EARLY LAYER OPTIMIZATION

Zahra Karimpour

A thesis
in
The Department
of
Computer Science and Software Engineering

Presented in Partial Fulfillment of the Requirements
For the Degree of Master of Science (Computer Science) at
Concordia University
Montréal, Québec, Canada

May 2025

Concordia University
School of Graduate Studies

This is to certify that the thesis prepared

By: Zahra Karimpour
Entitled: **Early Layer Optimization**

and submitted in partial fulfillment of the requirements for the degree of

Master of Science (Computer Science)

complies with the regulations of this University and meets the accepted standards with respect to originality and quality.

Signed by the final examining committee:

_____ Chair
Dr. Dhrubajyoti Goswami

_____ Examiner
Dr. Adam Krzyzak

_____ Examiner
Dr. Dhrubajyoti Goswami

_____ Supervisor
Dr. Sudhir P. Mudur

Approved by _____
Name, Graduate Program Director
Department of Computer Science and Software Engineering

_____ 20 _____ _____
Name, Dean
Faculty of Engineering and Computer Science

Abstract

Early Layer Optimization

Zahra Karimpour

In deep learning, early layers play a fundamental role in building general and transferable representations. In this thesis, we demonstrate how improving early layer features can consistently enhance generalization across diverse training settings.

First, we propose a novel iterative training method called Simulated Annealing in Early Layers (SEAL), which applies intermittent gradient ascent followed by descent to the early layers during training. This enables the early layers to escape local minima and refine their representations over time. Doing so reduces overfitting leading to state-of-the-art in in-distribution and transfer generalization in iterative training regime.

Second, we observed poor transfer generalization in greedy learning which we attribute to the lack of generic information especially in the early layers of the network. To address this, we utilize CS-KD regularization to encourage information gain in the early layers. Our results show that this adjustment mitigates the transfer performance drop typically observed in greedy training, while maintaining in-distribution accuracy.

Finally, we extend our investigation to federated learning, where early layer divergence due to gradient accumulation across clients can lead to poor representation learning, even under IID data distributions. We demonstrate that greedy training, by avoiding end-to-end backpropagation, mitigates divergence in the early layers and improves overall performance, particularly in challenging scenarios with deeper models or many clients.

Overall, this thesis highlights the importance of early layer learning in building models that generalize well, and introduces practical strategies for improving it across iterative, greedy, and federated learning paradigms.

Contributions

This thesis is comprised of three papers, each emphasizing importance of proper optimization of early layer learning in deep neural networks. The individual contributions for each paper are as follows:

Simulated Annealing in Early Layers (SEAL) This project was accepted to CVPR 2023 and appears in the proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition [1] (refer to section 3).

This work was a collaborative effort. The initial idea and codebase were developed by Amir Sarfi. Zahra Karimpour contributed significantly to the implementation and conceptual development of the method. Muawiz Chaudhary was responsible for conducting the few-shot learning experiments. Nasir Khalid extended the experiments to synthetic spiral data and contributed to the analysis of internal representation learning.

Encouraging Information Gain in Early Layers for Greedy Training. This work will be submitted to the NeurIPS 2025 Workshop (refer to section 4).

This project was led by Zahra Karimpour, who proposed the main idea, implemented the method, and conducted the primary experiments. Amir Sarfi collaborated in shaping the core idea and contributed to the codebase. Bobae Jeon assisted with the implementation and helped with the experimental setup.

Greedy Federated Learning This work will be submitted to the NeurIPS 2025 Workshop (refer to section 5).

This project was also led by Zahra Karimpour, who proposed the idea, implemented the method, and carried out the main experiments. Amir Sarfi provided guidance on the conceptual direction of the work and contributed to the coding. Bobae supported the implementation and experimental setup.

Acknowledgments

Completing this Master's thesis marks the end of an incredibly challenging yet deeply rewarding chapter in my academic journey, and I am sincerely grateful for the support and guidance I have received along the way.

I am very thankful to my supervisor, Dr. Sudhir Mudur, whose constant support, expertise, and encouragement have been the cornerstone of my research and academic growth. I would also like to thank Dr. Eugene Belilovsky for his insightful perspectives, guidance, and support with my research.

I would like to express my heartfelt thanks to my friends, who supported me during my Master's years by sharing ideas, thoughts, and the many joys and challenges that came our way. I am especially grateful to my labmates, whose encouragement and support kept me motivated every day.

Finally, I want to extend my deepest gratitude to my family. Their unwavering love and support made studying abroad possible, and even from afar, they have been with me every step of the way.

Contents

List of Figures

List of Tables

List of Abbreviations

CE Cross-Entropy Loss

CNN Convolutional Neural Networks

CS-KD Contrastive Self-Knowledge Distillation Loss

FedAvg Federated Averaging

FL Federated Learning

FSL Few-Shot Learning

IID independent and identically distributed

KNN K Nearest Neighbor

LLF Later Layer Forgetting

ResNet Residual Network

SA Simulated Annealing

SEAL Simulated Annealing in Early Layers

Chapter 1

Introduction

1.1 Overview

Deep learning models have achieved remarkable progress, yet the importance of early layers has not been thoroughly examined. Inefficient early layer representations can limit a network's ability to generalize effectively. This thesis addresses this gap by investigating targeted improvements in the network's initial layers. The work is organized into three studies that demonstrate how enhancing early layers can significantly boost overall performance:

- **Simulated Annealing in Early Layers (SEAL)**: The first study introduces a novel application of simulated annealing to deep learning. By encouraging employing a simplified version of simulated annealing in early layers, SEAL mitigates overfitting especially in prolonged training on relatively small datasets. Empirical evaluations on ResNet-18 and ResNet-50 demonstrate enhanced performance in both transfer learning and few-shot tasks, outperforming the previously established baselines.

- **Encouraging Information Gain in Early Layers for Greedy Training**: The second study examines the limitations of traditional greedy training methods in transfer settings. By selectively using CS-KD for early layers to encourage generic information gain while applying Cross-Entropy for later layers to promote task-specific features, we observe remarkable improvements in both transfer and in-distribution performance.

- **Greedy Federated Learning**: The third study addresses a challenge in federated learning where the compound effect of backpropagation causes early layers to

diverge more than the later layers. We mitigate this by adopting a greedy training approach that updates layers independently, resulting in significant performance gains, especially in deeper networks, with more clients, and fewer communication rounds.

1.2 Outline

This thesis is structured into seven chapters. Chapters 1 and 2 cover the introduction and background, while Chapters 3, 4, and 5 present the main contributions of the work. Chapter 6 details the experimental results, and Chapter 7 concludes the study.

Simulated Annealing in Early Layers (SEAL) Recently, a number of iterative learning methods have been introduced to improve generalization. These typically rely on training for longer periods of time in exchange for improved generalization. LLF (later-layer-forgetting) is a state-of-the-art method in this category. It strengthens learning in early layers by periodically re-initializing the last few layers of the network. Our principal innovation in this work is to use Simulated annealing in EArly Layers (SEAL) of the network in place of re-initialization of later layers. Essentially, later layers go through the normal gradient descent process, while the early layers go through short stints of gradient ascent followed by gradient descent. Extensive experiments on the popular Tiny-ImageNet dataset benchmark and a series of transfer learning and few-shot learning tasks show that we outperform LLF by a significant margin. We further show that, compared to normal training, LLF features, although improving on the target task, degrade the transfer learning performance across all datasets we explored. In comparison, our method outperforms both LLF and normal training across the same target datasets by a large margin. We also show that the prediction depth of our method is significantly lower than that of LLF and normal training, indicating on average better prediction performance (in chapter 3).

Encouraging Information Gain in Early Layers for Greedy Training Training deep neural networks typically involves end-to-end optimization, which naturally fosters a hierarchical learning process where early layers capture simple primitives while higher layers learn complex, class-specific features. Methods that break this hierarchy, such as LayerCNN, where each layer is optimized independently, offer more interpretable models and lower computational costs due to a reduced computational graph. However, while these methods outperform end-to-end training

in in-distribution settings, our empirical results demonstrate that such decoupling comes at the expense of transfer performance. To address this issue, we propose a novel training strategy that encourages information gain in the early layers via class-wise self-knowledge distillation (CS-KD) regularization, while allowing later layers to focus on the primary task using a cross-entropy objective. Extensive experiments on CIFAR-10, CIFAR-100, and Tiny-ImageNet demonstrate that our method improves the transfer performance of greedy training methods significantly, even surpassing normal training in some cases, and that employing distinct loss functions for early and later layers is crucial(in chapter 4).

Greedy Federated Learning Federated learning has emerged as a promising approach for training a global model across clients while preserving user privacy. Despite its success, recent findings reveal that deeper neural networks can lead to inferior performance even when client data is independently and identically distributed (i.i.d.). These observations suggest that the accumulation of divergence during backpropagation is a key contributing factor. In this paper, we investigate the use of greedy layer-wise training at the client level to mitigate this issue. Unlike traditional end-to-end optimization, greedy learning optimizes each layer individually, thereby eliminating the propagation of divergence across layers. We examine the impact of network depth, the number of local training steps per client, and the number of clients. Our experiments demonstrate that greedy training significantly outperforms end-to-end training across a variety of settings, especially in scenarios with deeper networks or more extensive local training, underscoring the potential of greedy layer-wise training in federated learning (in chapter 5).

Chapter 2

Background

2.1 Fundamentals of Deep Learning

The reader is assumed to have a basic understanding of deep learning concepts. In this section, we cover the essential background relevant to the work presented in this thesis. For more comprehensive information, readers are encouraged to refer to Goodfellow et al. [7].

Convolutional Neural Networks (CNNs) CNNs are deep neural networks built by stacking convolutional layers designed to learn features from grid-like data, such as images [8]. They typically include pooling layers to downsample the data and reduce computational complexity, and use activation functions to introduce non-linearity.

VGG Networks The VGG network, introduced by Simonyan and Zisserman [9], is a widely used CNN architecture valued for its simplicity and depth. It is built by stacking sequential 3x3 convolutional layers with ReLU [10] activations, with occasional max-pooling layers for downsampling. However, due to their deep and sequential structure, they suffer from the vanishing gradient problem during end-to-end training.

ResNets and Residual Blocks ResNets, introduced by He et al. [11], address the vanishing gradient problem in deep networks. They do this by using residual blocks that include skip connections, which help gradients flow directly to earlier layers. A typical residual block can be defined as:

$$x_{l+1} = x_l + F(x_l, W_l) \tag{1}$$

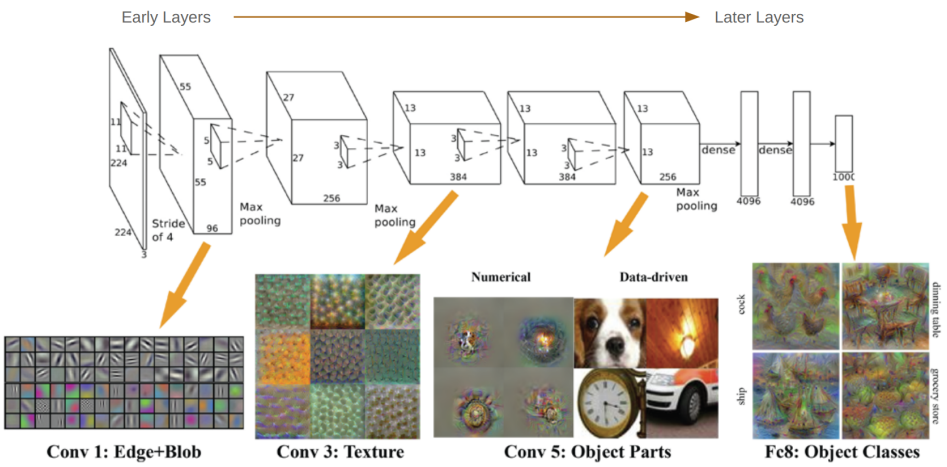

Figure 1: Hierarchical feature learning in a convolutional neural network: Early layers extract generic and low level features while later layers capture class specific representations.[1]

Here, x_l is the input to the block, and $F(x_l, W_l)$ is the transformation applied within the block, typically involving one or more convolutional layers. We utilize ResNet and VGG architectures for our experiments in this thesis.

