the mesh. The mesh is usually decomposed into relatively small fixed-sized quadrants of mesh cells. Each quadrant can be recursively refined into a set of quadrants of fine cells. A quadtree manages the mesh by maintaining explicit child-parent relationships between coarse and fine quadrants. At most one level of refinement difference is typically allowed between neighboring quadrants to maintain size relations. Traversing the quadrants across the three leaves corresponds to a Morton z-shaped space filling curve in the geometric domain.[113] Accordingly, sorting the tree leaf blocks by their Morton ID would give a series of blocks that are affine in the geometric space of the mesh.

A similar concept appears in computer graphics, under the name of *quadtrees*, where a mesh is replaced by an image, and mesh cells are replaced by image pixels. The history of quadtree structures dates back to early advancements in computer graphics and image processing.[114,114,115,115–117] In the work of,[114,115] quadtrees (octrees) were used in 2D (3D) computer games to detect the collision of two objects efficiently in $O(n \log n)$ time complexity, where n is the number of particles. Quadtrees are also used as an image representation at different resolution levels and have been efficiently applied in image[114] and video compression.[115] Recently, quadtrees have been used in image segmentation to improve attention efficiency, e.g., quadtree attention,[118] and octree transformer.[119] Both of those approaches employ quadtrees, like the work in this chapter. However, we introduce a quadtree-based pre-processing patching strategy without changing the model or attention scheme. In other words, our proposal doesn't involve additional complexity and custom model design; our solution can be integrated seamlessly into the current and future transformer-based encoders.

6.2.2 Vision Transformers and Attention

Our proposed methods act as a pre-processing step to feed patches to vision transformers, or U-Net[120] like models employing transformer encoders. ViTs[75] comprise an embedding layer, transformer encoder layers, and a classification head. The embedding layer linearly projects the image patches sequence input into a sequence of flattened embeddings. Transformer encoder layers process these embeddings, capturing local and global context through self-attention mechanisms.

The attention mechanism in transformers computes attention scores A between input tokens, forming the attention matrix. Let $x \in R^{N\times F}$ denote a sequence of N feature vectors of dimensions F. A transformer is a function $T : R^{N\times F} \rightarrow R^{N\times F}$ defined by the composition of L transformer layers $T_1(\cdot), ..., T_L(\cdot)$ as follows,:

$$T_l(x) = f_l(A_l(x) + x). \tag{6.1}$$

$A_l(\cdot)$ is the self-attention function. The function $f_l(\cdot)$ transforms each feature independently of the others, and is usually implemented with a small two-layer feedforward network. Formally, the input sequence x is projected by three matrices $W_Q \in R^{F\times D}, W_K \in R^{F\times D}$, and $W_V \in R^{F\times D}$, to corresponding representations Q, K and V. Thus, the attention scores are calculated as follows:

$$Q = xW_Q \tag{6.2}$$

$$K = xW_K \tag{6.3}$$

$$V = xW_V \tag{6.4}$$

$$A_{ij} = \text{Softmax}\left(\frac{(Q_i K_j)^T}{\sqrt{d_k}}\right) \tag{6.5}$$

where Q_i and K_j are query and key vectors for tokens i and j, and d_k is the dimension of the key vectors. The complexity of the attention matrix is $O(N^2)$, where N is the sequence

length. The same is true for the memory requirements because the full attention matrix must be stored to compute the gradients for the weights of the queries, keys, and values.

We further assume that the input is the content of a square image x with a resolution of Z, that is, let $x \in R^{Z \times Z}$, and by assuming that patches arise from the uniform grid patch method of patch size p. Thus the sequence $N = (\frac{Z}{P})^2$. Therefore, the total computation and memory cost of attention scores according to resolution and patch size is $O([\frac{Z}{P}]^4)$. This complexity demonstrates the difficulties of increasing the resolution while decreasing patch size P with the uniform grid patch strategy.

6.2.3 Long Sequence Problem

Due to the quadratic cost of transformers w.r.t. the sequence length, numerous efforts have been dedicated to overcoming the long-sequence problem by reducing the amount of work. The first approach questions the necessity of full attention between all input embedding pairs.

Longformer[102] introduced a localized sliding window-based mask with few global masks to reduce computation scales linearly with the input sequence. Child et al.[121] proposed a set of sparse attention kernels that reduces the complexity to $O(n\sqrt{n})$ and saves memory usage of the backward pass. Reformer[122] further reduces the complexity to $O(n \log n)$ based on locality-sensitive hashing. ETC[123] uses local and global attention instead of full self-attention to scale transformers to long documents. BigBird[124] is closely related to and built on the work of ETC. Linformer[125] assumes the self-attention is low rank, and also proposes a linear complexity transformer. Later, Performers[126] also achieved linear space and time complexity and did not rely on any priors such as sparsity or low-rankness.

The second approach reduces the attention computation by training a hierarchy of models at different resolutions. Hierarchical transformers for text classification[108] use three models to capture the structure in long sequences in documents. CrossViT[83] classifies images by running a dual-attention model, in which each branch creates a non-patch to-

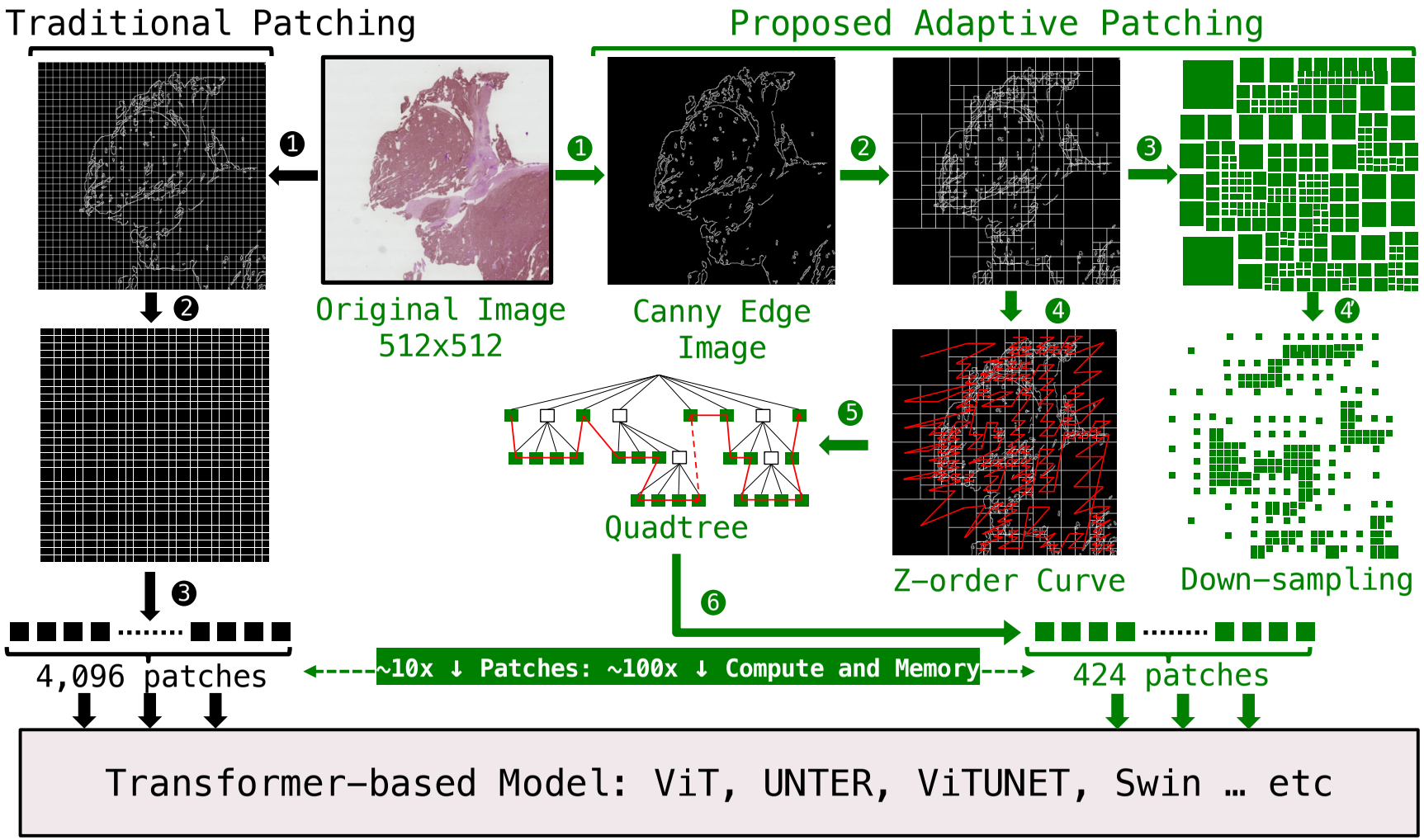


Figure 6.1: Overview of AP. The right-side flow (green) shows all the steps, starting from the original image, and ending up with feeding the patches (tokens) to an intact transformer-based model. The reduction from 4,096 to 424 patches (of size 4×4) while achieving the same dice score is from a real example of training 512×512 images from the PAIP[2] liver cancer dataset on the UNTER[3] model: $\sim 9.6\times$ reduction in sequence length, and $\sim 12.7\times$ speedup in end-to-end training.

ken to exchange information with the other branch by attention. HIPT[5] is a classifier for high-resolution images that trains multiple models at different resolutions to leverage the hierarchical geometric structure of visual tokens. The highest resolution model is trained with large patch sizes to reduce the sequence length. MEGABYTE[109] predicts patches of bytes by running local and global models at different patch sizes.

We summarize the core idea of each approach and their limitations in Table **??**. Hierarchical and attention approximation methods exploit the hierarchy and sparsity of the features inside the model. On the other hand, our solution is a lightweight mechanism that exploits the hierarchy and sparsity of features at different resolutions directly on the images in a pre-processing step, which leaves the attention mechanism and the model architecture intact.

6.2.4 High-Resolution Segmentation

High Resolution (HR) aggravates the long-sequence problem. Initially, the common way in literature to handle this problem was to rely on a convolutional input encoder, which first down-samples the image to learn low-resolution features[127,128] and then up-sample to complete the prediction.[129] To benefit from the effective entire-image receptive field of transformers, many efforts turned to transformer encoders (as pure ViT or CNN+ViT), and resorted to the techniques mentioned in the previous section for handling the long sequence problem. HRViT,[130] HRFormer,[131] and HRNet[132] learn the HR representations by cross-resolution stream. Vision-LongFormer[133] uses a pyramid-like hierarchical structure of models at different scales to combine local attention and global memory. HIPT[5] also applied a hierarchical pyramid transformer to a pathology dataset with the utmost $4K^2$ resolution. However, in comparison to these models, our method is a pre-processing strategy, which doesn't require additional revision to of the model or attention design.

