Distillation Techniques for Network Reconstruction in Adaptive Systems

Joost Weerheim
13769758

Bachelor thesis
Credits: 18 EC

Bachelor *Kunstmatige Intelligentie*

University of Amsterdam
Faculty of Science
Science Park 900
1098 XH Amsterdam

Supervisor
Sudaksh Kalra

Informatics Institute
Faculty of Science
University of Amsterdam
Science Park 900
1098 XH Amsterdam

Semester 2, 2024-2025

Abstract

In an adaptive convolutional neural network, there is a need for partial weight freezing, which can lead to overfitting during the reconstruction stages of the network. This thesis aims to investigate the effects of layer-wise distillation on the reconstruction performance of adaptive convolutional neural networks in image classification tasks. To achieve this, we implemented three training methods for our custom adaptive convolutional neural network. Namely, knowledge distillation, cross-entropy, and layer-wise distillation. We analyzed all three training methods on adaptive ResNet-18 and ResNet-56 architectures and found that these distillation techniques do not improve the performance further when compared to cross-entropy loss training. We also implemented layer-wise distillation on a ball classifier network which can be used for robot soccer. This showcased the validity of layer-wise distillation as a training method tasked with image classification tasks using non-adaptive convolutional neural networks.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Modern computer vision is largely driven by deep neural networks (DNN), more specifically convolutional neural networks (CNN) [4][5]. These network architectures have become integral in achieving state-of-the-art performance on a wide range of computer vision tasks ranging from image classification to robotics [6][7][8]. However, the deployment of CNNs is limited by their computational complexity and memory footprint. These constraints are especially prevalent in edge devices like self-driving cars, robotics, and other EdgeAI systems. Here, on-board compute and storage resources can be limited and fluctuating, preventing these devices from performing on-board data processing. While significant research has focused on reducing the computational complexity of CNNs [9][10][11][12], a research gap persists in developing solutions that achieve an optimal balance between performance and resource efficiency, especially for EdgeAI applications where these constraints are critical. Adaptive neural networks represent a possible solution to these challenges.

Adaptive neural networks are especially well-suited for environments with limited and fluctuating resources. They offer network elasticity, dynamically adapting their architecture based on available system resources while balancing performance and computational efficiency. This thesis only considers adaptive CNNs for classification tasks. In our approach, the CNN is first compressed using structured pruning and then reconstructed using partial weight freezing and retraining.

Challenges and Solutions: A current limitation of this approach is the saturating performance of adaptive CNNs during the reconstruction phase, see Section 3.2. This problem emerges from the utilization of partial weight freezing, and in order to address this challenge, we aim to investigate whether layer-wise distillation (LWD) or an increased architecture size could provide a solution.

1.0.1 Research Question

The overarching research question that this thesis aims to address is:
What effects do LWD and architecture size have on the reconstruction performance of adaptive CNNs in image classification tasks.
The motivation behind researching LWD specifically is to explore the benefits of LWD in the reconstruction phase of adaptive CNNs and whether it could help with alleviating the saturation problem in network performance, which is an unexplored research area.

Chapter 2

Background and Related Research

This chapter provides the background for CNNs, adaptive architectures, and some performance metrics. It briefly reviews pruning, knowledge distillation, LWD in LLMs, and early exit networks.

2.1 Background

2.1.1 Convolutional Neural Networks

In the context of computer vision, a CNN is a specialized type of neural network that is able to learn the characteristics of a dataset called features through a process of sliding a filter across an image or previous feature map. This sliding process across an input image or feature map produces a feature map that can contain important information about what is present in the input image. The accuracy of a CNN can be improved by increasing the depth of the network [4], having a singular input image that generates many consecutive feature maps that all work together to extract features at different levels of abstraction. Throughout many iterations of this process, the filters get optimized so that they become better at recognizing features. In the process of improving network performance, CNNs have required an increasingly large amount of computational resources, associated with the increased depth of the network. This is often a bottleneck to deploy CNNs in environments with constrained and fluctuating computational resources.