Cross-Entropy Loss and CS-KD Loss The Cross-Entropy (CE) loss is a widely used objective function for classification tasks. Given a model's posterior distribution $p(y|x)$ and the true label distribution $q(y)$, CE loss is defined as:

$$L_{CE} = -\sum_{y} q(y) \log p(y|x) \tag{2}$$

CE explicitly encourages the model to maximize the likelihood of the correct class while implicitly minimizing incorrect predictions.

Contrastive Self-Knowledge Distillation (CS-KD) loss [12] is a regularization term added to the CE loss. This technique uses the network's embedding outputs as soft targets to prevent the network from becoming overly task-specific by promoting the learning of more general and transferrable features. We utilize CS-KD as a solution to enhance the network's generalization in chapter 4.

Transfer Learning Instead of always starting the training from scratch, transfer learning leverages pre-trained models that have already learned useful features from one task, adapting them to a related task (figure 2). This can be done either by

[1]Image from `https://cs.brown.edu/courses/cs143/2017_Spring/proj6a/`

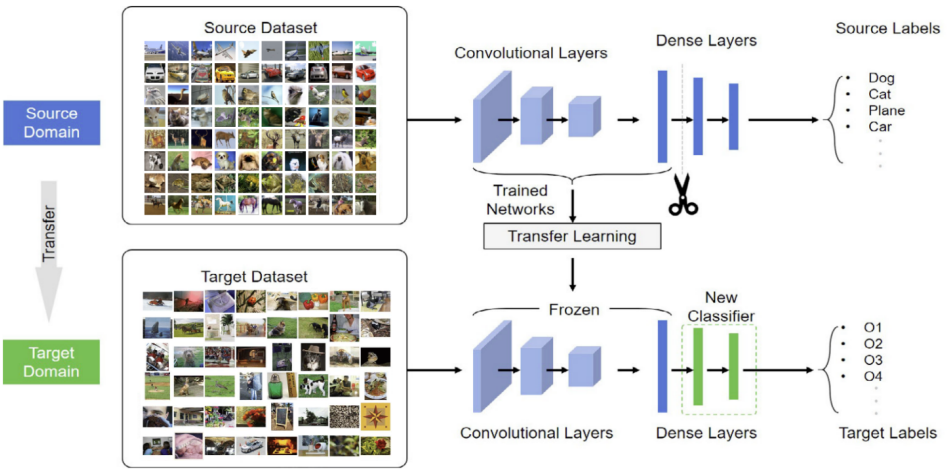

Figure 2: A convolutional network pre-trained on a source dataset (top) and transferred to a target domain (bottom). The shared convolutional backbone is kept frozen, while the task-specific classification head is re-initialized and fine-tuned on the target data[2].

freezing the pre-trained network as a fixed feature extractor or by fine-tuning the model end-to-end on new data [13, 14]. Moreover, transfer learning performance is a strong indicator of how general and robust the learned features are [14].

The success of transfer learning is largely dependent on how generic or task-specific learned representations across different layers of a deep network. As established in prior work [14], early layers in a CNN capture general, low-level features such as edges and textures, which are largely transferable across tasks. In contrast, later layers encode high-level, task-specific features that may not generalize well to different domains. Thus, strength of transfer learning is usually attributed to the generalizability of the early and mid layers of the network. This key insight forms the basis of chapter 4. Figure 1 demonstrates how feature extraction differs across different layers of a convolutional network.

2.2 Simulated Annealing

Simulated Annealing (SA) is an optimization algorithm inspired by the physical annealing process, where a material is heated and then gradually cooled to achieve a stable, low-energy state [15]. In the context of optimization, SA is used to escape

[2]Image from: `https://www.researchgate.net/publication/342400905_Real-Time_Assembly_Operation_Recognition_with_Fog_Computing_and_Transfer_Learning_for_Human-Centered_Intelligent_Manufacturing`

local optima and explore the search space more effectively. Specifically, the heating process (gradient-ascent phase) allows escaping from the local minima even at the cost of transient loss increase, followed by gradual cooling (gradient-descent phase) towards the global optimum. Inspired by this, we propose a novel training strategy and show the effectiveness of SA in deep learning 3.

2.3 Iterative Training Methods

Training deep neural networks for a large number of epochs presents significant challenges, with overfitting being the primary obstacle [7]. Overfitting occurs when the model memorizes training data instead of learning generalized patterns, leading to poor performance on unseen data. Overfitting is especially severe when working with small datasets or prolonged training regimes [4, 16].

To reduce overfitting in long training regimes, iterative training methods have been introduced. These methods typically dividing training into multiple "generations", where at the beginning of each generation, the model is slightly perturbed to encourage further optimization and an escape from the local minima. By preventing the model from becoming trapped in sharp minima or flat regions of the loss landscape, iterative training promotes better exploration and generalization [2–4, 16, 17].

In chapter 3, we propose Simulated Annealing in Early Layers (SEAL) to improve generalization in iterative training methods.

2.4 Greedy Layerwise Training

Greedy layerwise training is an alternative to conventional end-to-end backpropagation. Instead of optimizing all network parameters simultaneously, the training proceeds one layer or block at a time. Each layer is trained independently and then frozen before moving to the next. This approach reduces computational and memory demands by limiting the number of parameters involved in each training step and has been explored for its benefits in interpretability, parallelization, and scalability [18,19].

One of the early motivations for greedy training came from the difficulty of training deep networks with standard backpropagation, particularly due to the vanishing gradient problem [11]. Bengio et al. [18] introduced layer-wise pretraining as an unsupervised method that optimizes each layer independently before a final fine-tuning step. This idea was later extended into supervised methods like LayerCNN

[19], where each block is trained with an auxiliary classifier while previous layers remain fixed, ensuring stable intermediate representations. Greedy training offers clear advantages: it simplifies training dynamics by reducing inter-layer dependencies, lowers computational costs by updating fewer parameters at each step, and supports more modular and parallelizable learning.

In Chapter 4, we demonstrate the limited transfer generalization of Greedy Training and propose a novel solution to address it.

2.5 Federated Learning

Federated Learning (FL) is a machine learning technique that allows multiple clients to collaboratively train a shared model without exposing their local data. Unlike centralized machine learning where all data is processed on a single server, FL keeps data on local devices and only exchanges model updates. This setup provides privacy and adapts well to real-world scenarios, such as mobile applications and healthcare, where privacy is a critical [20–22].

Centralized and Decentralized Architectures In FL, architectures can be categorized into centralized and decentralized designs. In the centralized FL, a central server gathers updates from clients, aggregates them (often via weighted averaging), and redistributes the updated model [20]. Decentralized FL, by contrast, allows clients to directly communicate with each other, reducing single-point failures but introducing challenges in synchronization and efficiency [20–23].

Federated Averaging (FedAvg) The FedAvg algorithm [20] is a fundamental method in FL. In FedAvg, each client trains a model on its local data for several epochs and then sends its updated parameters to a central server. The server aggregates these updates typically by computing a weighted average and redistributes the new global model to the clients. This iterative process reduces communication costs by allowing multiple local updates per round.

Divergence in Federated Learning In Federated Learning, divergence refers to discrepancies among client models during training. Such divergence is problematic because significant differences in client updates can prevent the global model from converging effectively, leading to degraded overall performance [21,24]. A related issue is "divergence accumulation." When using end-to-end backpropagation, differences

among client models can be magnified as gradients are propagated backward through the network. This effect is especially harmful to early layers, which are critical for learning robust, general features. As divergence accumulates, the global model may fail to capture consistent and transferable representations, thereby undermining the benefits of federated training [24].

In Chapter 5, we propose methods to address early layer divergence in FedAvg and enhance FL performance by employing greedy training to mitigate client divergence observed with end-to-end gradient descent.

Chapter 3

Simulated Annealing in Early Layers Leads to Better Generalization

3.1 Proposed Method

In this work, we split the network into fit $\mathbf{H}_{fit}$ and forgetting hypotheses $\mathbf{H}_{forget}$, using a layer threshold, say, L. All weights prior to L are considered in the forgetting hypothesis $\mathbf{H}_{forget}$, and weights in layers $> L$ as the fit hypothesis $\mathbf{H}_{fit}$. This is the opposite of LLF [16], since their fit and forgetting hypotheses are swapped.

To induce forgetting, we perform gradient ascent on the forgetting hypothesis $\mathbf{H}_{forget}$ for k epochs. During the gradient ascent phase of the forgetting hypothesis $\mathbf{H}_{forget}$, we train the fit hypothesis $\mathbf{H}_{fit}$ normally (with gradient descent). This can be categorized under the high-level definition of forgetting that Zhou et al. [16] provide, in that it drops the training accuracy of the network to completely random. This is inspired by the simulated annealing algorithm to enhance the optimization of the network and to introduce a more definitive forgetting mechanism. Our method is different from the prior methods as they either remove [17, 25] or re-initialize [16] [2] the forgetting hypothesis $\mathbf{H}_{forget}$ (at once) before the first epoch of a new generation. We set k to $\frac{1}{4}$ of total epochs E in the generation.

We adjusted the sign of the weight decay for the layers that perform gradient ascent; so as to avoid the weights being encouraged to have a higher norm, and causing the network to diverge. However, we found that even this is not enough to stop the divergence. We noted that using the same learning rate for the ascending

phase was the reason for this. Hence we toned down the ascent learning rate using a factor S:

$$\eta_{\text{a}} = S \times \eta \tag{3}$$

Where η is the learning rate, and η_{a} is the ascent learning rate. We empirically fix $S = 0.01$. We summarize the whole process as follows:

$$\begin{aligned} \boldsymbol{\Theta}_{forget}^{e,t+1} &= \begin{cases} \boldsymbol{\Theta}_{forget}^{e,t} + \eta_{\text{a}} \nabla J(\boldsymbol{\Theta}_{forget}^{e,t}), & \text{if } \ e \ \% \ E < k \\ \boldsymbol{\Theta}_{forget}^{e,t} - \eta \nabla J(\boldsymbol{\Theta}_{forget}^{e,t}), & \text{otherwise} \end{cases}, \\ \boldsymbol{\Theta}_{fit}^{e,t+1} &= \boldsymbol{\Theta}_{fit}^{e,t} - \eta \nabla J(\boldsymbol{\Theta}_{fit}^{e,t}) \end{aligned} \tag{4}$$

Where $J(\boldsymbol{\Theta})$ is the objective function, $\boldsymbol{\Theta}_{forget}^{e,t}$ and $\boldsymbol{\Theta}_{fit}^{e,t}$ refer to parameters in the forgetting and fit hypotheses, respectively, and e, t refers to the iteration t during epoch e. Again, E refers to epochs in each generation and is fixed for all generations. Figure 3 illustrates this for LLF and our proposed method (SEAL).