6.3 Adaptive Patching for High-resolution Segmentation

Figure 6.1 gives an overview of the flow of AP, in comparison to the traditional method of dividing images uniformly into equal-sized patches. AP divides the image into patches of different sizes based on the level of details, and then downsamples the large patches so that all patches have the same size. In the next section, we follow the flow of AP starting from the original image up until the patches are fed to the model.

6.3.1 Quadtree-based Adaptive Patches

Image and Patches We use the following notation to distinguish the size of an "image" and the "patch" corresponding to that image. Consider an image dataset D consisting of input images $x \in R^{Z \times Z}$ where Z is the resolution of image x. Then, the sequence of non-overlapping patches can be noted as $\{x_i\}_{i=1}^{N} \in R^{N \times P}$ where N is the sequence length and P is the patch size. For the traditional uniform grid patching in ViT,[75] the sequence length is $N = (\frac{Z}{P})^2$. For an image x with resolution $Z = 512$ (i.e. the image is 512×512) and patch size $P = 8$ (i.e. the patch is 8×8), the sequence length N is 4096 patches (tokens).

Edge Extraction To ignore the irrelevant details in the images x in AP, as shown in step 1. of Figure 6.1, we apply Gaussian Blur with kernel k and Canny[134] edge detection with lower t_l and higher threshold t_h to the original input images x. The Gaussian blur smooths the irrelevant details, and the Canny edge detection extracts the grayscale edges x_e of the image. The kernel k and threshold t can also be used as hyper-parameters for controlling the smoothing effect. During our experiments, we kept the threshold as $[100, 200]$; the kernel size is set to be $[3, 3, 5, 7, 9, 11, 13]$ for resolutions $[512, 1024, 4096, 8192, 16384, 32768, 65536]$, respectively.

Quadtree Patches The input edge x_e undergoes a recursive quadtree partitioning shown by step 2 of Figure 6.1, creating nodes Q_h that represent specific regions where h is the

depth of the quadtree. The quadtree node Q_{h+1} is defined recursively as follows:

$$Q_{h+1} = \begin{cases} Q_h & \text{if } \sum_i D_i \leq v \text{ or } h = H \\ \{Q^h_{\text{NW}}, Q^h_{\text{NE}}, Q^h_{\text{SW}}, Q^h_{\text{SE}}\} & \text{if } \sum_i D_i > v \text{ and } h < H \end{cases} \tag{6.6}$$

where H is the maximum quadtree depth, v is the subdivision criterion, $Q^h_{\text{NW}}, Q^h_{\text{NE}}, Q^h_{\text{SW}}, Q^h_{\text{SE}}$ are the h-th depth child nodes representing the northwest, northeast, southwest, and southeast quadrants, respectively.[135,136] In our implementation, the subdivision criterion constraints the total number of pixels $\sum_i D_i$ confined in the data area by the split value v. The depth limitation H is set to $[9, 10, 12, 13, 14, 15, 16]$ w.r.t. resolutions, which practically allows the input x_e to be subdivided all the way down to the 2×2 patch size level.

For uniform grid patches, we concatenate horizontal lines of patches into a 1D sequence of patches. On the other hand, for adaptive patching, after the quadtree is constructed, the patches, that is, the leaf nodes, need to be arranged. Here we show by steps 4 and 5 of Figure 6.1, we use a Morton Z-order curve[113] to arrange the nodes starting from the left end of the tree and going to the right. Z-order curves have the desirable property of keeping geometrically affine patches closer in the constructed sequence. After arranging the patches, since different images have different quadtree sequence lengths, firstly, we project all the different patches into the same minimized size P_m (step 4' of Figure 6.1). Next, we randomly drop or pad them to the same length L. Finally, the sequence of patches $x_p \in R^{L \times P_m}$ are fed to the model (step 6) of Figure 6.1) to train any underlying segmentation model using a transformer encoder $f(x_p; \theta)$. We summarize the above steps in Algorithm 2.

It is worth mentioning that for quadtree in the worst case, where all objects and details are in the same quadrant at the deepest level of the tree, the time complexity becomes $O(N^2)$. In the best cases, the quadtree patching strategy leads to $O(\log^2 N)$, where N is the total number of patches. However, from empirical observations in pathology datasets, instead of the best or worst cases, we observed sub-linear growth in sequence length as

Algorithm 2 Adaptive Patch.

Require: $v, H, k, t_l, t_h, f(x;\theta), N, T, D, D_p$

Initialize segment model $f(x;\theta)$.
for $n \leftarrow 1$ **to** N **do**
    $x_g = GaussianBlur(x_n; k)$
    $x_e = CannyEdge(x_g; (t_l, t_h))$
    $x_p = QuadTreePatch(x_g; v, H)$
    Add to $D_p = D_p \cup (x_p, x_n)$
end for
for $t \leftarrow 1$ **to** T **do**
    for $n \leftarrow 1$ **to** N **do**
        $x_p = D_p.pop()$
        Train $f(x;\theta)$ on the x_p
    end for
    Evaluate $f(x;\theta)$ on validation set
end for
Evaluate $f(x;\theta)$ on Test set

the average patch size decreased. This linear complexity in sequence length suggests the empirical time complexity is approximately $O(n)$.

6.4 Evaluation

6.4.1 Experimental Setup

All the experiments were performed using the Frontier Supercomputer[137] at ORNL. Each Frontier node has a single 64-core AMD EPYC CPU and four AMD Instinct MI250X GPUs (128GB per GPU). The four MI250X GPUs are connected with Infinity Fabric GPU-GPU of 50GB/s. The nodes are connected via a Slingshot-11 interconnect with 100GB/s, to a total of 9,408 nodes. For the software stack, we used Pytorch 2.4 nightly build 03/16/2024. ROCm v5.7.0, MIOpen v2.19.0, RCCL v2.13.4 with libfabric v1.15.2 plugin.

6.4.2 Datasets

PAIP[2] is a high-resolution liver cancer pathology (real-world) dataset. The sample resolution size is close to $64K$, far higher than the resolution of conventional image datasets. PAIP includes $2,457$ Whole-Slide Images (WSIs). When needed to use smaller resolutions, we down-scale the images into uniform $[512, 1,024, 4,096, 8,192, 16,384, 32,768]$ square images. Before applying our quadtree patching method, we first apply Gaussian smoothing with kernel size 3×3 and $\sigma = 0$. Then, we used Canny edge detection with a lower/higher threshold of $[100, 200]$ to extract the edges from the smoothed input. During the training process, we randomly select 0.7 samples for training, 0.1 samples for validation, and 0.2 samples for testing. All data sets are shuffled and normalized to $[0.0, 1.0]$ when used as model input.

BTCV challenge[4] for 3D multi-organ segmentation contains 30 subjects with abdominal CT scans where 13 organs are annotated by experts. Each CT scan consists of 80 to 225 slices with 512^2 pixels. The multi-organ segmentation problem is formulated as a 13 classes segmentation task where the dice score typically reported is the average of the 13 classes. BTCV is relatively low in resolution in comparison to the PAIP dataset (512^2 vs. $64K^2$), yet is widely used as a benchmark by the high-resolution medical segmentation community.

6.4.3 Models

Because our method is a patching strategy, it can easily replace the uniform grid patching method typically used in transformers. In our experiments, we use one of the widely-used models, UNETR,[3] as the baseline model we use for AP to conduct experiments on the high-resolution medical image segmentation task. It is worth noting that in all our results we train the model from scratch for the target dataset: we do not do any pre-training on other datasets or fine-tune. We also report results for various other highly performing models

as baselines, TransUnet,[138] HIPT,[5] Swin UNETR,[139] ViT,[75] and U-Net,[120] to demonstrate different aspect about the performance and efficiency of AP.

UNETR uses a contraction-expansion pattern consisting of several transformers as an encoder. It is connected to the decoder via a skip connection. UNETR's initial target application was 3D medical imaging for human organs. The original work[3] also discussed the impact of patch size on the model: the smaller the patch size, the better the model performance will be. However, due to the memory capacity and compute power limitation associated with quadratic attention, the authors reported that conducting experiments with a small patch size is unfeasible. Since our target experimental data is 2D medical images, we only swap the 3D convolution and deconvolution blocks in UNETR with the 2D version without additional changes to the model structure. Other than that, we make no changes nor do we tune the original UNETR model.

The encoder is a transformer stack of 12 blocks comprising Multi-head Self-Attention (MSA) and Multi Layer Perceptron (MLP) sublayers. The embedding size is $K = 768$. After the encoder, UNETR extracts a sequence representation from each transformer block $z_i, [i \in 3, 6, 9, 12]$ and reshapes it to a sequence of patches. Then, similarly to the idea of U-Net, we apply a deconvolutional bottleneck layer to the transformed feature map to increase its resolution by a factor of 2. Then, output concat with $z_i, [i \in 3, 6, 9, 12]$ sent to 3×3 convolutional layer and upsampling deconvolution. This process continues until the output size meets the mask size. The final layer is 1×1 convolutional layer to project the class to the channel of the mask.

6.4.4 Training Setup

The loss function we applied is a combination of dice loss and binary cross-entropy loss:

$$
\begin{aligned}
L(\hat{y}, y) =& w \cdot L_{bce}(\hat{y}, y) + (1 - w) \cdot L_{dice}(\hat{y}, y) \\
=& - w \cdot \frac{1}{N} \sum_{i=1}^{N} [y_i \log(\hat{y}_i) + (1 - y_i) \log(1 - \hat{y}_i)] \\
& + (1 - w) \cdot (1 - \frac{2 \sum_{i=1}^{N} (\hat{y}_i \cdot y_i) + \epsilon}{\sum_{i=1}^{N} \hat{y}_i + \sum_{i=1}^{N} y_i + \epsilon})
\end{aligned}
\tag{6.7}
$$

where $L(\hat{y}, y)$ represents the combined loss function, composed of a weighted sum of Binary Cross-Entropy (BCE) loss and dice loss. w is the weight parameter controlling the contribution of BCE loss versus the dice loss:, we set it to 0.5 during the experiments. ϵ is a smoothing term, and we keep it to 1.0 during the experiments. For the resolutions $[512, 1024, 4, 096]$, all models were trained with a batch size of 16, using the AdamW optimizer[140] with an initial learning rate of 0.0001 for 300 epochs and decay by a factor of 0.1 at epoch step $[500, 750, 875]$. For the resolutions $[8, 192, 16, 384, 32, 768, 65, 536]$, we countered the problem of fitting a single sample in memory by tuning the sequence length and training for 200 epochs.