2.1.2 Adaptive and Early-Exit Architectures

Adaptive Architectures

In order to increase the feasibility of DNN deployment in environments with constrained and fluctuating computational resources, the concept of network adaptivity has been proposed [13][14][15][16]. Adaptive networks are able to dynamically adjust their architecture based on the availability of resources. This allows systems with limited and variable resources to utilize full network capacity when resources are ample and scale down the network capacity when resources become more scarce.

Early-exit Architectures

Early-exit architectures are used for similar purposes as adaptive networks by providing an alternative way to deploy networks on systems with constrained and fluctuating computational resources. These networks have multiple exits throughout the architecture which allow the network to exit early during inference if there is not enough compute available to complete a full pass through the network [17][18].

2.1.3 Performance Metrics

This section describes the two main performance metrics which are used throughout the experiments to test the network performance during reconstruction.

F1-Score

F1-score is a performance metric for classification problems. It combines precision and recall into a single score [19].

$$\begin{aligned} \text{F1-score} &= \frac{2 \cdot \text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} \\ \text{Precision} &= \frac{\text{TP}}{\text{TP} + \text{FP}} \\ \text{Recall} &= \frac{\text{TP}}{\text{TP} + \text{FN}} \end{aligned} \tag{2.1}$$

Where TP refers to true positive, FP to false positive, and FN to false negative.

Accuracy

Accuracy is a straightforward measure that indicates the ratio of correct predictions over all predictions.

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}} \tag{2.2}$$

2.2 Related Research

2.2.1 Pruning

Modern DNNs such as large language models (LLM) or CNNs require a significant amount of computational and storage resources. Furthermore, it has become increasingly interesting to deploy these capabilities on edge devices. For example, many self-driving cars make extensive use of computer vision to detect their surroundings, and human-oriented robots utilize LLMs to naturally communicate with their human counterparts. These machines often have limited hardware that is not suitable for running large AI networks. Especially since the introduction of the transformer [20], there has been a significant increase in pruning research, as can be seen in Figure 2.1. This is a clear indication of people requiring compute heavy AI capabilities in computationally constrained environments.

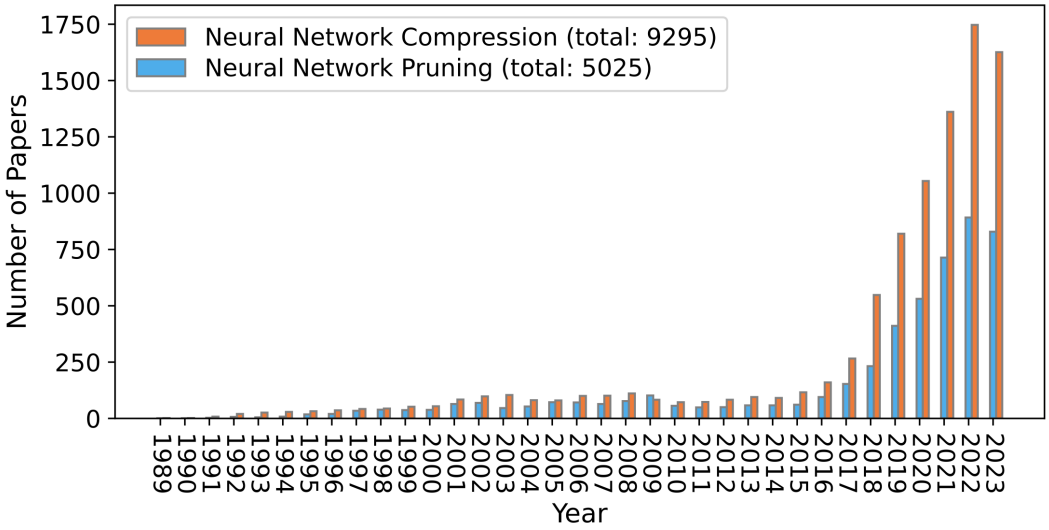


Figure 2.1: Research papers published over time on pruning and network compression. Source: [1]

There are many pruning variants, and the choice of pruning method is use-case dependent. The core concept of pruning is to reduce the complexity of a

network by removing the components which contribute the least to the network performance while minimizing the performance penalty. Pruning techniques can be broadly categorized into unstructured pruning and structured pruning.