Complete implementation details and experimental results are presented in Section 6.1

Chapter 4

Encouraging Information Gain in Early Layers for Greedy Training

4.1 Introduction

The typical approach to training convolutional neural networks for classification involves using cross-entropy as the loss function at the end, resulting in a hierarchical learning process. The early layers of the network learn low- and mid-level representations, such as shape primitives, while the later layers, which are closer to the loss function, concentrate on high-level representations that are more relevant to the classification task. This hierarchical learning strategy allows the network to gradually acquire more complex features and ultimately make more accurate predictions [26]. Zhao et al. [27] claimed that what matters in transfer learning is the low- and mid-level representations, and not the high-level, class-specific representations.

Several methods break the hierarchy in neural network learning, either explicitly or implicitly, by using various training methodologies [1,16,19]. Explicit modification of the hierarchy involves applying the objective function on multiple layers of the network instead of just the last layer as in normal training. Belilovsky et al. [19] proposed LayerCNN, where they optimize each layer to classify the input. Compared to normal training, this method applies the objective function to every layer, updates only one layer at a time, and freezes each fully-updated layer before moving on to the next. This results in strong priors being established from the previous, whereas normal training produces outputs from near-random priors in the early stages.

Implicit modification of the hierarchy involves making changes to subparts of the network. For example, Zhou et al. [16] proposed later layer forgetting (LLF), where

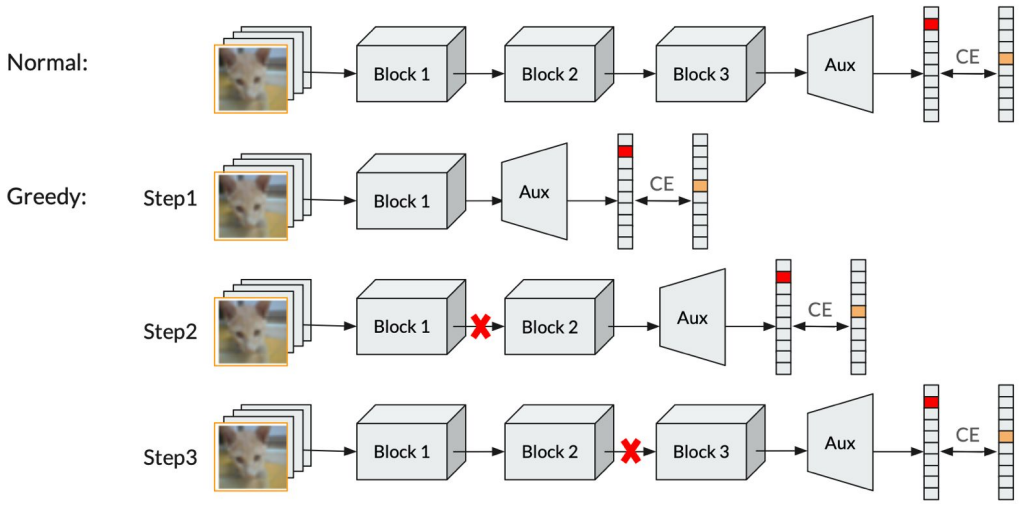

Figure 3: Normal training compared to greedy training. In normal training, the network is optimized end to end using the loss function that is calculated on the final layer. In greedy training, the loss function is directly applied to each layer, and only one layer is updated at a time. The x mark denotes “stop gradient" which prevents the backpropagation from reaching earlier layers.

later layers are periodically reinitialized, and Sarfi et al. [1] periodically perform gradient ascent on early layers.

Sarfi et al. [1] demonstrated that implicit frameworks have substantially different transfer learning behavior than normal training. For instance, LLF has much worse transfer learning than a network that is trained normally for much fewer epochs. They further introduced their method, Simulated Annealing in Early Layers (SEAL), which has better transfer learning than both LLF and normal training.

In this work, we analyze the frameworks that explicitly break the hierarchy of convolutional networks in transfer learning scenarios, in an attempt to find a training schema that has both stronger in-distribution and transfer learning performance. We perform multiple experiments using the popular CIFAR-10, CIFAR-100, and Tiny-ImageNet datasets. We observe that even though LayerCNN leads to comparable (and sometimes stronger) distribution performance compared to normal training, it has a much worse transfer performance.

Tapp [28] suggested that the neural network’s top layers should focus on optimizing for the task at hand, while the lower layers should prioritize maximizing information gain. However, in LayerCNN, all layers are optimized for the classification task. To test whether this is the reason behind the poor transfer performance of LayerCNN, we modify the objective function of different layers of the network to encourage

information gain in the early layers of the network and leave the later layers of the network to focus on the classification task (using cross-entropy). Yun et al. [12] proposed a regularization technique, namely CS-KD, where they aim to either match or distill the predictive distribution of deep neural networks across multiple samples that share the same label. They do so by first sampling an auxiliary batch with the same labels as the batch of the data at hand and then performing Kullback-Leibler (KL) divergence on the predicted distribution between the two batches of data (as the corresponding images have the same labels). To encourage the information gain in early layers of the network, we make use of the CS-KD regularization, and up to which layer should use this regularizer is a hyper-parameter that we investigate. We observe that CS-KD eliminates the gap between normal training and LayerCNN in transfer learning while having a minor effect on the in-distribution performance. We also observe that in the transfer setting, it is best to perform CS-KD in the middle layers of the network rather than all or none of the network.

4.2 Background and Related Work

Iterative training: When training a network for many epochs, its generalization performance deteriorates. In iterative training methods, after the model well-fits the data, they perform some operation onto the network that lowers the training accuracy and makes room for more training. Then, they retrain the network and repeat this process when the data is well fit. Zhou et al. [16] categorize these methods as a "forget-and-relearn" mechanism, and propose their own method Later Layer Forgetting (LLF) where they periodically randomly initialize the later layers of the network. Sarfi et al. [1] proposed simulated annealing in early layers (SEAL) where they periodically perform short periods of gradient ascent on the early layers of the network, imitating simulated annealing. They further show that LLF and SEAL have a significant impact on the transfer generalization of the network compared to normal training. We argue that iterative training methods (such as LLF and SEAL) break the hierarchical learning of the network and the difference in in-distribution and transfer behavior of these methods compared to normal training is a result of that. In this paper, we analyze the in-distribution and transfer performance of training frameworks that explicitly break the hierarchical training of neural networks.

Greedy training: LayerCNN [19] is an example of such method. In this method, they optimize each layer on the target task separately, freeze the optimized layers, and use them to serve the upper layers. Compared to normal training, first,

each layer is optimized separately which results in less computational and memory costs as the computation graph becomes smaller. Second, at the beginning of training in normal training, the input of later layers is completely random as the weights are randomly initialized. On the other than, in LayerCNN, as they optimize all layers before going to the new one, the input of higher layers is already optimized and meaningful. Belilovsky et al. [19] indicated that having normal, end-to-end learning can make the behavior of intermediate layers hard to interpret. Moreover, it is challenging to understand the connection between shallow neural networks and deep neural networks. Lastly, it results in high costs in both computation and memory resources since it involves end-to-end backpropagation. However, we argue that this way of training makes all layers of the network very dependent on the task at hand, and may have negative consequences in transfer learning.

4.3 Proposed Method

Consider a network that consists of k consecutive blocks $[B^1_{\phi^B_1}, B^2_{\phi^B_2}, ..., B^k_{\phi^B_k}]$, where the output of each block is the input of the next, and the parameters of a block B^i are given by ϕ^B_i. Each block can be any subpart of the network consisting of any number of layers. In normal cross-entropy training, a single auxiliary block A_{ϕ^A}, typically consisting of average pooling and a linear layer, is added to the end of the network that makes the prediction based on the learned features by the network, and the whole model is trained end-to-end using the cross-entropy loss, and each parameter update modifies all of the parameters in the network $[\phi^B_1, ..., \phi^B_k, \phi^A]$. Belilovsky et al. [19] argue that such training leads to poor interpretability and high computational and memory costs.

In LayerCNN [19], the authors propose a new training framework to alleviate the flaws of normal training. Considering the same network as above, they define k auxiliary blocks $[A^1_{\phi^A_1}, A^2_{\phi^A_2}, ..., A^k_{\phi^A_k}]$, where A^i corresponds to block B^i and is parameterized by ϕ^A_i. The auxiliary blocks can be thought of as classifiers for the feature representations of each block. At each instance of time, only the parameters of the current block and its corresponding auxiliary block are updated. For instance, At step 1, the network consists of $[B^1_{\phi^B_1}, A^1_{\phi^A_1}]$ where only parameters in $[\phi^B_1, \phi^A_1]$ are updated for E epochs. Then, at any other step i, where the goal is to update the ith block, all of the blocks prior to this target block are frozen, and only the parameters in $[\phi^B_i, \phi^A_i]$ are optimized for the same number of epochs E. This means that when a layer is optimized, it will not get changed during the training, and at each step

of time, much fewer parameters are updated and stored in the computational graph, rendering it faster and more memory efficient. Furthermore, all layers are performing classification on their own, and thus interpretability of intermediate layers is much improved. The comparison between normal training and LayerCNN is illustrated in figure 3.

We argue that local losses (LayerCNN) inject class-specific information into all layers of the network, weakening the low- and mid-level representations. Thus, we hypothesize that their transfer learning performance may be weaker than normal training. To test our hypothesis, we compare the transfer learning performance of LayerCNN to that of Normal training in tables 17 and 16, where we observe that the transfer performance of LayerCNN is far worse than that of normal training.

Tapp [28] argues that it is best to encourage the information gain in the early layers of the network, while only specializing the later layers of the network for the task at hand. To enhance the information gain of the early layers, we apply the CS-KD regularization [12] up to a given layer threshold L (inclusive). Therefore, early layers of the network $[B^1_{\phi^B_1}, ..., B^L_{\phi^B_L}]$ are updated using the cross-entropy loss function with CS-KD regularizer, while the rest of the networks $[B^{L+1}_{\phi^B_{L+1}}, ..., B^k_{\phi^B_k}]$ are optimized using only the cross-entropy loss. Note that as this regularization is added to the normal loss, the loss becomes much larger than that of normal cross-entropy, leading to divergence of the network. Therefore, when this regularizer is used, we multiply the learning rate by 1/10 to take the increase of the loss into account. We observe that just doing so eliminates the gap between the transfer performance of greedy training and normal training while having minimal impact on the in-distribution performance (tables 17 and 16). In the future, we wish to investigate other loss functions that aim at solely improving the information gain for the early layers.

Section 6.2 provides implementation details and experimental results are presented.

Chapter 5

Greedy Federated Learning

5.1 Introduction

Federated learning is a promising paradigm for training a global model across multiple clients while preserving user privacy [20,21]. This approach has achieved great success in a variety of applications [20–22]. However, recent work by Wang et al. [24] showed that federated learning with deeper neural networks can suffer from a performance decline. Their study indicates that differences between client models (divergence) tend to accumulate during backpropagation. More specifically, at each layer the gradient is influenced both by the error signal from the next layer and by variations in the local input data. As these discrepancies build up layer by layer, deeper networks experience a compounded effect, ultimately leading to degraded performance.

Given these observations, we propose adopting greedy layer-wise training in federated learning. Traditional training typically relies on end-to-end optimization, where all layers are updated based on the objective function computed from the final output [11, 29, 30]. In contrast, greedy learning breaks this end-to-end training by optimizing each layer individually [18, 19, 31]. For example, in LayerCNN [19], the objective is computed at every layer, and only one layer is updated at a time. We hypothesize that isolating the training of each layer can reduce early layer divergence by limiting the propagation of error signals between layers, thereby preventing the accumulation of discrepancies.

We evaluate both greedy learning and classical end-to-end training within federated settings under varying conditions such as network depth, the number of local training iterations, and the number of clients. Our experiments reveal that as the network deepens, when clients perform more local iterations prior to communication, and as the number of clients increases, end-to-end training suffers

from amplified early layer divergence, leading to significantly worse performance. In contrast, greedy training proves to be much more robust under these conditions, consistently outperforming the baselines (see Table 18).