6.4.5 Evaluation Metrics

For computational performance, we report the seconds/image of end-to-end training. For the quantitative evaluation of the segmentation result, we use the dice score, which measures the similarity between a predicted segmentation mask and the ground truth segmentation mask. The dice score (also known as the dice similarity coefficient) is defined as:

$$
\text{Dice(X,Y)} = \frac{2 \times |X \cap Y|}{|X| + |Y|}
$$

where X and Y are the two sets being compared. $|X \cap Y|$ represents the cardinality of the intersection of sets X and Y. $|X|$ and $|Y|$ represent the cardinality of sets X and Y respectively. A dice score of 100% means identical similarity between the prediction and the ground truth.

6.4.6 Results

6.4.6.1 Speedup of End-to-end Training at the Same Segmentation Quality

In Table 6.1 we show that under the same dice score, AP is just a pre-processing step (on top of UNTER as baseline) that achieves a geomean speedup of $4.1\times$, if we compare on the basis that both AP and the baseline run to the same number of epochs. Since we further observe the convergence speed in AP to be $1.7\times$ faster, the speedup to get to the same dice score goes up to the geomean speedup of $6.9\times$. At the highest resolution of 64^2 training on 2,048 GPUs, AP achieves $\sim$4$\times$ speedup. It is worth mentioning that AP also brings significant savings in memory and not just speedup.

6.4.6.2 Gain in Segmentation Quality

Table 6.2 shows segmentation improvement over different models, at different PAIP resolutions. At similar resolution, with adaptive patches we can use nearly $8\times$ smaller patch sizes at the same, computational complexity, and improve upon the original model dice score with an average of 5.5%. It is worth noting that on top of improving the dice score, we achieve those improvements with additional speedups to the training time up to $4.6\times$.

Table 6.3 shows segmentation results for BTCV (512^2 resolution). Following,[84,141] we applied AP to each 2D slice of each CT sample and inferred all the slices to reconstruct the final 3D predictions. As shown in the table, AP-UNTER gives higher quality than other models, with the exception of Swin UNTER (which has the advantage of being pre-trained on five other datasets before fine-tuning on BTCV). On top of getting the highest dice score, this is achieved at $>8\times$ faster training time over models with similar dice score.

6.4.6.3 Segmentation Qualitative Results

We demonstrate the quality of segmentation at different resolutions using different models: TransUNet, U-Net, UNETR, and our proposed AP-UNETR. We summarize the real results

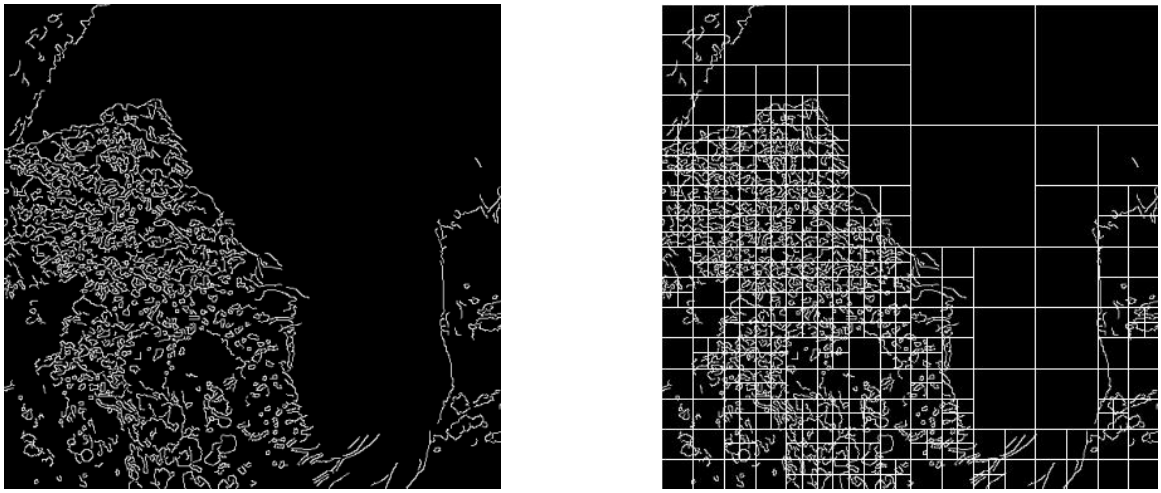
of the mask and display them in Figure 6.2. The first column shows the original input, where the label is the resolution and the scaling percentage we use to show a portion of the image. The reason why we use a fixed acquisition ratio is that it can be better displayed. At large resolutions, using a smaller patch size can lead to a more accurate prediction of details.

In our experiments, the window is uniformly set to 512×512, so for the resolution $[512, 1024, 4096, 8192, 16384, 32768, 65536]$, the proportion of the sampling window is $[100\%, 25\%, 1.56\%, 0.39\%, 0.024\%, 0.006\%]$. The second column shows the ground truth, followed by the prediction results of different models. It can be seen that at 512 resolution, small patches cannot fully express the subtle differences. However, for high-resolution images, the deviations in subtle details will become larger and larger. This difference comes from the TransUNet and UNETR models. At higher resolutions, uniform grid patching can only allow for very large patch sizes, such as $16K^2$ patch size with UNTER at input image of 64^2 resolution. The size is 512×512, and this is the size of our sampling window. However, at the same input image resolution of $64K^2$, AP-UNETR, can still use patch sizes as small as 8^2 in areas of detail by having more depth in the tree. This is the core benefit of adaptive patching.

6.4.6.4 Classification: AP vs. HIPT[5]

To demonstrate the versatility of AP, we compare classification for the PAIP dataset with the top performing and most sophisticated hierarchical multi-resolution model designed specially for microscopic pathology classification: HIPT.[5] In this experiment, we divided the PAIP dataset, designed originally for segmentation, into six categories according to organs. Each category contains 40 samples, 28 of which are used for model training, 8 for testing, and 4 for validation. For HIPT, we resize all samples to three resolution scales $[256, 1024, 16384]$ and set the patch size for each scale to $[16, 256, 4096]$ according to the original settings. For the AP method, we only applied a level $16,384$ image for the classi-

Figure 6.2: Example of segmentation quality for PAIP dataset. From $4K^2$ to $64K^2$ we zoom-in to show a portion of the image.



(a) Edges of Sample (b) Quadtree Patches

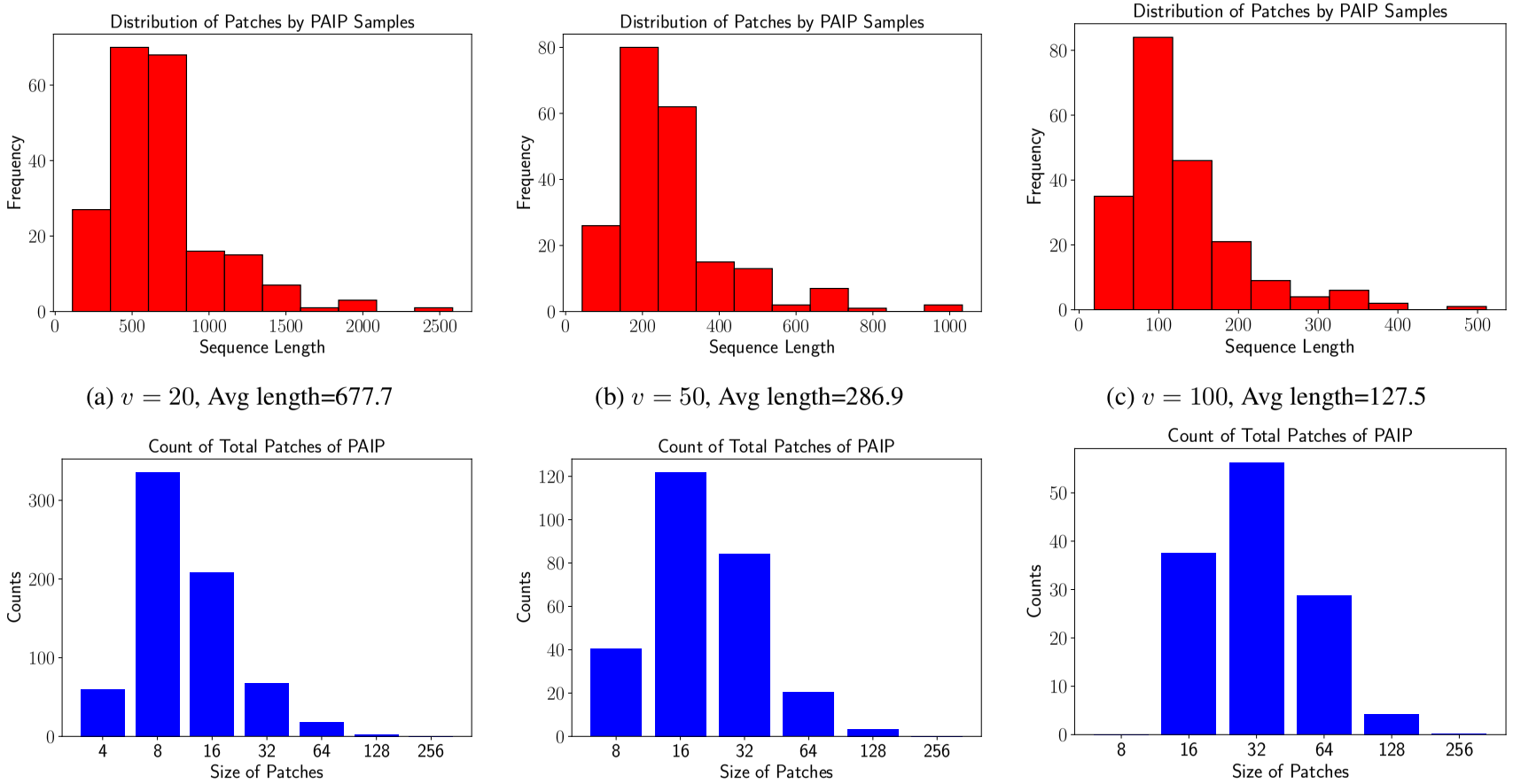
Figure 6.3: The linear and quadratic cases for the quadtree patches. In the worst case, quadtree behaves like a uniform grid patching method in the rich edge area (left bottom).

fication; instead of using a decoder for segmentation work, we added an additional output channel for the class prediction. As seen in Table 6.4, with the same compute budget, using AP with a vanilla ViT gains a huge improvement in accuracy ($>7\%$) over the very well-tuned and highly customized HIPT. At high-resolution ($16K^2$), the smallest patch size HIPT can handle, before going OOM, is $4,096^2$. AP on the other hand can go down to patches of size 2^2 at the regions of highest resolution in the images. This big gain in accuracy, despite using a vanilla ViT with AP, indicates: a) the effectiveness of AP, and b) that smaller patch sizes matter more than the sophistication of the model.