Unstructured Pruning

Unstructured pruning turns a dense neural network into a sparse one, see Figure 2.2. This is achieved by removing individual weights based on optimizing the trade-off between network complexity and performance [21].

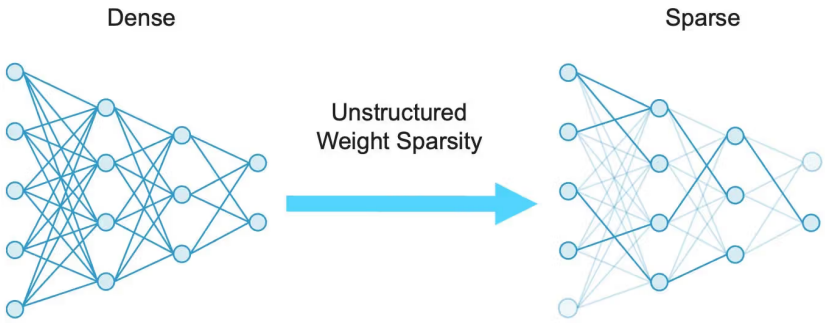


Figure 2.2: Unstructured pruning turns a network from a dense into a sparse one

Definition 1 [1] Given a set of weights $W = w_0, w_1, ..., w_K$, a dataset containing input x and output y pairs: $D = (x_i, y_i)_{i=1}^{N}$, and the amount of weights which should be left unpruned: k with $k < K$. Unstructured pruning can be mathematically described as the following optimization problem [22]:

$$\min_{W} \mathcal{L}(W; D) = \min_{W} \frac{1}{N} \sum_{i=1}^{N} \ell(W; (x_i, y_i)) \quad s.t. ||W||_0 \leq k \tag{2.3}$$

Here, $\ell(\cdot)$ is the standard loss function (e.g., cross-entropy loss), w is the set of parameters of the neural network, and $|| \cdot ||_0$ is the standard L_0 norm.
Given the unstructured pruning of weights, the resulting sparse tensors maintain the identical spatial representations of their unpruned variant. Hence, it is not a viable option for improving the inference time on systems that utilize GPUs that are optimized for spatial regularities in the weight tensors and thus require a structured reduction of the weight tensors while maintaining the same spatial regularities to achieve faster inference time.

Structured Pruning

Structured pruning has a more coarse-grained granularity than unstructured pruning, but it is able to achieve network speedups in addition to reducing network size. It does this through removing filters [23], channels [24], transformer attention heads [25], or even layers [26]. Structured pruning can be defined similarly to Definition 2.1, although it now operates on filters or channels in comparison to individual weights. Our framework for creating adaptive CNNs exclusively uses structured channel pruning. Channel pruning reduces the width of the feature maps making it efficient on GPU since spatial regularities are maintained [27].

2.2.2 Knowledge Distillation

Knowledge distillation (KD) is a technique that transfers knowledge from a large and computationally complex network to a smaller one. This enables deploying smaller AI networks on less powerful hardware while maintaining acceptable network performance [11].

The most common variant of KD utilizes the logits from the teacher as transferable knowledge. In order for the student network to learn from the teacher network, the difference between the smoothed logits from the teacher and the logits from the student is introduced as a distillation loss, which is added to the total loss. The teacher network usually produces outputs where the correct class is assigned an output probability close to 1 while all other classes are assigned an output probability close to 0. This is not very informative for the student network since it can benefit from information about which classes were also likely. In order to transfer this more nuanced information, a temperature parameter is used [11]. T is the temperature parameter with a default value of 1. Using a higher value for T produces softer target distributions, which are more informative for the student network as they reveal the relative probabilities of incorrect classes instead of just the correct one.

$$q_i = \frac{\exp(z_i/T)}{\sum_j \exp(z_j/T)} \tag{2.4}$$

where z_i denotes the logits and q_i denotes the class probabilities.