5.2 Background and Related Work

Impact of deeper neural networks in federated learning: In recent federated learning studies, the emphasis has been primarily on shallow networks. However, Wang et al. [24] shed light on the adverse effects of deeper networks on federated learning performance. Within the federated learning setting, deeper networks often lead to a decline in performance, even when client data is independently and identically distributed (i.i.d.). Their work illustrates that deep layers exhibit increasing and non-converging divergences. Notably, their findings indicate that backpropagation results in the accumulation of divergences, consequently causing a deterioration in performance in deeper networks due to a longer chain of accumulation. Wang et al. also propose guidelines to enhance federated learning performance in deeper networks, emphasizing the importance of strategic model architectures and enhanced data preprocessing strategies. They successfully demonstrate the relationship between parameter divergence and architecture, offering valuable insights and guidelines to improve the performance of federated learning with deeper neural networks. However, there is still potential for further exploration with greedy learning.

Greedy learning: The conventional approach to training convolutional neural networks (CNNs) in classification tasks involves the application of a single loss function, establishing a hierarchical learning paradigm. Explicitly altering this hierarchy requires implementing the objective function across multiple layers of the network, deviating from the standard practice of exclusive application at the last layer. LayerCNN, as proposed by Belilovsky et al. [19], introduces a unique strategy for optimizing each layer for input classification. Unlike regular training, LayerCNN applies the objective function to every layer, updating one layer at a time and freezing each fully-updated layer before proceeding. The distinctive feature of LayerCNN, where gradient updates come exclusively from the preceding layer, motivates exploring its application in federated learning.

5.3 Proposed Method

Local model training In a federated learning setting, each client iteratively updates its local parameters using distributed data over multiple epochs. The

local models are parameterized as $[W_1, ..., W_n]$, where n is the number of clients. Subsequently, the server aggregates these local model parameters to form a global model W_g. This iterative process spans multiple communication rounds.

For local parameter updates on clients, we compare the performance of greedy learning with normal learning. In the conventional training scenario, the output of the preceding layer is forwarded to the subsequent layer. Once the final prediction is obtained, the global loss is backpropagated through every layer. Now, consider a CNN block B composed of multiple layers. The network consists of a sequence of k consecutive blocks $[B^1_{\phi^B_1}, B^2_{\phi^B_2}, ..., B^k_{\phi^B_k}]$, where ϕ^B_k denotes the parameters of block B^k. At the end of the sequence, an auxiliary block A_{ϕ^A} with average pooling and a fully-connected layer is added. The entire network passes through end-to-end training using cross-entropy loss, optimizing all parameters for improved predictive performance. We argue that such training leads to more challenging convergence due to the accumulation of divergence.

In greedy learning, the implementation of LayerCNN [19] is applied. While utilizing the same network architecture as in normal learning, instead of adding a single auxiliary block solely at the end of the entire network, an auxiliary block is attached to each CNN block. This results in the definition of k auxiliary blocks $[A^1_{\phi^A_1}, A^2_{\phi^A_2}, ..., A^k_{\phi^A_k}]$, where A^i corresponds to block B^i and is parameterized by ϕ^A_i. The addition of an auxiliary block at the end of each block allows us to leverage the classifier output of each block, enabling the updating of parameters one block at a time. The pair of blocks $[B^{i-1}_{\phi^B_{i-1}}, A^{i-1}_{\phi^A_{i-1}}]$, is updated for E epochs before moving on to the ith pair. Importantly, as the training progresses to subsequent pairs of blocks, the weights of previously trained blocks are frozen. This means that the parameters of earlier blocks are no longer subject to updates during the training of subsequent pairs. Compared to the normal end-to-end training, far fewer parameters are updated from a shallower depth at each training step.

Federated averaging We employ FedAvg [32] for aggregation, a widely used baseline. Following independent local training by each client, the updated local model parameters are transmitted to the server for aggregation. The central server accumulates model updates from all participating clients and aggregates them into a global model using a straightforward averaging method. Formally, the updated global model parameter W_g, is calculated as the average of corresponding parameters from all clients: $W_g = \frac{1}{m} \sum_{i=1}^{m} W_i$, where W_i is the model parameter update from the ith client, and m is a fraction of the total number of clients (the fraction is always

set to 0.1). Subsequently, the updated global model W_g is distributed to all clients, initiating the next round of local training.

We propose that the challenge lies in the conventional backpropagation training within federated learning context, that the backpropagation signal given to early layers is weak, resulting in divergence. To test our hypothesis, we adopt a greedy training approach using LayerCNN for each client. This strategy ensures that every network update originates from a shallow backpropagation signal. If our hypothesis holds true, the greedy training is expected to mitigate divergence and yield improved performance, particularly in scenarios prone to divergence, such as those involving a larger number of clients and epochs per client.

Implementation details and experimental results can be found in Section 6.3.

Chapter 6

Experimental Results

6.1 SEAL

6.1.1 Implementation Details

We use Tiny-ImageNet [33] to train models and then evaluate on both the Tiny-ImageNet test set and a wide set of downstream transfer learning tasks, including popular few shot learning benchmarks. Following LLF, we use ResNet50 and train using SGD optimizer with a momentum of 0.9 and weight decay of 5e−4. We train $G = 10$ generations for $E = 160$ epochs using a batch size of 32. As in LLF, we use cross entropy loss with label-smoothing [34, 35] with $\alpha = 0.1$. We also use cosine learning rate decay [36] with an initial learning rate of 0.01. For data augmentations, we perform horizontal flip and random crop with a padding of 4. We use layer threshold $L = 23$ for both LLF and our method (the third block in ResNet50). This means that in LLF, the first two blocks are considered the fit hypothesis $\mathbf{H}_{fit}$, while the fit hypothesis in our method is the last two blocks. In all experiments, normal training refers to $G \times E$ epochs of training with the same optimization settings.

For our few shot learning evaluation, we evaluated our models in an episodic fashion. Each episode has a train set of 5 classes with K examples each (5-way K-shot) and a test set with 15 examples for each class. These sets are sampled from a target dataset. Models are fine tuned on the train set and then used to produce predictions on the test set. The accuracies are reported over 600 episodes. Cross Domain Few Shot Learning Benchmark (CD-FSL) [37] was selected as the dataset for the task, which includes data from four different data sets, namely CropDiseases [38], EuroSAT [39], ISIC2018 [40, 41], and ChestX [42].

Overall we compare our method with the following three approaches:

Method	Tiny-ImageNet	Flower	CUB	Aircraft	MIT	Dogs
Normal	54.37	34.31	6.49	6.24	25.67	8.99
Normal (long)	49.27	26.96	8.07	6.30	24.85	11.53
LLF	56.92	22.84	5.33	4.65	23.8	8.69
SEAL (Ours)	**59.22**	**45.68**	**8.49**	**9.81**	**35.37**	**12.61**

Table 1: Transfer learning accuracy of models trained on tiny-imagenet, fine tuned on other datasets using linear probe. Normal, and Normal (long) refer to $G = 1$ and $G = 10$ generations of training, respectively. LLF and SEAL were trained for $G = 10$ generations. Transfer accuracy of LLF after $1,600$ epochs is significantly lower than normal training with both 160 and $1,600$ epochs; our method after $1,600$ epochs surpasses normal training by a large margin. This demonstrates that our method learns much more generalizable features compared to Normal training and LLF.

Normal denotes the standard training with 160 epochs (corresponding to $G = 1$ generations under the conventions of iterative training).
Normal (long) refers to training the model with the standard settings for $1,600$ epochs (corresponding to $G = 10$ generations without any forgetting).
LLF refers to fortuitous forgetting [16] where at the beginning of each generation the later layers of the network are re-initialized.
SEAL refers to our proposal which performs a gradient ascent and subsequent descent phase during a generation.

Generation	Normal	LLF	Ours
Gen=1	54.37	-	-
Gen=3	51.16	56.12	**58.25**
Gen=10	49.27	56.92	**59.22**

Table 2: Comparison of our method with normal training and LLF on Tiny-ImageNet. Please note that the behavior of the first generation for all methods is the same. We significantly outperform standard long training and LLF.

Datasets Tiny-ImageNet consists of 200 classes and has $100,000$ training and $10,000$ validation images selected from the ImageNet dataset. These images are downsized to 64x64 colored images. For our transfer learning evaluations we use the natural image datasets Flower, CUB, Aircraft, MIT, and Stanford Dogs, the statistics of these datasets are provided in Table 13.

	num_classes	Train	Valid	Test	Total
Flower [43]	102	1,020	1,020	6,149	8,189
CUB [44]	200	5,994	N/A	5,794	11,788
Aircraft [45]	100	3,334	3333	3,333	10,000
MIT [46]	67	5,360	N/A	1,340	6,700
Dogs [47]	120	12,000	N/A	8,580	20,580
CIFAR-100 [48]	100	50,000	N/A	10,000	60,000

Table 3: Summary of the datasets used in tables 2 and 1, adopted from Taha et al. [2].

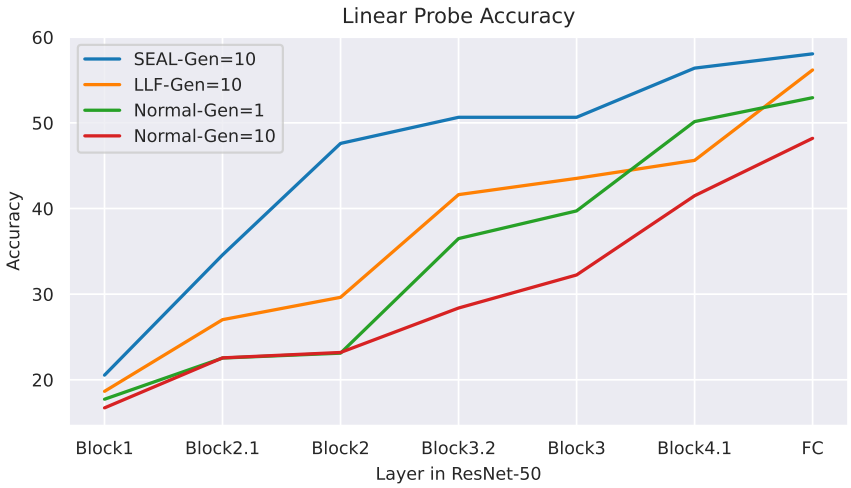

Figure 4: Comparison of layer-wise prediction depth. SEAL gives comparably much stronger predictions early on in the network. Note that block.X.Y denotes the output activations of intermediate layer Y in residual block X. This indicates that our method encourages learning the difficult examples using conceptually simpler and more general features of the early layers. This leads to better overall performance as we progress deeper into the network.

For the few-shot learning, we utilized the 4 datasets from the CD-FSL benchmark, which include ChestX, ISIC, CropDisease, and EuroSAT. From these, we sample training and evaluation sets with 5 classes and a varying number of samples per class.

6.1.2 Evaluation

We now present our primary evaluations of SEAL for both improved generalization, transfer, and few-shot transfer learning.

In-Distribution Generalization: Table 2 compares the in-distribution accuracy of our method with the state-of-ther-art iterative training method, namely, LLF.

We note that LLF outperforms Normal training (as reported in [16]). Our method significantly outperforms both LLF and normal training in this setting. We conduct our experiment using the same hyper-parameters used in LLF to ensure fairness.

Furthermore, we compare our method with both classical self-distillation methods such as classical born-again neural networks [3, 4], and a state-of-the-art self-distillation method [5] in an in-distribution setting.