6.4.7 Discussion

6.4.7.1 Adaptive Patches Empirical Growth Complexity

The core reason why AP can handle small patch sizes at high resolutions is that the sequence size can be reduced with adaptive meshing. In Figure 6.4, we show the extent to which the sequence length can be reduced by adjusting the split value of the quadtree, without significantly losing prediction performance. The split value v controls the total length



(a) $v = 20$, Avg length=677.7 (b) $v = 50$, Avg length=286.9 (c) $v = 100$, Avg length=127.5

(d) $v = 20$, Avg size=9.37 (e) $v = 50$, Avg size=20.21 (f) $v = 100$, Avg size=30.73

Figure 6.4: Average quadtree patch size $[9.37, 20.21, 30.73]$ of training images in PAIP lead to empirical linear scaling of the corresponding average sequence length $[677.7, 286.9, 127.5]$, for different split values $[20, 50, 100]$.

and distribution of patch sizes. The first row in Figure 6.4 shows that when the split value is halved $[100, 50, 20]$, the patch size distribution or the average patch size $[9.37, 20.21, 30.73]$ is also close to being halved. This means the average patch size grows linearly with the split value. For the uniform grid patching strategy, the sequence length grows by $O(\frac{Z}{P})^2$. However, we observed an approximately linear increase in the average sequence length as the average patch size decreased. Note that AP sequence length depends on the complexity of the image itself, while the best case attention complexity is $O(log^2 N)$, the worst case would be $O(N^2)$ (it becomes like uniform grid patching). For example, the real scene image in Fig 6.3 will grow fast, the same as uniform grid patching strategy $O(\frac{Z^2}{P^2})$. Based on these observations, we can bound the length growth rate below:$[677.7, 286.9, 147.5]$ and $[9.37, 20.21, 30.73]$.

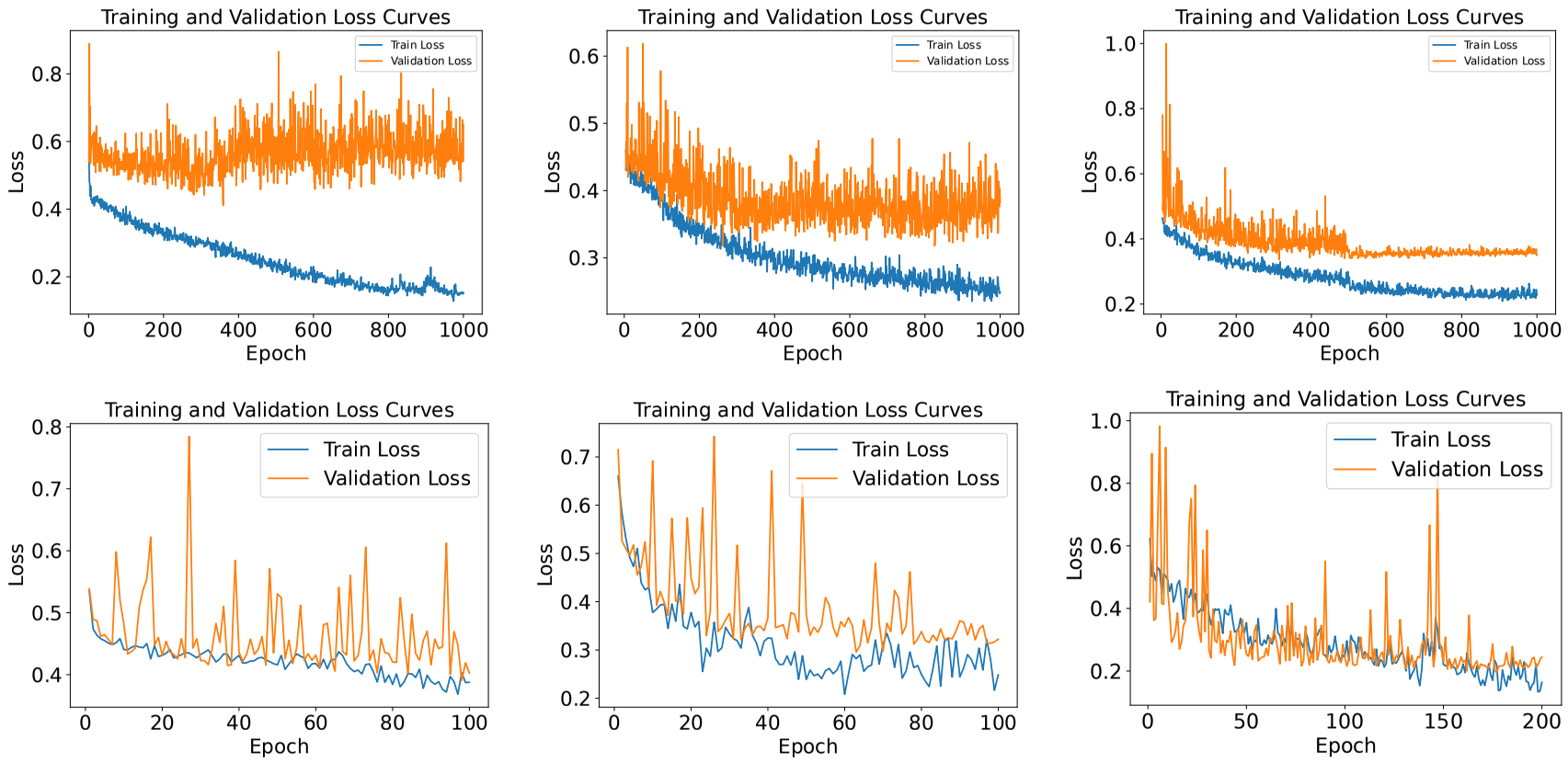
Theorem 7. *Let Z be the resolution of the image, $P = \frac{1}{N}\sum_{n=1}^{N} p_n$ be the average patch size, and L be the patch length. Then, the growth rate of L according to the Z and P is* :

$$O(\frac{Z^2}{P}) < O_{Z,P}(L) < O(\frac{Z^2}{P^2})$$

where N is the total number of mix-scale patches, p_n is the n-th patch size. Based on this boundary, we can conclude the AP time complexity for attention is $O(n)$ for the best case and $O(n^2)$ for the worst case.

6.4.7.2 Training Stability and Patch Size

Figure 6.5 shows the training and validation curves of the models: U-Net, UNETR-32 (patch size = 32), and AP-UNETR-2 (min. patch size = 2) at $4K^2$ resolution. We can see that at the same resolution and model complexity, the UNETR model using AP can converge to a better solution that is more stable than U-Net and UNETR. We hypothesize this is because AP allows the same model to use a smaller patch size under the same model complexity. To test this hypothesis, we further tried the performance of the UNETR model



(a) U-Net (b) UNETR (c) AP-UNETR

(d) UNETR: 64×64 patches (e) UNETR: 16×16 patches (f) UNETR: 4×4 patches

Figure 6.5: Training and validation loss. (Top) different models. (Bottom) UNTER with different patch sizes.

with different patch sizes $[4 \times 4, 16 \times 16, 64 \times 64]$ at $1K^2$ resolution. The results further confirmed our thoughts. In Figure 6.5 (d,e,f), the UNETR model using a smaller patch size 4×4 tends to converge more stably than the bigger patch size 64×64.

6.4.7.3 Overhead of AP: Negligable

In our experiments, the time is taken for the PAIP dataset with resolutions from 512 to 65536 is 4.232 to 286.568 in seconds. This is negligible when compared with training time (Hours).

6.5 Conclusion

In this chapter, we propose a solution that adaptively patches high-resolution images based on image details, drastically reducing the number of patches fed to vision transformer models. This pre-processing approach incurs minimal overhead. We achieve segmentation quality for $64K^2$ images comparable to SoTA models operating on no more than $4K^2$, at much higher efficiency (geomean speedup of $6.9\times$).

Table 6.1: Speedup of AP end-to-end training for PAIP dataset at the same segmentation quality of the baseline. We use the highest dice score of the baseline model (in Table 6.2), and report the AP configuration with similar dice scores.

Resolution	**Model-Patch**	**Sec/Image**	**Sequence Length**	**Quadtree Depth**	**Dice Score** (%)	**Speedup** (Sec/Image)	**Speedup** (Time to Convergence)
512×512	**AP-4**	0.06495	1,024	7	77.88	$7.48\times$	$12.71\times$
1 GPU	UNETR-4	0.4863	16,384	-	77.31		
$1,024 \times 1,024$	**AP-8**	0.14284	1,024	7	75.63	$7.6\times$	$12.92\times$
8 GPUs	UNETR-8	1.0863	16,384	-	75.72		
$4,096 \times 4,096$	**AP-16**	0.32231	2,116	8	75.74	$5.77\times$	$9.8\times$
128 GPUs	UNETR-32	1.8613	16,384	-	75.77		
$8,192 \times 8,192$	**AP-16**	1.1613	2,116	9	76.13	$2.29\times$	$3.89\times$
256 GPUs	UNETR-64	2.6618	16,384	-	75.27		
$16,384 \times 16,384$	**AP-32**	1.7613	1,024	9	75.92	$2.9\times$	$4.93\times$
512 GPUs	UNETR-128	5.1179	16,384	-	75.89		
$32,768 \times 32,768$	**AP-32**	2.1567	2,116	10	75.32	$3.79\times$	$6.44\times$
1024 GPUs	UNETR-256	8.1896	16,384	-	74.96		
$65,536 \times 65,536$	**AP-32**	5.733	4,096	11	75.82	$2.3\times$	$3.91\times$
2048 GPUs	UNETR-512	13.218	16,384	-	75.31		

Table 6.2: Improvement in quality of segmentation for the PAIP against different baselines.