Yang et al. [4] conducted experiments on CIFAR-100 [48] using a 110-layers ResNet. CIFAR-100 consists of $60,000$ RGB images of 32×32 resolution, with $50,000$ training examples and $10,000$ test samples. They choose this dataset over CIFAR-10 since CIFAR-10 does not contain fine-level classes, and thus self-distillation methods do not bring significant gains. For training, they use SGD with a weight decay of 1e$-$4, and a momentum of 0.9, using batch sizes of 128. Each generation is trained for $E = 164$ epochs with a base learning rate of 0.1 which is then divided by 10 after 150 and 225 epochs. They also use standard data augmentation techniques during training such as padding of four pixels, and random cropping, and horizontal flipping with a probability of 50% (without any augmentation during the test phase). Using this exact setting, we compare our method with their method in table 4. We observe that compared to Furlanello et al. [3], our method outperforms them consistently by the end of each generation. Furthermore, we compare our best model after 10 generations with that of [4] (each generation is trained for the same number of epochs in both works). Our method demonstrates 1.71% higher accuracy compared to [4] under this setting.

Generation	**Furlanello et al. [10]**	**Yang et al. [41]**	**Ours**
Gen=0	71.55	–	–
Gen=1	71.41	–	**72.83**
Gen=2	72.30	–	**73.53**
Gen=3	72.26	–	**73.88**
Gen=4	72.52	–	**74.18**
Gen=10 (Best)	72.61	73.72	**75.43**

Table 4: Comparison with [3, 4] on CIFAR100 using ResNet-110 with inferior hyper-parameter settings. The last row lists the best accuracy of each method throughout 10 generations. The hyperparameters and baseline accuracies are adopted (and not changed) from [4] to ensure fairness.

Pham et al. [5] observed that the hyper-parameter settings used in previous self-distillation methods are not properly tuned, and hence there is a gap between their teachers' performance and their capacity. Therefore, they question whether self-distillation methods only work when the teacher is far from their capacity, or do they always enhance the teacher, even when the teacher is properly trained. To this end, they trained a teacher with well-tuned hyperparameters using ResNet-18 on the CIFAR-100 dataset. Precisely, they use SGD with a weight decay of 3e−4, and a Nestrov momentum of 0.9, using batch sizes of 96 with standard data augmentation techniques. Using the same hyperparameter setting, SEAL outperforms self-distillation techniques by a 1.18% margin (refer to table 5.

Therefore, our method outperforms self-distillation methods in both an inferior hyper-parameter setting [3, 4] and a well-tuned one [5].

Generation	**Pham et al. [5]**	**SEAL (Ours)**
Gen=0 (Teacher)	76.30	76.15
Gen=Last (Student)	77.32	**78.50**

Table 5: Comparison with Pham et al. [5] on CIFAR100 using ResNet-18 with a well-tuned hyper-parameter setting. We train our method for the same number of epochs and under the same hyper-parameter setting as [5] to ensure fairness.

Transfer Learning We now turn to evaluate the transfer learning properties of our method and that of LLF. We begin by studying the transfer learning from the Tiny-ImageNet pretrained models to 5 different image datasets. Specifically, CUB-200-2011 (CUB) [44] contains images of 200 wild bird species, Flower102 (Flower) [43] contains images from 102 flower categories, FGVC-Aircraft (Aircraft) [45] consists of 100 aircraft model variants, and Stanford Dogs dataset contains images of 120 breeds of dogs for fine-grained classification. MIT Indoor 67 (MIT) [46] is an indoor scene recognition dataset that includes 67 different scene classes.

We train linear models on top of pretrained models from each method to measure their transfer learning properties. Specifically, we re-initialize the last linear layer of the network and freeze the rest. Then, we train the linear head using the train set of the target datasets and evaluate on their test sets. For training on the target dataset, we again use SGD with a momentum of 0.9, weight decay of 1e−4, and flat learning rates of [1e−1, 1e−2, 1e−3] for 120 epochs and report the highest accuracy.

Table 1 demonstrates the transfer accuracy of LLF, normal training, and our

Updates	Base Model	ChestX	ISIC	EuroSAT	CropDisease
		5-way, 1-shot			
Linear	Normal	21.01 ± 0.35	25.7 ± 0.47	53.41 ± 0.92	50.31 ± 1.03
	Normal (long)	20.40 ± 0.23	22.59 ± 0.35	36.60 ± 0.75	27.50 ± 0.69
	LLF	20.38 ± 0.24	24.61 ± 0.46	36.58 ± 0.86	26.28 ± 0.68
	SEAL (ours)	**21.67 ± 0.36**	**27.93 ± 0.55**	**57.75 ± 0.88**	**63.64 ± 0.96**
LA	Normal	21.14 ± 0.35	26.71 ± 0.56	47.36 ± 1.26	64.75 ± 1.13
	Normal (long)	20.90 ± 0.37	26.41 ± 0.54	45.51 ± 1.25	63.73 ± 1.05
	LLF	20.24 ± 0.24	23.28 ± 0.45	34.02 ± 1.35	35.12 ± 1.61
	SEAL (ours)	**21.3 ± 0.39**	**29.14 ± 0.57**	**55.68 ± 1.05**	**67.87 ± 0.54**
		5-way, 5-shot			
Linear	Normal	22.79 ± 0.36	33.28 ± 0.49	71.74 ± 0.75	77.05 ± 0.80
	Normal(long)	21.00 ± 0.28	28.93 ± 0.48	53.40 ± 0.77	57.10 ± 1.07
	LLF	22.03 ± 0.33	29.60 ± 0.48	60.69 ± 0.87	53.55 ± 1.12
	SEAL (ours)	**24.42 ± 0.42**	**37.58 ± 0.54**	**74.61 ± 0.65**	**85.00 ± 0.61**
LA	Normal	20.82 ± 0.26	28.57 ± 0.78	53.35 ± 1.74	63.22 ± 2.35
	Normal(long)	21.59 ± 0.34	29.67 ± 0.82	55.79 ± 1.67	71.55 ± 2.00
	LLF	20.44 ± 0.19	22.01 ± 0.44	30.83 ± 1.45	28.06 ± 1.47
	SEAL (ours)	**22.98 ± 0.36**	**39.64 ± 0.79**	**73.83 ± 0.93**	**88.24 ± 0.54**

Table 6: Few-shot transfer results for the CFSDL benchmark (extreme distribution shift) for 1 and 5 shots. All methods make use of a ResNet50 backbone trained on Tiny-ImageNet evaluated over 600 episodes. We consider finetuning both the linear layer (Linear) and the linear layer and affine parameters (LA), the best performer in both categories highlighted in red. We observe that SEAL outperforms standard training even for a very long number of epochs, while LLF under-performs.

method (SEAL). Even though LLF outperforms normal training with a 2.55% margin in the in-distribution setting, we observe that in transfer learning, LLF's performance is substantially lower than normal training for 1/10 of epochs ($G = 1$), as well as normal training for the same number of epochs ($G = 10$). On the other hand, our method not only dominates in the in-distribution setting, but it also has a much stronger transfer learning performance than both LLF and normal training across all of the target datasets in our experiments.

Few-Shot Transfer Learning We now consider evaluating our approach for a challenging distant transfer learning task studied in [49, 50]. Here we are presented with a few shot learning problem on multiple datasets from medical imaging to satellite images, with only one dataset of natural images available for pre-training. Several approaches to this problem exist, some utilizing meta-learning methods [51] and others focused on transfer learning [50]. It has been shown in multiple works

Updates	Base Model	ChestX	ISIC	EuroSAT	CropDisease
		5-way, 20-shot			
Linear	Normal	25.16 ± 0.37	41.24 ± 0.50	79.14 ± 0.67	86.74 ± 0.57
	Normal (long)	21.79 ± 0.27	32.85 ± 0.50	60.95 ± 0.81	70.20 ± 1.01
	LLF	22.54 ± 0.29	32.26 ± 0.48	67.79 ± 0.85	68.43 ± 1.07
	SEAL (ours)	**27.44 ± 0.40**	**46.96 ± 0.53**	**82.45 ± 0.53**	**92.47 ± 0.39**
LA	Normal	22.91 ± 0.36	49.40 ± 1.14	81.52 ± 1.38	87.43 ± 1.67
	Normal (long)	23.38 ± 0.37	47.36 ± 1.25	80.03 ± 1.53	89.63 ± 1.53
	LLF	21.10 ± 0.25	30.54 ± 1.22	46.55 ± 2.42	39.80 ± 2.30
	SEAL (ours)	**26.99 ± 0.46**	**55.12 ± 0.77**	**87.70 ± 0.52**	**95.67 ± 0.28**
		5-way, 50-shot			
Linear	Normal	26.55 ± 0.36	46.29 ± 0.47	82.20 ± 0.61	90.51 ± 0.45
	Normal (long)	22.56 ± 0.29	36.57 ± 0.53	67.14 ± 0.77	80.60 ± 0.78
	LLF	23.78 ± 0.31	35.59 ± 0.5	73.76 ± 0.79	79.67 ± 0.78
	SEAL (ours)	**29.78 ± 0.40**	**51.46 ± 0.50**	**84.99 ± 0.52**	**94.91 ± 0.29**
LA	Normal	24.2 ± 0.40	60.27 ± 1.10	88.09 ± 1.30	93.80 ± 1.33
	Normal (long)	24.62 ± 0.45	56.98 ± 1.32	85.17 ± 1.58	89.8 ± 1.9
	LLF	22.56 ± 0.34	41.53 ± 1.65	58.69 ± 2.76	51.35 ± 2.92
	SEAL (ours)	**27.13 ± 0.45**	**60.59 ± 1.11**	**91.94 ± 0.53**	**97.91 ± 0.40**

Table 7: Low-shot transfer results for the CFSDL benchmark (extreme distribution shift) for 20 and 50 shots. We fine-tuned both the linear layer (Linear) and the linear layer along with affine parameters (LA). We observed that SEAL outperforms standard training and that LLF severely under-performs in this case. Tuning the affine parameters and linear layer provides consistent performance gains for both Normal training models and SEAL.

that for this distant few shot learning task, transfer learning approaches exceed meta-learning [50]. We use our Tiny-ImageNet models from the previous section as base model for FSL transfer. In transfer we train a linear head as in the standard protocol [49, 50], we also consider jointly training linear head and affine parameters as suggested by [50]. Results of our evaluations for 1, 5, 20, and 50 shots are shown in Table 6 and Table 7. We observe that in this challenging benchmark LLF substantially underperforms standard training, suggesting features learned by LLF do not generalize well. We note that the training accuracies on Tiny-Imagenet of all the suggested models are 100% and the testing accuracies are the ones shown in Table 2. Although LLF has higher in-distribution performance than normal training, its FSL transfer properties are much worse. On the other hand, SEAL features generalize both in-distribution and to the distant FSL tasks.

Evaluating on Smaller Models: In this experiment, we evaluate both the in-domain and transfer learning performance of SEAL, LLF, and normal training using ResNet-18 on Tiny-ImageNet. We do not modify any of the hyper-parameters that were used for ResNet-50. We observe that SEAL outperforms both LLF and Normal training on this model as well (Table 8).

Generation	Normal	LLF	Ours
Gen=1	50.47	-	-
Gen=3	48.32	52.36	**53.69**
Gen=10	46.66	53.64	**54.75**

Table 8: Comparison of our method with normal training and LLF on Tiny-ImageNet with ResNet-18. Please note that the behavior of the first generation for all methods is the same. We outperform standard long training and LLF on ResNet-18 as well.

Furthermore, we transfer the models trained on ResNet-18 to multiple datasets. We observe the same performance improvements as we saw with ResNet-50 with this smaller model as well (Table 9).

Method	Tiny-ImageNet	Flower	CUB	Aircraft	MIT	Dogs
Normal	50.47	31.47	7.47	7.14	28.20	11.85
Normal (long)	46.66	19.11	5.36	4.80	21.71	8.17
LLF	53.64	31.66	7.19	6.09	25.67	11.64
SEAL (Ours)	**54.75**	**40.68**	**9.87**	**8.85**	**33.65**	**14.61**

Table 9: Transferring features learned from Tiny-ImageNet with ResNet-18 to other datasets using linear probe. Normal, and Normal (long) refer to $G = 1$ and $G = 10$ generations of training, respectively. LLF and SEAL were trained for $G = 10$ generations. Our method, after $1,600$ epochs, surpasses both LLF and normal training by a large margin. This demonstrates that our method learns much more generalizable features as compared to Normal training or LLF.

6.1.3 Analysis and Ablation Studies

In this section we present additional analysis of our method, specifically we study the effects on the prediction depth as well as on the eigenvalues of the Hessian at the end of model training. Finally, we perform ablation studies to demonstrate that the early layers indeed benefit the most from SEAL.

Analysis of Hessian Eigenvalues We first study the eigenvalue spectra of the Hessian for the different proposed models. We utilize the same process for estimating the eigenvalues suggested in [52]. The results for the 4 models are summarized in Table 10 where we report both the maximum eigenvalues and the percentage of negative eigenvalues. We observe that the maximum eigenvalues are smaller for SEAL than for normal long training and also for LLF, suggesting a flatter minimum. Flatter minima have been previously associated with improved generalization [53].

We further observe that SEAL has no negative eigenvalues. This suggests that SEAL obtains some of its advantages by helping to avoid saddle points during training. This is consistent with prior uses of simulated annealing [54]. Following [52] we hypothesize the absence of negative eigenvalues can indicate a more robust solution.

Method	**Max Eigenval**	**Negative % Eigenval**
Normal	889.06	4.25%
Normal (long)	2353.74	0%
LLF	1027.21	14.63%
SEAL (Ours)	847.89	0%

Table 10: In this table, we demonstrate the statistics of the Hessian eigenvalues. We observe that our method has a lower max eigenvalue which suggests flatter local minima. Furthermore, our method has no negative eigenvalues, suggesting SEAL can avoid saddle points.

Prediction Depth: Zhou et al. [16] proposed LLF to specifically enhance the prediction depth of the model. Their intuition was that periodically resetting the final layers would decrease the prediction depth. Following Zhou et al. [16], we approximate the prediction depth using the K Nearest Neighbor (KNN) probe (with $K = 5$) on different layers of the network. To do so, for every image in the test set, we use all of the images in the train set for the KNN.

We affirm that the prediction depth of LLF is improved over normal training (Figure 4). However, with SEAL, we achieve a much stronger prediction depth. For instance, the KNN accuracy of our method is more than 18.54% stronger than LLF and 25.04% stronger than normal training on the outputs of the second block of the network. This is the layer threshold L used in our method and LLF. The layer-wise accuracy of the other layers indicates the superiority of our method across all layers.

Baldock et al. [55] demonstrated that example difficulty is correlated with prediction depth; decreasing the prediction depth corresponds to lower example

difficulty, which is desired. They show this correlation by analyzing the speed of learning, the input and output margin, and the adversarial input margin for each data point. In Figure 5, we measure the prediction depth evolution of the three methods over different generations ($G = [1, 2, 4, 10]$). For normal training, the prediction depth gets worse over time, which explains its poor in-distribution performance. This suggests that in normal training, the early layers are becoming weaker after $G = 1$ and more samples are being classified by the later layers.

LLF slightly improves the prediction depth of the model. However, this comes with a deterioration of the performance of the later layers. For instance, the KNN probe on the activations of Block 4.1 in $G = 1$ has 50.15% accuracy which decreases to 45.62 in $G = 10$. On the other hand, our method does a better job of pushing more examples to be classified in the early layers than normal training and LLF. This implies that SEAL promotes relearning the more difficult samples using the simpler and more general features of the early layers. Furthermore, the later layers of the network improve over time with our method.

Ablation Study: We now investigate different strategies for the fit hypothesis $\mathbf{H}_{fit}$ during the forgetting phase. By default, during this phase, we perform gradient descent on the fit hypothesis and gradient ascent on the forgetting hypothesis. In "Ours+Reinit", following LLF [16], at the beginning of the forgetting phase, we reinitialize the fit hypothesis and during this phase, we perform gradient descent on these layers. We observe that not using the re-initialization leads to higher accuracy. Further, we demonstrate that freezing the final layers during the forgetting phase has a negative impact on the training. Finally, in "Ours+Reverse", we swap the fit and forgetting hypotheses, where we perform the gradient ascent on the later layers. We observe that performing simulated annealing in later layers fails drastically. This shows the importance of promoting the early layers and affirms the observations of Baldock et al. [55].

Furthermore, we demonstrate that **SEAL is not sensitive to forgetting frequencies**. To this end, in an experiment, we evaluate the sensitivity of SEAL to different forgetting frequencies. For this, we test multiple values for the number of epochs per generation E. The fewer the number of epochs in a generation more is the number of forgetting stages. We do not modify any other hyper-parameter. As summarized in Table 12, we observe that our method is not sensitive to the forgetting frequency and significantly improves over Normal training with any forgetting frequency. Please note that in this experiment, the maximum number of

Gen	Normal	SEAL+Freeze	SEAL+Reinit	SEAL+Reverse	SEAL+Descent
Gen1	54.37	-	-	-	-
Gen3	51.16	52.45	56.82	50.25	**58.25**
Gen10	49.27	51.17	58.87	41.05	**59.22**

Table 11: Fitting hypothesis $\mathbf{H}_{fit}$ ablation study. While performing gradient ascent on the early layers during forgetting, we freeze, reinitialize, and perform gradient descent on the later layers. In reverse, we swap the fit and forgetting hypotheses. We observe that doing gradient descent on the fitting hypothesis during the forgetting phase leads to the best performance.

training epochs is different as each model is trained for E epochs for 10 generations.

Gen	E=160(default)	E=60	E=70	E=80	E=90	E=100	E=120	E=200
Gen1	54.37	53.33	53.62	53.59	53.66	53.84	53.92	54.07
Gen3	58.25	54.00	55.36	57.04	57.8	57.21	57.59	57.78
Gen10	**59.22**	**56.56**	**57.49**	**58.35**	**58.37**	**59.36**	**59.50**	**59.47**

Table 12: Dependency of SEAL on forgetting frequency in ResNet-50. Numbers in the columns indicate the number of epochs per generation E. Every E epochs, we perform gradient ascent for $k = \frac{E}{4}$ epochs. Each model is trained for $G = 10$ generations. We can see that our method has a significant positive impact on every forgetting frequency.

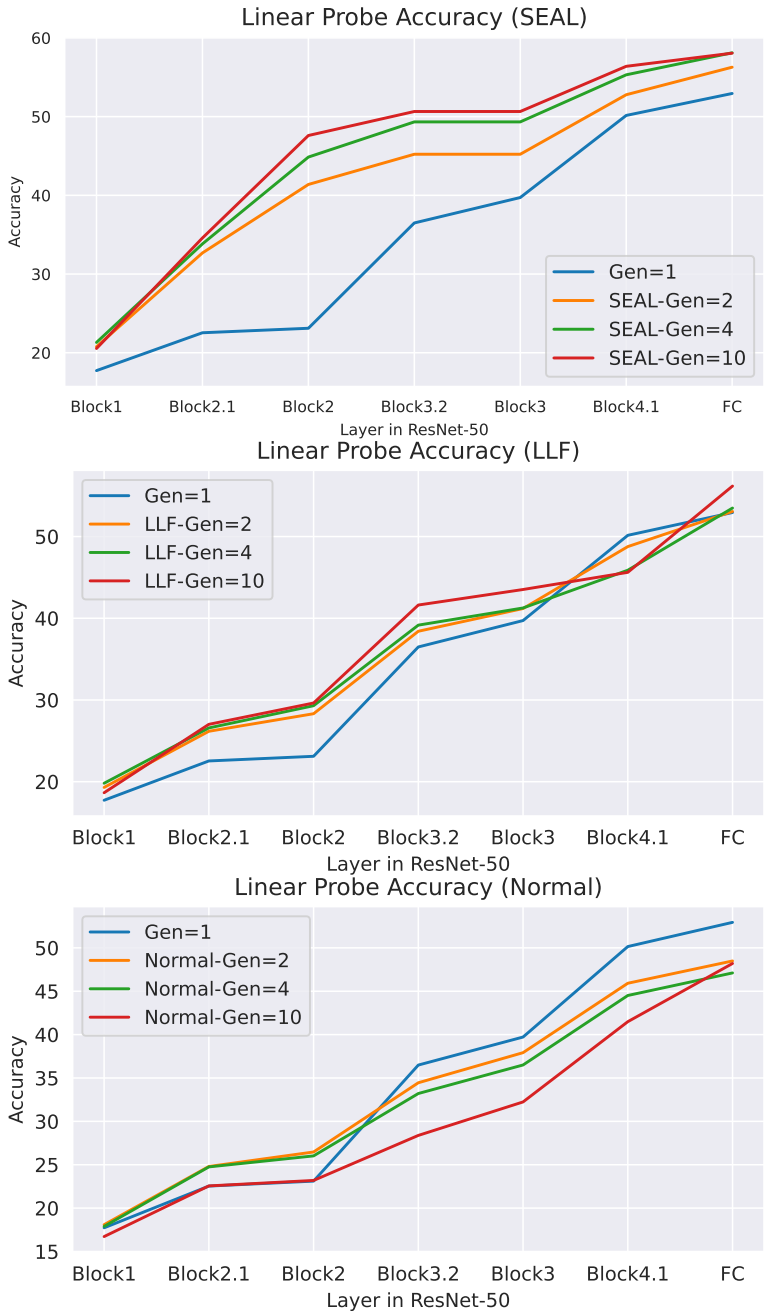

Figure 5: Evolution of Prediction Depth over Epochs for LLF, SEAL, and Normal training. Normal training worsens the prediction depth after the first generation which explains its poor in-distribution performance. LLF slightly improves the prediction depth of the model, however, it hurts the performance of the later layers of the network. SEAL shows the most significant improvement in prediction depth, while the later layers are improving over time.

6.2 Encouraging Information Gain in Early Layers for Greedy Training

6.2.1 Implementation Details

We conduct experiments on the CIFAR-10 [56], CIFAR-100 [57], and Tiny-ImageNet [58] datasets, which are widely used benchmark datasets for image classification tasks. These datasets consist of RGB images with varying numbers of classes. Table 13 presents additional statistics of the datasets. We use a predefined train and test split for each dataset during the training process.

To preprocess the train set, we first perform a RandomCrop with padding to the size of 32x32, and then randomly flip the images horizontally. Finally, we normalize the images. For the test set, only normalization is applied.

	CIFAR-10	**CIFAR-100**	**Tiny-ImageNet**
num_cls	10	100	200
Train	50,000	50,000	100,000
Test	10,000	10,000	10,000
Resolution	(32, 32)	(32, 32)	(64, 64)

Table 13: Statistics of the used datasets.