Resolution	Model	Patch	GPUs	Sec/GPU	Depth	Seq_L	Dice.S	Dice.Imp
512×512	**AP (+UNTER)**	2	1	0.06112	8	729	**78.32**	4.11%
		4	1	0.05975	7	676	77.88	
		8	1	0.05812	6	576	75.17	
	UNETR	4	1	0.4863	-	16,384	77.31	
		8	1	0.3746	-	4,096	75.23	
		16	1	0.1477	-	1,024	74.88	
	TransUNet	-	1	0.1783	-	1,024	73.32	
	U-Net	-	1	0.0438	-	-	70.32	
$1,024 \times 1,024$	**AP (+UNTER)**	2	8	0.2314	9	1,024	**78.42**	7.10%
		4	8	0.1786	8	900	77.64	
		8	8	0.1428	7	784	75.63	
		16	8	0.1313	6	576	74.88	
	UNETR	8	32	1.0863	-	16,384	75.72	
		16	16	0.9731	-	4,096	75.12	
		32	8	0.8874	-	1,024	73.22	
	TransUNet	-	8	1.3247	-	4,096	72.38	
	U-Net	-	1	0.0981	-	-	68.92	
$4,096 \times 4,096$	**AP (+UNTER)**	2	128	0.6938	11	4,096	**79.63**	5.09%
		4	128	0.4695	10	2,116	78.17	
		8	64	0.3824	9	1,521	75.74	
		16	32	0.3223	8	1,024	74.96	
	UNETR	32	128	1.8613	-	16,384	75.77	
	TransUNet	-	128	2.1637	-	-	71.32	
	U-Net	-	16	0.3712	-	-	64.11	
$8,192 \times 8,192$	**AP (+UNTER)**	2	256	2.3314	12	10,609	**79.56**	5.70%
		4	256	2.1314	11	8,464	78.31	
		8	128	1.7867	10	4,096	77.61	
		16	64	1.1613	9	2,116	76.13	
	UNETR	64	256	2.6618	-	16,384	75.27	
	TransUNet	-	256	2.3678	-	-	70.89	
	U-Net	-	32	1.2858	-	-	63.21	
$16,384 \times 16,384$	**AP (+UNTER)**	2	512	4.8792	13	16,384	**80.62**	6.23%
		4	256	3.1231	12	8,464	79.31	
		8	256	1.8574	11	4,096	78.84	
		16	128	1.6421	10	2,116	77.43	
	UNETR	128	512	5.1179	-	16,384	75.89	
	TransUNet	-	512	6.1296	-	-	70.46	
	U-Net	-	256	2.7825	-	-	62.97	
$32,768 \times 32,768$	**AP (+UNTER)**	4	1024	7.8916	13	16,384	**78.98**	5.36%
		8	512	6.1792	12	8,464	78.31	
		16	512	4.1685	11	4,096	77.61	
		32	256	2.1567	10	2,116	76.13	
	UNETR	256	1024	8.1896	-	16,384	74.96	
	TransUNet	-	1024	10.001	-	-	69.88	
	U-Net	-	512	4.2714	-	-	61.38	
$65,536 \times 65,536$	**AP (+UNTER)**	8	2048	12.697	13	16,384	**77.77**	3.27%
		16	1024	8.793	12	8,464	76.11	
		32	512	5.733	11	4,096	75.41	
		64	256	3.961	10	2,116	75.13	
	UNETR	512	2048	13.218	-	16,384	75.31	
	TransUNet	-	2048	14.3516	-	-	67.67	
	U-Net	-	1024	5.961	-	-	59.69	

Table 6.3: Segmentation of BTCV[4] for multi-organ segmentation on one GPU. *Time* reported is the end-to-end time to reach the reported dice score.

Model	**Patch Size**	**Time**	**Speedup**	**Dice (%)**
U-Net[120]	N/A	843.90 Sec	0.79×	80.2
TransUNet[138]	N/A	3115.25 Sec	2.91×	83.8
UNETR[3]	4	8386.56 Sec	7.85×	89.1
Swin UNETR*[139]	4	6609.45 Sec	6.19×	91.8
AP-UNETR	2	1067.88 Sec	1×	89.7

*Unlike AP-UNTER, Swin UNTER is pre-trained on five datasets.

Table 6.4: Classification (Top-1 accuracy) of vanilla ViT, HIPT,[5] and AP-ViT on PAIP dataset ($16,384^2$ res.)

Model	**Num. GPUs**	**Patch Size**	**Accuracy**
ViT[75]	128	4,096	68.97
HIPT[5]	128	[16,256,4096]	72.69
AP-ViT-4096	8	4096	67.73
AP-ViT-2	128	2	**79.73**

Chapter 7

Conclusion and Future Work

7.1 Conclusion

This thesis introduces innovative methods to address challenges in meta-learning, few-shot learning, and high-resolution image segmentation through three primary contributions: **Validation Loss Landscape Awareness Meta-optimizer via Transferable Q-Net (TLLE)**, **Validation Loss Aware Sample Generator for Few-shot Training (MGDAO)**, and **Adaptive Patching (AP) for ViT-based High-resolution Segmentation**. Each contribution offers novel approaches to improve model performance, efficiency, and generalization across diverse tasks.

1. Validation Loss Landscape Awareness Meta-optimizer via Transferable Q-Net (TLLE)

TLLE is a meta-optimization framework that uses a Q-Net to map model weights to Q-values rather than traditional hyperparameter tuning. This approach provides two significant advantages. First, it eliminates the need for online controller training during inner optimization, which enhances sample collection speed and enables dynamic online hyperparameter updates. Second, the Q-Net generates useful meta-gradients without requiring

computationally expensive Hessian matrix calculations. The method effectively plans optimal paths in the loss landscape by adjusting learning rates and penalty factors, allowing for more efficient exploration of the validation loss landscape. Importantly, TLLE demonstrates transferability across datasets, as the Q-Net, once fine-tuned, can accurately predict Q-values for new tasks. The thesis also introduces practical improvements to make TLLE computationally efficient, including downsampling worker weights, online updates, and normalized weight representations to enhance generalization.

2. Validation Loss Aware Sample Generator for Few-shot Training (MGDAO)

MGDAO addresses the challenge of few-shot learning by generating validation loss-oriented augmented samples. The method outperforms existing automatic data augmentation techniques by creating "fake test samples" that closely align with the validation loss. Two approaches within MGDAO are highlighted: pixel-wise similarity and feature-level similarity. The pixel-wise method redistributes augmentation rules similar to human-designed strategies, such as noise addition and masking. Meanwhile, the feature-level method alters sample features in ways that are difficult to achieve manually. Experiments across various image and system intrusion classification tasks demonstrate significant performance improvements in few-shot learning. The generalization capability of MGDAO is validated across both image-based and text-based datasets, showcasing its versatility in different domains.

3. Adaptive Patching (AP) for ViT-based High-resolution Segmentation

AP is introduced as a solution for efficiently handling high-resolution image segmentation in Vision Transformers (ViTs). It reduces the number of patches extracted from images, thereby lowering computational and memory costs. This approach allows for smaller patch

sizes (e.g., 2×2 or 4×4), which enhances segmentation quality. Experiments show that AP can achieve the same computational cost as traditional patching while handling up to eight times smaller patch sizes or 64 times longer sequence lengths. AP is tested on high-resolution pathology datasets using the Frontier supercomputer with up to 2,048 MI250X GPUs, demonstrating scalability to resolutions as high as $16K^2$. The method allows for deep multi-resolution scaling without modifying the ViT's attention mechanisms, ensuring compatibility and seamless integration with existing ViTs. AP's low overhead, amortized over epochs, results in negligible computational costs while significantly improving performance in high-resolution tasks.

In summary, this thesis contributes novel methodologies to enhance meta-optimization, data augmentation, and high-resolution segmentation. TLLE leverages transferable Q-Nets for efficient hyperparameter updates and loss landscape exploration. MGDAO provides a validation loss-driven augmentation strategy for few-shot learning, enabling superior generalization across tasks. Lastly, AP optimizes patch extraction in ViTs, achieving significant computational efficiency and scalability in high-resolution image segmentation. These contributions collectively advance the state of the art in deep learning, offering practical solutions to key challenges in modern machine learning.

7.2 Future Work

In future research endeavors, we aim to develop high-quality methodologies and problem-specific algorithms further to tackle diverse optimization problems in both academic domains and real-world scenarios.

Author's Publications

1 Zhang E, Wahib M, Zhong R, et al. Learning from the Past Training Trajectories: Regularization by Validation[J]. Journal of Advanced Computational Intelligence and Intelligent Informatics, 2024, 28(1): 67-78. doi: 10.20965/jaciii.2024.p0067 (IF:1.03)

2 Zhang E, Dong B, Wahib M, et al. Meta generative image and text data augmentation optimization[J]. The Journal of Supercomputing, 2024: 1-19. DOI10.1007/s11227-024-05912-5(IF:2.5)

3 E. Zhang, I. Lyngaas, P. Chen, et al. Adaptive Patching for High-resolution Image Segmentation with Transformers, 2024 The International Conference for High Performance Computing, Networking, Storage, and Analysis, ATLANTA, GA.

4 E. Zhang, R. Zhong, M. Munetomo and M. Wahib, Validation Loss Landscape Exploration with Deep Q-Learning, 2024 International Joint Conference on Neural Networks (IJCNN), 2024, pp. 1-9, doi: 10.1109/IJCNN60899.2024.10651182.

5 E. Zhang, R. Wang, M. Wahib, R. Zhong and M. Munetomo, Training Knowledge Inheritance Through Deep Q-Net, 2023 IEEE International Conference on Systems, Man, and Cybernetics (SMC), Honolulu, Oahu, HI, USA, 2023, pp. 899-904, doi: 10.1109/SMC53992.2023.10393928.

6 E. Zhang, B. Dong, M. Wahib, R. Zhong and M. Munetomo, Meta Generative Data Augmentation Optimization, 2023 Congress in Computer Science, Computer Engineering, Applied Computing (CSCE), Las Vegas, NV, USA, 2023, pp. 2218-2225, doi: 10.1109/CSCE60160.2023.00362.

7 E. Zhang, M. Wahib and M. Munetomo, Learning from the Past: Regularization by Validation, 2022 Joint 12th International Conference on Soft Computing and Intel-

ligent Systems and 23rd International Symposium on Advanced Intelligent Systems (SCIS and ISIS), Ise, Japan, 2022, pp. 1-8, doi:10.1109 /SCISISIS55246.2022.10002143.

Shi Y, Zhang A, Zhang E, et al. Relm: Leveraging language models for enhanced chemical reaction prediction[J]. arXiv preprint arXiv:2310.13590, 2023.

8 Zhong R, Zhang E, Munetomo M. Cooperative coevolutionary surrogate ensemble-assisted differential evolution with efficient dual differential grouping for large-scale expensive optimization problems[J]. Complex and Intelligent Systems, 2024, 10(2): 2129-2149.doi:https://doi.org/10.1007/s40747-023-01262-6

9 Liu Z, Shi Y, Zhang A, et al. Rethinking tokenizer and decoder in masked graph modeling for molecules[J]. Advances in Neural Information Processing Systems, 2024, 36. https://doi.org/10.48550/arXiv.2310.14753

10 Zhong Rui, Zhang Enzhi, Munetomo Masaharu. Cooperative coevolutionary differential evolution with linkage measurement minimization for large-scale optimization problems in noisy environments[J]. Complex and Intelligent Systems, 2023, 9(4): 4439-4456.https://doi.org/10.1007/s40747-022-00957-6

11 Zhong R, Peng F, Zhang E, et al. Vegetation evolution with dynamic maturity strategy and diverse mutation strategy for solving optimization problems[J]. Biomimetics, 2023, 8(6): 454.doi:https://doi.org/10.3390/biomimetics8060454

12 Zhong R, Zhang E, Munetomo M. Evolutionary multi-mode slime mold optimization: a hyper-heuristic algorithm inspired by slime mold foraging behaviors[J]. The Journal of Supercomputing, 2024: 1-32. doi:https://doi.org/10.1007/s11227-024-05909-0

13 Liu Z, Zhang A, Fei H, et al. ProtT3: Protein-to-Text Generation for Text-based Protein Understanding[J]. arXiv preprint arXiv:2405.12564, 2024. doi:10.18653/v1/2024.acl-long.324

14 Chen Y, Tan J, Zhang A, et al. On Softmax Direct Preference Optimization for Recommendation[J]. arXiv preprint arXiv:2406.09215, 2024.

15 Zhong Rui, Zhang Enzhi, Munetomo Masaharu. Accelerating the Genetic Algorithm for Large-scale Traveling Salesman Problems by Cooperative Coevolutionary Pointer Network with Reinforcement Learning[J]. arXiv preprint arXiv:2209.13077, 2022.

16 Liu Z, Shi Y, Zhang A, Zhang Enzhi, et al. ReactXT: Understanding Molecular Reaction-ship via Reaction-Contextualized Molecule-Text Pretraining[J]. arXiv preprint arXiv:2405.14225, 2024.

17 Wang R, Noguchi W, Zhang E, et al. Robust Animal Tracking and Stereotypical Behavior Detection Under Real Environment Using Temporal Averaging Background Subtraction[C] Proceedings of SAI Intelligent Systems Conference. Cham: Springer Nature Switzerland, 2023: 857-875.doi: 10.1007/978-3-031-47724-9-57

18 Zhang E, Zhong R, Du X, et al. Vision Transformer-based Meta Loss Landscape Exploration with Actor-Critic Method[J]. 2024.

19 Zhong R, Cao Y, Zhang E, et al. Introducing Competitive Mechanism to Differential Evolution for Numerical Optimization[J]. arXiv preprint arXiv:2406.05436, 2024.

20 Zhong R, Tu B, Zhang E, Munetomo Masaharu et al. Cooperative coevolutionary differential evolution with adjacent intensity matrix with linkage identification for large-scale optimization problems in noisy environments[J]. Evolutionary Intelligence, 2024: 1-21. https://doi.org/10.1007/s12065-024-00941-8

21 Zhong R, Zhang E, Munetomo M. Evolutionary Multi-mode Slime mould Optimization: A hyper-heuristic algorithm inspired by slime mould foraging behaviors[C] 2023 Congress in Computer Science, Computer Engineering, and Applied Computing (CSCE). IEEE, 2023: 2153-2160. doi:https://doi.org/10.1007/s11227-024-05909-0

22 Zhong R, Tu B, Zhang E, Yu Jun, Munetomo Masaharu et al. Adjacent Intensity Matrix with Linkage Identification for Large-Scale Optimization in Noisy Environments[C] 2023 IEEE Congress on Evolutionary Computation (CEC). IEEE, 2023: 01-10.doi:10.1109/CEC53210.2023.10253969

23 Zhong R, Zhang E, Munetomo M. A Hierarchical Cooperative Coevolutionary Approach to Solve Very Large-Scale Traveling Salesman Problem[C] International Conference on Optimization and Learning. Cham: Springer Nature Switzerland, 2023: 74-84.doi:https://doi.org/10.1007/978-3-031-34020-8-6

24 Zhong R, Zhang E, Munetomo M. Accelerating the Evolutionary Algorithms by Gaussian Process Regression with epsilon-greedy acquisition function[J]. arXiv preprint arXiv:2210.06814, 2022.

25 Ma Y, Zhang E, Tsujino K, et al. An Object Matrix Input Format for a Deep AI in Angry Birds and the Like[C] 2018 IEEE 7th Global Conference on Consumer Electronics (GCCE). IEEE, 2018: 596-598.doi: 10.1109/GCCE.2018.8574799.

Bibliography

[1] Devlin, J.; Chang, M.-W.; Lee, K.; Toutanova, K. *arXiv preprint arXiv:1810.04805* **2018**,

[2] Kim, Y. J. et al. *Medical Image Analysis* **2021**, *67*, 101854.

[3] Hatamizadeh, A.; Tang, Y.; Nath, V.; Yang, D.; Myronenko, A.; Landman, B.; Roth, H. R.; Xu, D. Unetr: Transformers for 3d medical image segmentation. Proceedings of the IEEE/CVF winter conference on applications of computer vision. 2022; pp 574–584.

[4] Landman, B.; Xu, Z.; Igelsias, J.; Styner, M.; Langerak, T.; Klein, A. Miccai multi-atlas labeling beyond the cranial vault–workshop and challenge. Proc. MICCAI Multi-Atlas Labeling Beyond Cranial Vault—Workshop Challenge. 2015.

[5] Chen, R. J.; Chen, C.; Li, Y.; Chen, T. Y.; Trister, A. D.; Krishnan, R. G.; Mahmood, F. Scaling Vision Transformers to Gigapixel Images via Hierarchical Self-Supervised Learning. 2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR). New York, NY, USA, 2022; pp 16123–16134.

[6] Shwartz-Ziv, R.; Tishby, N. *arXiv preprint arXiv:1703.00810* **2017**,

[7] Vanschoren, J. *arXiv preprint arXiv:1810.03548* **2018**,

[8] Franceschi, L.; Frasconi, P.; Salzo, S.; Grazzi, R.; Pontil, M. Bilevel program-

ming for hyperparameter optimization and meta-learning. International Conference on Machine Learning. 2018; pp 1568–1577.

[9] Jomaa, H. S.; Grabocka, J.; Schmidt-Thieme, L. *arXiv preprint arXiv:1906.11527* **2019**,

[10] Lorraine, J.; Vicol, P.; Duvenaud, D. Optimizing millions of hyperparameters by implicit differentiation. International Conference on Artificial Intelligence and Statistics. 2020; pp 1540–1552.

[11] Finn, C.; Abbeel, P.; Levine, S. Model-agnostic meta-learning for fast adaptation of deep networks. International conference on machine learning. 2017; pp 1126–1135.

[12] Hanson, S.; Pratt, L. *Advances in neural information processing systems* **1988**, *1*.

[13] Morgan, N.; Bourlard, H. *Advances in neural information processing systems* **1989**, *2*.

[14] Srivastava, N.; Hinton, G.; Krizhevsky, A.; Sutskever, I.; Salakhutdinov, R. *The journal of machine learning research* **2014**, *15*, 1929–1958.

[15] Goodfellow, I. J.; Shlens, J.; Szegedy, C. *arXiv preprint arXiv:1412.6572* **2014**,

[16] Madry, A.; Makelov, A.; Schmidt, L.; Tsipras, D.; Vladu, A. *arXiv preprint arXiv:1706.06083* **2017**,

[17] Shorten, C.; Khoshgoftaar, T. M. *Journal of big data* **2019**, *6*, 1–48.

[18] Raghunathan, A.; Xie, S. M.; Yang, F.; Duchi, J. C.; Liang, P. *arXiv preprint arXiv:1906.06032* **2019**,

[19] Zhang, C.; Bengio, S.; Hardt, M.; Recht, B.; Vinyals, O. *Communications of the ACM* **2021**, *64*, 107–115.

[20] Ishida, T.; Yamane, I.; Sakai, T.; Niu, G.; Sugiyama, M. *arXiv preprint arXiv:2002.08709* **2020**,

[21] Cubuk, E. D.; Zoph, B.; Mane, D.; Vasudevan, V.; Le, Q. V. *arXiv preprint arXiv:1805.09501* **2018**,

[22] Ho, D.; Liang, E.; Chen, X.; Stoica, I.; Abbeel, P. Population based augmentation: Efficient learning of augmentation policy schedules. International Conference on Machine Learning. 2019; pp 2731–2741.

[23] Lim, S.; Kim, I.; Kim, T.; Kim, C.; Kim, S. *Advances in Neural Information Processing Systems* **2019**, *32*.

[24] Hataya, R.; Zdenek, J.; Yoshizoe, K.; Nakayama, H. Faster autoaugment: Learning augmentation strategies using backpropagation. Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XXV 16. 2020; pp 1–16.

[25] Hataya, R.; Zdenek, J.; Yoshizoe, K.; Nakayama, H. Meta approach to data augmentation optimization. Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision. 2022; pp 2574–2583.

[26] Antoniou, A.; Storkey, A.; Edwards, H. *arXiv preprint arXiv:1711.04340* **2017**,

[27] Baluja, S.; Fischer, I. *arXiv preprint arXiv:1703.09387* **2017**,

[28] Goodfellow, I.; Bengio, Y.; Courville, A. *Deep learning*; MIT press, 2016.

[29] Krizhevsky, A.; Sutskever, I.; Hinton, G. E. *Advances in neural information processing systems* **2012**, *25*.

[30] Li, H.; Xu, Z.; Taylor, G.; Studer, C.; Goldstein, T. *Advances in neural information processing systems* **2018**, *31*.