Following Belilovsky et al. [19], we train the models using an SGD optimizer with a momentum of 0.9, a weight decay of $5e-4$, and a batch size of 32. Additionally, we schedule learning rate decay with an initial learning rate of 0.1, reducing it every 15 epochs by $lr/5$, and train for 50 epochs. Furthermore, each block B^i of the network starts with a downsampling if specified, which is then followed by a stack of a convolutional layer, batch norm, and Relu activation function. The auxiliary network always ends with an average pooling and linear layer that makes the predictions using the features of the network. The auxiliary network can also have a depth d that denotes the convolutional depth of the auxiliary network. For instance, if $d = 0$, the auxiliary network consists of only a linear layer, whereas if $d = K$, the auxiliary network begins with a stack of K convolutional layers (followed by batch norm and Relu), and finally end with the typical average pooling and linear layer. Please note that the kernel size of all convolutional layers used in all the experiments is 3.

Method	Accuracy	Ensemble Acc
CONFIG: k = 5, d = 0		
Normal (CE)	86.73	-
Normal (CS-KD)	87.84	-
LayerCNN [19] (CE-All)	88.07	88.20
Ours (L = 1)	**88.23**	**88.33**
Ours (L = 2)	87.18	87.47
Ours (L = 3)	87.28	87.41
Ours (L = 4)	87.19	87.11
Ours (L = All)	86.77	86.50
CONFIG: k = 4, d = 2		
Normal (CE)	**91.16**	-
Normal (CS-KD)	90.93	-
LayerCNN [19] (CE-All)	91.00	**91.77**
Ours (L = 1)	90.54	91.11
Ours (L = 2)	90.47	90.67
Ours (L = 3)	90.87	91.27
Ours (L = All)	90.69	91.20

Table 14: In-distribution performance on CIFAR-10 of networks with different numbers of blocks k and auxiliary depths d.

6.2.2 Evaluation

We assess the performance of our models based on their accuracy, which measures the number of correct predictions out of the total number of samples. We distinguish between two types of accuracy: the end-of-training accuracy and the ensemble accuracy, which corresponds to the highest accuracy achieved by any block.

In-Distribution Generalization Tables 14, 15, 16, and 17, demonstrates the in-distribution performance of normal training, LayerCNN, and our method across different datasets and settings. In these tables, CE refers to cross-entropy loss, and in ours, $L = i$ denotes how many early layers of the network are optimized by CS-KD, where $L = \mathit{All}$ means all layers use this regularizer. We observe that there is no significant difference between our method, normal training, and LayerCNN [19] in an in-distribution setting.

Transfer Learning Tables 16, 17 show the transfer learning performance of the three methods. First, we transfer the models pre-trained on the Tiny-ImageNet

Method	Accuracy	Ensemble Acc
Normal (CE)	68.95	-
Normal (CS-KD)	**72.29**	-
LayerCNN [19] (CE-All)	69.90	**72.23**
Ours (L = 1)	69.03	70.11
Ours (L = 2)	69.29	70.56
Ours (L = 3)	69.74	70.85
Ours (L = All)	71.25	71.89

Table 15: In-distribution accuracies of a model configured with $k = 4$ blocks and $d = 1$ auxiliary depth on CIFAR-100 dataset. We observe close results across all three methods.

dataset (table 16), where the network had $k = 5$ blocks, auxiliary depth of $d = 1$, and the 2nd and 3rd blocks of the network perform downsampling. In this experiment, we observe that although LayerCNN has better in-distribution performance than that of normal training with Cross-Entropy, it has 1.52% and 1.91% lower transfer accuracy on CIFAR-10, and CIFAR-100, respectively. On the other hand, our method outperforms normal training with Cross-Entropy, and demonstrates comparable results to that of normal training with CS-KD while having all the benefits of LayerCNN, such as being more efficient in memory and computational consumption and being more interpretable.

Second, we transfer models pre-trained on the CIFAR-10 dataset (table 17), wehere the network had $k = 4$ blocks, and auxiliary depth of $d = 1$, and only the 2nd block performs downsampling. We observe significantly worse transfer generalization performance in LayerCNN. For example, compared to conventional cross-entropy training, LayerCNN achieved transfer accuracies that were 3.67% and 4.70% lower on the CIFAR-100 and Tiny-ImageNet datasets, respectively. On the other hand, our method exceeds normal training with cross-entropy on these datasets and demonstrates very competitive transfer results to that of normal training with CS-KD.

Another observation here is that the transfer performance of the model using our method is best when the layer threshold is more towards the middle of the network, compared to all layers performing CS-KD ($L = ALL$) or none (LayerCNN). This affirms that it is best to encourage early layers to optimize for information gain and let the layers focus on the task at hand [28].

	In-Distribution		Transfer	
Method	**Acc**	**Ens. Acc**	**CIFAR-10**	**CIFAR-100**
Normal (CE)	47.38	-	<u>79.57</u>	57.60
Normal (CS-KD)	**56.87**	-	**80.05**	**60.07**
LayerCNN [19] (CE-All)	50.16	53.61	78.05	55.69
Ours (L = 1)	50.66	52.72	77.77	57.55
Ours (L = 2)	50.31	52.20	78.32	56.63
Ours (L = 3)	51.19	54.72	79.28	<u>58.09</u>
Ours (L = 4)	<u>54.06</u>	<u>55.66</u>	79.14	57.58
Ours (L = All)	51.82	**56.23**	78.05	58.02

Table 16: This table demonstrates the in-distribution and transfer generalization of models trained on Tiny-ImageNet dataset. The network was configured to have $k = 5$ blocks with auxiliary depth of $d = 1$. We observe that the in-distribution accuracy of our method out performs LayerCNN and normal training using Cross-Entropy loss function by a large margin. Furthermore, LayerCNN has much worse transfer performance compared to normal training, while our method outperforms LayerCNN and Normal (CE) by a large margin, achieving comparable results to that of Normal (CS-KD).

	In-Distribution		Transfer	
Method	**Acc**	**Ens. Acc**	**CIFAR-100**	**Tiny-ImageNet**
Normal (CE)	**90.38**	-	56.91	36.07
Normal (CS-KD)	<u>90.33</u>	-	**57.91**	35.81
LayerCNN [19] (CE-All)	90.03	<u>90.50</u>	53.24	31.37
Ours (L = 1)	89.23	89.64	56.25	36.12
Ours (L = 2)	89.38	89.79	<u>57.18</u>	<u>36.25</u>
Ours (L = 3)	89.85	90.24	56.90	**36.48**
Ours (L = All)	90.25	**90.64**	56.51	35.50

Table 17: CIFAR-10 in-distribution and transfer accuracies with $k = 4$ blocks and auxiliary depth $d = 1$. LayerCNN lags behind normal training in transfer learning, while our method achieves competitive performance.

6.3 Greedy Federated Learning

6.3.1 Implementation Details

We conduct experiments on the CIFAR-10 [56] dataset, which is a widely used benchmark dataset for image classification tasks. It consists of RGB images with 10 class, 50,000 train and 10,000 test images of (32,32) resolution. We use a predefined train and test split for each dataset during the training process. To preprocess the train set, we first perform a RandomCrop with padding to the size of 32x32 and occasionally flip the images horizontally. Finally, we normalize the images. For the test set, only normalization is applied.

Following Belilovsky et al. [19] and Wang et al. [24], we train the models using an SGD optimizer with a momentum of 0.9, a constant learning rate of 0.01, and a batch size of 128. Furthermore, each block B^i of the network starts with a downsampling if specified, which is then followed by a convolutional layer, batch norm, and Relu activation function. The downsampling method is maxpooling layers unless specified otherwise. The auxiliary network always contains an average pooling and linear layer that makes the predictions using the features of the network. Note that the kernel size of all convolutional layers used in all the experiments is 3.

We conducted our experiments on 3 different model sizes:

- **Small model**: We use 3 convolutional blocks with downsampling in layers 1 and 2.
- **Medium model**: We use a 6-layered model with downsampling after layers 1, 2, and 4.
- **Large model**: We use VGG16 [59] with downsampling on layers 2, 4, 7, 10 and 13.

In our experiments, we compare the normal and greedy training of each client under i.i.d. federated settings. We conduct our experiments with 1, 30, and 100 clients N where each client is trained for 8, 20, and 40 epochs E for 120 communication (server) epochs and report the maximum accuracy. Note that the number of clients refers to training the models normally without any federated learning, as training only 1 client is a normal learning regime. Also, at each server update, 10% of the clients are randomly selected.

		E=8		E=20		E=40	
Size	**Clients**	**Normal**	**Greedy**	**Normal**	**Greedy**	**Normal**	**Greedy**
Small	1	82.56	80.93	**83.36**	80.67	83.02	80.57
	30	62.63	58.19	65.85	64.39	65.45	**69.23**
	100	58.26	56.83	55.35	59.81	51.04	**66.01**
Medium	1	87.20	81.74	87.22	81.88	**87.29**	82.84
	30	56.14	55.23	56.27	63.70	51.04	**66.80**
	100	59.71	49.28	60.24	59.69	54.24	**62.04**
Large	1	89.41	80.77	89.16	81.15	**90.58**	80.46
	30	52.04	27.09	30.79	66.51	38.32	**69.71**

Table 18: **Comparison of normal and greedy training in FL.** E denotes the number of local client updates. As the number of clients and local update intervals increase, normal training declines, while greedy training improves. Note that Clients = 1 represents a normal (non-federated) learning regime.

6.3.2 Evaluation

Our main experiment demonstrates that normal training outperforms greedy training in non-federated learning scenarios. However, greedy training significantly outperforms normal training in federated learning scenarios, which is severe in bigger model sizes or as the number of clients and local updates increase (table 18). Belilovsky et al. [19] used invertible downsampling [6] in place of maxpooling layers as maxpooling strongly encourages loss of information. In table 19, we demonstrate that the same pattern consistently persists across different experimental settings.

		E=8		E=20		E=40	
Size	**Clients**	**Normal**	**Greedy**	**Normal**	**Greedy**	**Normal**	**Greedy**
Small	1	83.89	80.84	83.66	81.41	**84.04**	81.52
	30	57.15	54.77	60.73	62.10	63.35	**66.49**
	100	51.17	51.24	48.90	63.25	49.52	**63.70**
Medium	1	87.07	81.94	**87.17**	83.35	86.26	82.29
	30	49.58	48.87	39.79	**65.33**	41.54	65.11
	100	48.74	41.31	44.82	58.58	40.68	**60.57**

Table 19: **Comparison of normal and greedy training in FL using invertible downsampling [6].** While results are slightly worse than maxpooling (Table 18), the same trend holds: under complex settings, greedy training outperforms normal training.

Chapter 7

Conclusions and Future Work

This thesis explored the role of early layer optimization in improving generalization across various training paradigms. We began by introducing Simulated Annealing in Early Layers(SEAL), a novel iterative training technique that applies intermittent gradient ascent to the early layers of a network. Inspired by simulated annealing, SEAL enables the model to escape poor local minima by periodically "heating" and then gradually "cooling" the early representations. Compared to the previous state-of-the-art in iterative training, namely Later-Layer Forgetting (LLF), we observed that SEAL not only shows stronger in-distribution performance, but it also significantly outperforms LLF in transfer generalization.

In the second part of the thesis, we focused on greedy layer-wise training and its generalization behavior. While greedy methods such as LayerCNN can match conventional training in in-distribution settings, we observed a significant drop in their transfer performance. We attributed this issue to weakened generic early-layer representations, as they tend to overfit to class-specific features due to the use of cross-entropy loss in all layers. To address this, we proposed enhancing early layers with CS-KD regularization, which encourages information gain, while keeping later layers trained with standard cross-entropy. This adjustment substantially improved the transfer performance of greedy training, matching or even surpassing that of end-to-end training using cross-entropy or CS-KD in transfer learning tasks. This result suggests a promising direction for improving greedy training which we intend to explore further.