[31] Keskar, N. S.; Mudigere, D.; Nocedal, J.; Smelyanskiy, M.; Tang, P. T. P. *arXiv preprint arXiv:1609.04836* **2016**,

[32] Garipov, T.; Izmailov, P.; Podoprikhin, D.; Vetrov, D. P.; Wilson, A. G. *Advances in neural information processing systems* **2018**, *31*.

[33] Izmailov, P.; Podoprikhin, D.; Garipov, T.; Vetrov, D.; Wilson, A. G. *arXiv preprint arXiv:1803.05407* **2018**,

[34] Goyal, P.; Dollár, P.; Girshick, R.; Noordhuis, P.; Wesolowski, L.; Kyrola, A.; Tulloch, A.; Jia, Y.; He, K. *arXiv preprint arXiv:1706.02677* **2017**,

[35] Falkner, S.; Klein, A.; Hutter, F. BOHB: Robust and efficient hyperparameter optimization at scale. International Conference on Machine Learning. 2018; pp 1437–1446.

[36] Zoph, B.; Le, Q. V. *arXiv preprint arXiv:1611.01578* **2016**,

[37] Li, Y.; Dong, M.; Wang, Y.; Xu, C. **2020**, 5853–5862.

[38] Watkins, C. J.; Dayan, P. *Machine learning* **1992**, *8*, 279–292.

[39] Mnih, V.; Kavukcuoglu, K.; Silver, D.; Rusu, A. A.; Veness, J.; Bellemare, M. G.; Graves, A.; Riedmiller, M.; Fidjeland, A. K.; Ostrovski, G.; others *nature* **2015**, *518*, 529–533.

[40] Hinton, G.; Vinyals, O.; Dean, J.; others *arXiv preprint arXiv:1503.02531* **2015**, *2*.

[41] Deng, L. *IEEE Signal Processing Magazine* **2012**, *29*, 141–142.

[42] Krizhevsky, A.; Nair, V.; Hinton, G.

[43] Krizhevsky, A.; Nair, V.; Hinton, G.

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

[45] Glorot, X.; Bengio, Y. Understanding the difficulty of training deep feedforward neural networks. Proceedings of the thirteenth international conference on artificial intelligence and statistics. 2010; pp 249–256.

[46] Azuma, K. *Tohoku Mathematical Journal, Second Series* **1967**, *19*, 357–367.

[47] Loshchilov, I.; Hutter, F. *arXiv preprint arXiv:1608.03983* **2016**,

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

[49] Kingma, D. P.; Ba, J. *arXiv preprint arXiv:1412.6980* **2014**,

[50] Sutton, R. S.; Barto, A. G. *Reinforcement learning: An introduction*; MIT press, 2018.

[51] Li, K.; Malik, J. *arXiv preprint arXiv:1606.01885* **2016**,

[52] Foret, P.; Kleiner, A.; Mobahi, H.; Neyshabur, B. *arXiv preprint arXiv:2010.01412* **2020**,

[53] Li, Z.; Zhou, F.; Chen, F.; Li, H. *arXiv preprint arXiv:1707.09835* **2017**,

[54] Antoniou, A.; Edwards, H.; Storkey, A. *arXiv preprint arXiv:1810.09502* **2018**,

[55] You, Y.; Gitman, I.; Ginsburg, B. *arXiv preprint arXiv:1708.03888* **2017**, *6*, 6.

[56] Pham, H.; Guan, M.; Zoph, B.; Le, Q.; Dean, J. Efficient neural architecture search via parameters sharing. International conference on machine learning. 2018; pp 4095–4104.

[57] Mnih, V.; Kavukcuoglu, K.; Silver, D.; Graves, A.; Antonoglou, I.; Wierstra, D.; Riedmiller, M. *arXiv preprint arXiv:1312.5602* **2013**,

[58] Smith, L. N. Cyclical learning rates for training neural networks. 2017 IEEE winter conference on applications of computer vision (WACV). 2017; pp 464–472.

[59] Snoek, J.; Larochelle, H.; Adams, R. P. *Advances in neural information processing systems* **2012**, *25*.

[60] Franceschi, L.; Donini, M.; Frasconi, P.; Pontil, M. Forward and reverse gradient-based hyperparameter optimization. International Conference on Machine Learning. 2017; pp 1165–1173.

[61] MacKay, M.; Vicol, P.; Lorraine, J.; Duvenaud, D.; Grosse, R. *arXiv preprint arXiv:1903.03088* **2019**,

[62] Brown, T. et al. Language Models are Few-Shot Learners. Advances in Neural Information Processing Systems. 2020; pp 1877–1901.

[63] Kaplan, J.; McCandlish, S.; Henighan, T.; Brown, T. B.; Chess, B.; Child, R.; Gray, S.; Radford, A.; Wu, J.; Amodei, D. *arXiv preprint arXiv:2001.08361* **2020**,

[64] Ratner, A. J.; Ehrenberg, H.; Hussain, Z.; Dunnmon, J.; Ré, C. *Advances in neural information processing systems* **2017**, *30*.

[65] Ravuri, S.; Vinyals, O. **2019**,

[66] Goodfellow, I. J.; Pouget-Abadie, J.; Mirza, M.; Xu, B.; Warde-Farley, D.; Ozair, S.; Courville, A.; Bengio, Y. *Advances in neural information processing systems* **2014**, *27*, 2672–2680.

[67] Li, Y.; Hu, G.; Wang, Y.; Hospedales, T.; Robertson, N. M.; Yang, Y. *arXiv preprint arXiv:2003.03780* **2020**,

[68] Liu, A.; Huang, Z.; Huang, Z.; Wang, N. Direct differentiable augmentation search. Proceedings of the IEEE/CVF International Conference on Computer Vision. 2021; pp 12219–12228.

[69] Liu, H.; Simonyan, K.; Yang, Y. *arXiv preprint arXiv:1806.09055* **2018**,

[70] Berthelot, D.; Carlini, N.; Cubuk, E. D.; Kurakin, A.; Sohn, K.; Zhang, H.; Raffel, C. *arXiv preprint arXiv:1911.09785* **2019**,

[71] Cubuk, E. D.; Zoph, B.; Shlens, J.; Le, Q. V. Randaugment: Practical automated data augmentation with a reduced search space. Proceedings of the IEEE/CVF conference on computer vision and pattern recognition workshops. 2020; pp 702–703.

[72] Xie, M.; Hu, J. Evaluating host-based anomaly detection systems: A preliminary analysis of adfa-ld. 2013 6th international congress on image and signal processing (CISP). 2013; pp 1711–1716.

[73] Simonyan, K.; Zisserman, A. *arXiv preprint arXiv:1409.1556* **2014**,

[74] Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A. N.; Kaiser, L.; Polosukhin, I. Attention is All You Need. Advances in Neural Information Processing Systems. 2017.

[75] Dosovitskiy, A.; Beyer, L.; Kolesnikov, A.; Weissenborn, D.; Zhai, X.; Unterthiner, T.; Dehghani, M.; Minderer, M.; Heigold, G.; Gelly, S.; others *arXiv preprint arXiv:2103.17239* **2021**,

[76] Hu, J.; Shen, L.; Sun, G. *Proceedings of the IEEE conference on computer vision and pattern recognition* **2018**,

[77] Wang, X.; Girshick, R.; Gupta, A.; He, K. *Proceedings of the IEEE conference on computer vision and pattern recognition* **2018**,

[78] Carion, N.; Massa, F.; Synnaeve, G.; Usunier, N.; Kirillov, R.; Zagoruyko, S. *European Conference on Computer Vision* **2020**,

[79] Lyngaas, I.; Meena, M. G.; Calabrese, E.; Wahib, M.; Chen, P.; Igarashi, J.; Huo, Y.; Wang, X. *Electronic Imaging* **2024**, *36*, 1–7.

[80] Thisanke, H.; Deshan, C.; Chamith, K.; Seneviratne, S.; Vidanaarachchi, R.; Herath, D. Semantic Segmentation using Vision Transformers: A survey. 2023.

[81] Liu, Z.; Lin, Y.; Cao, Y.; Hu, H.; Wei, Y.; Zhang, Z.; Lin, S.; Guo, B. Swin transformer: Hierarchical vision transformer using shifted windows. Proceedings of the IEEE/CVF international conference on computer vision. 2021; pp 10012–10022.

[82] Li, Y.; Wang, Z.; Li, B.; Zhang, L.; Fu, Y.; He, Y.; Xie, G.; Zeng, Z.; Yu, H.; Chen, D.; others *arXiv preprint arXiv:2108.05620* **2021**,

[83] Chen, C.-F. R.; Fan, Q.; Panda, R. Crossvit: Cross-Attention Multi-Scale Vision Transformer for Image Classification. Proceedings of the IEEE/CVF international conference on computer vision. New York, NY, USA, 2021; pp 357–366.

[84] Chen, J.; Lu, Y.; Yu, Q.; Luo, X.; Adeli, E.; Wang, Y.; Wang, L.; Lu, C. *arXiv preprint arXiv:2102.04306* **2021**,

[85] Shamshad, F.; Khan, S.; Zamir, S. W.; Khan, M. H.; Hayat, M.; Khan, F. S.; Fu, H. *Medical Image Analysis* **2023**, 102802.

[86] Strudel, R.; Garcia, R.; Laptev, I.; Schmid, C. Segmenter: Transformer for semantic segmentation. Proceedings of the IEEE/CVF international conference on computer vision. 2021; pp 7262–7272.

[87] Parmar, N.; Vaswani, A.; Uszkoreit, J.; Kaiser, L.; Shazeer, N.; Ku, A.; Tran, D. Image Transformer. International Conference on Learning Representations. 2018.

[88] Jacobs, S. A.; Tanaka, M.; Zhang, C.; Zhang, M.; Song, S. L.; Rajbhandari, S.; He, Y. DeepSpeed Ulysses: System Optimizations for Enabling Training of Extreme Long Sequence Transformer Models. 2023.

[89] Li, D.; Shao, R.; Xie, A.; Xing, E. P.; Gonzalez, J. E.; Stoica, I.; Ma, X.; Zhang, H. LightSeq: Sequence Level Parallelism for Distributed Training of Long Context Transformers. 2023.

[90] Liu, H.; Zaharia, M.; Abbeel, P. Ring Attention with Blockwise Transformers for Near-Infinite Context. 2023.

[91] Wang, X.; Lyngaas, I.; Tsaris, A.; Chen, P.; Dash, S.; Shekar, M. C.; Luo, T.; Yoon, H.-J.; Wahib, M.; Gouley, J. Ultra-Long Sequence Distributed Transformer. 2023.

[92] Dao, T.; Fu, D. Y.; Ermon, S.; Rudra, A.; Ré, C. FlashAttention: Fast and Memory-Efficient Exact Attention with IO-Awareness. NeurIPS: Proceedings of the 35th Neural Information Processing Systems Conference. New York, NY, USA, 2022.

[93] Dao, T. FlashAttention-2: Faster Attention with Better Parallelism and Work Partitioning. 2023.

[94] Dao, T.; Chen, B.; Sohoni, N. S.; Desai, A.; Poli, M.; Grogan, J.; Liu, A.; Rao, A.; Rudra, A.; Re, C. Monarch: Expressive Structured Matrices for Efficient and Accurate Training. Proceedings of the 39th International Conference on Machine Learning. 2022; pp 4690–4721.

[95] Bo, D.; Shi, C.; Wang, L.; Liao, R. Specformer: Spectral Graph Neural Networks Meet Transformers. The Eleventh International Conference on Learning Representations. 2023.

[96] Kreuzer, D.; Beaini, D.; Hamilton, W.; Létourneau, V.; Tossou, P. Rethinking Graph Transformers with Spectral Attention. Advances in Neural Information Processing Systems. 2021; pp 21618–21629.

[97] Choromanski, K.; Likhosherstov, V.; Dohan, D.; Song, X.; Gane, A.; Sarlos, T.; Hawkins, P.; Davis, J.; Mohiuddin, A.; Kaiser, L.; Belanger, D.; Colwell, L.; Weller, A. Rethinking Attention with Performers. The International Conference on Learning Representations (ICLR). New York, NY, USA, 2021.

[98] Katharopoulos, A.; Vyas, A.; Pappas, N.; Fleuret, F. Transformers Are RNNs: Fast Autoregressive Transformers with Linear Attention. Proceedings of the 37th International Conference on Machine Learning. New York, NY, USA, 2020.

[99] Child, R.; Gray, S.; Radford, A.; Sutskever, I. Generating Long Sequences with Sparse Transformers. 2019; `https://arxiv.org/abs/1904.10509`.

[100] Kitaev, N.; Kaiser, L.; Levskaya, A. Reformer: The Efficient Transformer. The International Conference on Learning Representations (ICLR). New York, NY, USA, 2020.

[101] Roy, A.; Saffar, M.; Vaswani, A.; Grangier, D. *Transactions of the Association for Computational Linguistics* **2021**, *9*, 53–68.

[102] Beltagy, I.; Peters, M. E.; Cohan, A. *arXiv preprint arXiv:2004.05150* **2020**,

[103] Zaheer, M.; Guruganesh, G.; Dubey, K. A.; Ainslie, J.; Alberti, C.; Ontanon, S.; Pham, P.; Ravula, A.; Wang, Q.; Yang, L.; others *Advances in Neural Information Processing Systems* **2020**, *33*, 17283–17297.

[104] Ying, C.; Ke, G.; He, D.; Liu, T.-Y. LazyFormer: Self Attention with Lazy Update. 2021.

[105] Rabe, M. N.; Staats, C. Self-attention Does Not Need $O(n^2)$ Memory. 2022.

[106] Chen, B.; Dao, T.; Winsor, E.; Song, Z.; Rudra, A.; Ré, C. *Advances in Neural Information Processing Systems* **2021**, *34*, 17413–17426.

[107] Shi, H.; Gao, J.; Ren, X.; Xu, H.; Liang, X.; Li, Z.; Kwok, J. T. SparseBERT: Rethinking the Importance Analysis in Self-attention. Proceedings of the 38th International Conference on Machine Learning. New York, NY, USA, 2021; pp 9547–9557.

[108] Si, Y.; Roberts, K. Three-level Hierarchical Transformer Networks for Long-sequence and Multiple Clinical Documents Classification. 2021; `https://arxiv.org/abs/2104.08444`.

[109] Yu, L.; Simig, D.; Flaherty, C.; Aghajanyan, A.; Zettlemoyer, L.; Lewis, M. MEGABYTE: Predicting Million-byte Sequences with Multiscale Transformers. 2023.

[110] Berger, M. J.; Oliger, J. E. *Adaptive mesh refinement for hyperbolic partial differential equations*; 1983.

[111] TU, T.; O'HALLARON, D. R.; GHATTAS, O. Scalable Parallel Octree Meshing for TeraScale Applications. Proceedings of the 2005 ACM/IEEE Conference on Supercomputing. USA, 2005; p 4.

[112] Wahib, M.; Maruyama, N.; Aoki, T. Daino: a high-level framework for parallel and efficient AMR on GPUs. SC. 2016; pp 621–632.

[113] Tropf, H.; Herzog, H. *Angew. Inform.* **1981**, *23*, 71–77.

[114] Bergen, G. v. d. *Journal of graphics tools* **1997**, *2*, 1–13.

[115] Klosowski, J. T.; Held, M.; Mitchell, J. S.; Sowizral, H.; Zikan, K. *IEEE transactions on Visualization and Computer Graphics* **1998**, *4*, 21–36.

[116] Redding, J.; Amin, J.; Boskovic, J.; Kang, Y.; Hedrick, K.; Howlett, J.; Poll, S. A real-time obstacle detection and reactive path planning system for autonomous small-scale helicopters. AIAA Guidance, Navigation and Control Conference and Exhibit. 2007; p 6413.

[117] Ericson, C. *Real-time collision detection*; Crc Press, 2004.

[118] Tang, S.; Zhang, J.; Zhu, S.; Tan, P. *arXiv preprint arXiv:2201.02767* **2022**,

[119] Ibing, M.; Kobsik, G.; Kobbelt, L. Octree transformer: Autoregressive 3d shape generation on hierarchically structured sequences. Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. 2023; pp 2697–2706.

[120] Ronneberger, O.; Fischer, P.; Brox, T. *International Conference on Medical Image Computing and Computer-Assisted Intervention* **2015**,

[121] Child, R.; Gray, S.; Radford, A.; Sutskever, I. *arXiv preprint arXiv:1904.10509* **2019**,

[122] Kitaev, N.; Kaiser, Ł.; Levskaya, A. *arXiv preprint arXiv:2001.04451* **2020**,

[123] Ainslie, J.; Ontanon, S.; Alberti, C.; Cvicek, V.; Fisher, Z.; Pham, P.; Ravula, A.; Sanghai, S.; Wang, Q.; Yang, L. *arXiv preprint arXiv:2004.08483* **2020**,

[124] Zaheer, M.; Guruganesh, G.; Dubey, K. A.; Ainslie, J.; Alberti, C.; Ontanon, S.; Pham, P.; Ravula, A.; Wang, Q.; Yang, L.; others *Advances in neural information processing systems* **2020**, *33*, 17283–17297.

[125] Wang, S.; Li, B. Z.; Khabsa, M.; Fang, H.; Ma, H. *arXiv preprint arXiv:2006.04768* **2020**,

[126] Choromanski, K.; Likhosherstov, V.; Dohan, D.; Song, X.; Gane, A.; Sarlos, T.; Hawkins, P.; Davis, J.; Mohiuddin, A.; Kaiser, L.; others *arXiv preprint arXiv:2009.14794* **2020**,

[127] Chen, L.-C.; Zhu, Y.; Papandreou, G.; Schroff, F.; Adam, H. Encoder-decoder with atrous separable convolution for semantic image segmentation. Proceedings of the European conference on computer vision (ECCV). 2018; pp 801–818.

[128] Long, J.; Shelhamer, E.; Darrell, T. Fully convolutional networks for semantic segmentation. Proceedings of the IEEE conference on computer vision and pattern recognition. 2015; pp 3431–3440.

[129] Badrinarayanan, V.; Kendall, A.; Cipolla, R. *IEEE transactions on pattern analysis and machine intelligence* **2017**, *39*, 2481–2495.

[130] Gu, J.; Kwon, H.; Wang, D.; Ye, W.; Li, M.; Chen, Y.-H.; Lai, L.; Chandra, V.; Pan, D. Z. Multi-scale high-resolution vision transformer for semantic segmentation. Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. 2022; pp 12094–12103.

[131] Yuan, Y.; Fu, R.; Huang, L.; Lin, W.; Zhang, C.; Chen, X.; Wang, J. *arXiv preprint arXiv:2110.09408* **2021**,

[132] Wang, J.; Sun, K.; Cheng, T.; Jiang, B.; Deng, C.; Zhao, Y.; Liu, D.; Mu, Y.; Tan, M.; Wang, X.; others *IEEE transactions on pattern analysis and machine intelligence* **2020**, *43*, 3349–3364.

[133] Zhang, P.; Dai, X.; Yang, J.; Xiao, B.; Yuan, L.; Zhang, L.; Gao, J. Multi-scale vision longformer: A new vision transformer for high-resolution image encoding. Proceedings of the IEEE/CVF international conference on computer vision. 2021; pp 2998–3008.

[134] Canny, J. *IEEE Transactions on Pattern Analysis and Machine Intelligence* **1986**, *PAMI-8*, 679–698.

[135] Samet, H. *ACM Computing Surveys (CSUR)* **1984**, *16*, 187–260.

[136] Finkel, H.; Bentley, J. *Acta Informatica* **1974**, *4*, 1–9.

[137] The Frontier supercomputer. `https://www.olcf.ornl.gov/frontier/`.

[138] Chen, J.; Lu, Y.; Yu, Q.; Luo, X.; Adeli, E.; Wang, Y.; Lu, L.; Yuille, A. L.; Zhou, Y. *CoRR* **2021**, *abs/2102.04306*.

[139] Tang, Y.; Yang, D.; Li, W.; Roth, H. R.; Landman, B. A.; Xu, D.; Nath, V.; Hatamizadeh, A. Self-Supervised Pre-Training of Swin Transformers for 3D Medical Image Analysis. CVPR. 2022; pp 20698–20708.

[140] Loshchilov, I.; Hutter, F. *arXiv preprint arXiv:1711.05101* **2017**,

[141] Zhou, Y.; Xie, L.; Shen, W.; Wang, Y.; Fishman, E. K.; Yuille, A. L. A fixed-point model for pancreas segmentation in abdominal CT scans. International conference on medical image computing and computer-assisted intervention. 2017; pp 693–701.