Finally, [24] demonstrated that even in IID centralized federated learning, the accumulation of gradients leads to divergence in early layers during end-to-end optimization, resulting in poor representation learning in these layers. To tackle this gradient accumulation problem, we proposed the use of greedy training in this setting,

as it avoids the multiplicative effect of gradient accumulation caused by end-to-end backpropagation. We showed that greedy training can mitigate early layer divergence and improve overall performance in scenarios with large numbers of clients, many local updates, and deeper models.

In summary, this thesis underscores the importance of early layer learning in improving both generalization and stability across diverse deep learning paradigms. While we introduced practical methods to enhance early layer representations, we encourage future work to theoretically investigate their role. Furthermore, while we opened the door to using greedy training in centralized federated learning, we believe it is also well-suited for many other federated learning scenarios, which we intend to investigate in the future.

References

[1] A. Sarfi, Z. Karimpour, M. Chaudhary, N. Khalid, M. Ravanelli, S. Mudur, and E. Belilovsky, "Simulated annealing in early layers leads to better generalization," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2023. iv, 13, 14, 15

[2] A. Taha, A. Shrivastava, and L. S. Davis, "Knowledge evolution in neural networks," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 12843–12852, 2021. ix, 8, 11, 24

[3] T. Furlanello, Z. Lipton, M. Tschannen, L. Itti, and A. Anandkumar, "Born again neural networks," in *International Conference on Machine Learning*, pp. 1607–1616, PMLR, 2018. ix, 8, 25, 26

[4] C. Yang, L. Xie, S. Qiao, and A. L. Yuille, "Training deep neural networks in generations: A more tolerant teacher educates better students," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, pp. 5628–5635, 2019. ix, 8, 25, 26

[5] M. Pham, M. Cho, A. Joshi, and C. Hegde, "Revisiting self-distillation," *arXiv preprint arXiv:2206.08491*, 2022. ix, 25, 26

[6] L. Dinh, J. Sohl-Dickstein, and S. Bengio, "Density estimation using real nvp," *arXiv preprint arXiv:1605.08803*, 2016. xi, 39, 40

[7] I. Goodfellow, Y. Bengio, A. Courville, and Y. Bengio, *Deep learning*, vol. 1. MIT Press, 2016. 5, 8

[8] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998. 5

[9] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014. 5

[10] V. Nair and G. E. Hinton, "Rectified linear units improve restricted boltzmann machines," in *Proceedings of the 27th international conference on machine learning (ICML-10)*, pp. 807–814, 2010. 5

[11] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016. 5, 8, 18

[12] S. Yun, J. Park, K. Lee, and J. Shin, "Regularizing class-wise predictions via self-knowledge distillation," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 13876–13885, 2020. 6, 15, 17

[13] S. Pan, "Q.: A survey on transfer learning," *IEEE Transactions on Knowledge and Data Engineering*, vol. 22, no. 10, pp. 1345–1359, 2010. 7

[14] J. Yosinski, J. Clune, Y. Bengio, and H. Lipson, "How transferable are features in deep neural networks?," *Advances in neural information processing systems*, vol. 27, 2014. 7

[15] S. Kirkpatrick, C. D. Gelatt Jr, and M. P. Vecchi, "Optimization by simulated annealing," *science*, vol. 220, no. 4598, pp. 671–680, 1983. 7

[16] H. Zhou, A. Vani, H. Larochelle, and A. Courville, "Fortuitous forgetting in connectionist networks," *arXiv preprint arXiv:2202.00155*, 2022. 8, 11, 13, 15, 23, 25, 30, 31

[17] J. Frankle, G. K. Dziugaite, D. M. Roy, and M. Carbin, "Stabilizing the lottery ticket hypothesis," *arXiv preprint arXiv:1903.01611*, 2019. 8, 11

[18] Y. Bengio, P. Lamblin, D. Popovici, and H. Larochelle, "Greedy layer-wise training of deep networks," *Advances in neural information processing systems*, vol. 19, 2006. 8, 18

[19] E. Belilovsky, M. Eickenberg, and E. Oyallon, "Greedy layerwise learning can scale to imagenet," in *International conference on machine learning*, pp. 583–593, PMLR, 2019. 8, 9, 13, 15, 16, 18, 19, 20, 34, 35, 36, 37, 38, 39

[20] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, "Communication-efficient learning of deep networks from decentralized data," in *Artificial intelligence and statistics*, pp. 1273–1282, PMLR, 2017. 9, 18

[21] T. Li, A. K. Sahu, A. Talwalkar, and V. Smith, "Federated learning: Challenges, methods, and future directions," *IEEE signal processing magazine*, vol. 37, no. 3, pp. 50–60, 2020. 9, 18

[22] Q. Li, Z. Wen, Z. Wu, S. Hu, N. Wang, Y. Li, X. Liu, and B. He, "A survey on federated learning systems: Vision, hype and reality for data privacy and protection," *IEEE Transactions on Knowledge and Data Engineering*, vol. 35, no. 4, pp. 3347–3366, 2021. 9, 18

[23] A. Lalitha, S. Shekhar, T. Javidi, and F. Koushanfar, "Fully decentralized federated learning," in *Third workshop on bayesian deep learning (NeurIPS)*, vol. 12, 2018. 9

[24] H. Wang, X. Liu, J. Niu, S. Tang, and J. Shen, "Unlocking the potential of federated learning for deeper models," *arXiv preprint arXiv:2306.02701*, 2023. 9, 10, 18, 19, 38, 41

[25] J. Frankle and M. Carbin, "The lottery ticket hypothesis: Finding sparse, trainable neural networks," *arXiv preprint arXiv:1803.03635*, 2018. 11

[26] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Communications of the ACM*, vol. 60, no. 6, pp. 84–90, 2017. 13

[27] N. Zhao, Z. Wu, R. W. Lau, and S. Lin, "What makes instance discrimination good for transfer learning?," *arXiv preprint arXiv:2006.06606*, 2020. 13

[28] A. Tapp, "A new approach in machine learning," *arXiv preprint arXiv:1409.4044*, 2014. 14, 17, 36

[29] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *nature*, vol. 521, no. 7553, pp. 436–444, 2015. 18

[30] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, pp. 1097–1105, 2012. 18

[31] E. Belilovsky, M. Eickenberg, and E. Oyallon, "Decoupled greedy learning of cnns," in *International Conference on Machine Learning*, pp. 736–745, PMLR, 2020. 18

[32] H. B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, "Communication-efficient learning of deep networks from decentralized data," 2016. 20

[33] Y. Le and X. Yang, "Tiny imagenet visual recognition challenge," *CS 231N*, vol. 7, no. 7, p. 3, 2015. 22

[34] R. Müller, S. Kornblith, and G. E. Hinton, "When does label smoothing help?," *Advances in neural information processing systems*, vol. 32, 2019. 22

[35] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 2818–2826, 2016. 22

[36] I. Loshchilov and F. Hutter, "Sgdr: Stochastic gradient descent with warm restarts," *arXiv preprint arXiv:1608.03983*, 2016. 22

[37] Y. Guo, N. C. Codella, L. Karlinsky, J. V. Codella, J. R. Smith, K. Saenko, T. Rosing, and R. Feris, "A broader study of cross-domain few-shot learning," ECCV, 2020. 22

[38] S. P. Mohanty, D. P. Hughes, and M. Salathé, "Using deep learning for image-based plant disease detection," *Frontiers in plant science*, vol. 7, p. 1419, 2016. 22

[39] P. Helber, B. Bischke, A. Dengel, and D. Borth, "Eurosat: A novel dataset and deep learning benchmark for land use and land cover classification," *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, vol. 12, no. 7, pp. 2217–2226, 2019. 22

[40] N. Codella, V. Rotemberg, P. Tschandl, M. E. Celebi, S. Dusza, D. Gutman, B. Helba, A. Kalloo, K. Liopyris, M. Marchetti, *et al.*, "Skin lesion analysis toward melanoma detection 2018: A challenge hosted by the international skin imaging collaboration (isic)," *arXiv preprint arXiv:1902.03368*, 2019. 22

[41] P. Tschandl, C. Rosendahl, and H. Kittler, "The ham10000 dataset, a large collection of multi-source dermatoscopic images of common pigmented skin lesions," *Scientific data*, vol. 5, no. 1, pp. 1–9, 2018. 22

[42] X. Wang, Y. Peng, L. Lu, Z. Lu, M. Bagheri, and R. M. Summers, "Chestx-ray8: Hospital-scale chest x-ray database and benchmarks on weakly-supervised classification and localization of common thorax diseases," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 2097–2106, 2017. 22

[43] M.-E. Nilsback and A. Zisserman, "Automated flower classification over a large number of classes," in *2008 Sixth Indian Conference on Computer Vision, Graphics & Image Processing*, pp. 722–729, IEEE, 2008. 24, 26

[44] C. Wah, S. Branson, P. Welinder, P. Perona, and S. Belongie, "The Caltech-UCSD Birds-200-2011 Dataset," Tech. Rep. CNS-TR-2011-001, California Institute of Technology, 2011. 24, 26

[45] S. Maji, E. Rahtu, J. Kannala, M. Blaschko, and A. Vedaldi, "Fine-grained visual classification of aircraft," *arXiv preprint arXiv:1306.5151*, 2013. 24, 26

[46] A. Quattoni and A. Torralba, "Recognizing indoor scenes," in *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pp. 413–420, IEEE, 2009. 24, 26

[47] A. Khosla, N. Jayadevaprakash, B. Yao, and F.-F. Li, "Novel dataset for fine-grained image categorization: Stanford dogs," in *Proc. CVPR workshop on fine-grained visual categorization (FGVC)*, vol. 2, Citeseer, 2011. 24

[48] A. Krizhevsky, G. Hinton, *et al.*, "Learning multiple layers of features from tiny images," tech. rep., University of Toronto, 2009. 24, 25

[49] C. P. Phoo and B. Hariharan, "Self-training for few-shot transfer across extreme task differences," 2021. 27, 28

[50] M. Yazdanpanah, A. A. Rahman, M. Chaudhary, C. Desrosiers, M. Havaei, E. Belilovsky, and S. E. Kahou, "Revisiting learnable affines for batch norm in few-shot transfer learning," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 9109–9118, June 2022. 27, 28

[51] C. Finn, P. Abbeel, and S. Levine, "Model-agnostic meta-learning for fast adaptation of deep networks," in *International conference on machine learning*, pp. 1126–1135, PMLR, 2017. 27

[52] N. Park and S. Kim, "How do vision transformers work?," *arXiv preprint arXiv:2202.06709*, 2022. 30

[53] S. Hochreiter and J. Schmidhuber, "Flat minima," *Neural computation*, vol. 9, no. 1, pp. 1–42, 1997. 30

[54] Z. Cai, "Sa-gd: Improved gradient descent learning strategy with simulated annealing," *arXiv preprint arXiv:2107.07558*, 2021. 30

[55] R. Baldock, H. Maennel, and B. Neyshabur, "Deep learning through the lens of example difficulty," *Advances in Neural Information Processing Systems*, vol. 34, pp. 10876–10889, 2021. 30, 31

[56] A. Krizhevsky, V. Nair, and G. Hinton, "Cifar-10 (canadian institute for advanced research)," 34, 38

[57] A. Krizhevsky, V. Nair, and G. Hinton, "Cifar-100 (canadian institute for advanced research)," 34

[58] A. Krizhevsky, "Tiny imagenet challenge," in *Learning Representations Workshop, International Conference on Learning Representations*, 2015. 34

[59] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," 2015. 38