Hint-based KD

An alternative approach to distillation is hint-based KD. The core concept with this approach is to not only use the outputs of the teacher but also the intermediate representations of the activations from hidden layers, learned by the teacher as hints for the student. This additional information can lead to improved performance of

the student [28]. The loss function is thus extended by a mean squared error loss between the intermediate representation of the student and teacher networks. It is possible to construct a loss function which incorporates both distillation loss and hint-based intermediate representation loss, or alternatively, a loss function which only uses the intermediate representation loss in addition to the regular network loss, as can be seen in Figure 2.3.

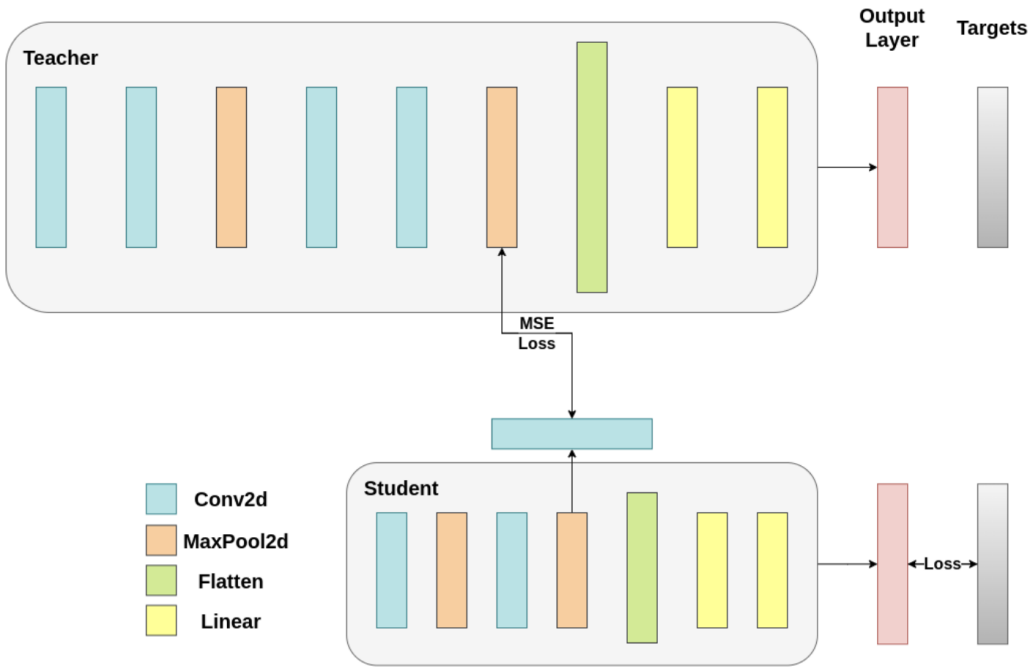


Figure 2.3: Hint-based KD. Source: [2]

2.2.3 Layer-wise Distillation

LWD is similar to hint-based KD; there is a student and a teacher network, and the student distills knowledge from the teacher by mimicking the hidden representations of the teacher at every intermediate layer. LWD has successfully been used to compress LLMs [29].

Chapter 3

Method

3.1 A Framework for Adaptive CNNs

This thesis utilizes a custom framework for classification that is able to generate adaptive CNNs known as descendant networks, as illustrated in Figure 3.1. This is done through a process of compressing a pretrained network down into the core network and then reconstructing the network iteratively by reintroducing the previously pruned channels and retraining the weights of those channels. Compressing the network down is done through structured pruning, and reconstructing is done through a two-step process of partial weight freezing and retraining the unfrozen weights, as described in Section 3.1.1. This retraining step can utilize losses like cross-entropy or knowledge distillation but suffers from overfitting or performance saturation. This is caused by partial weight freezing, this is described in more detail in Sections 3.2 and 5.3.3.

The pruning criterion used is L1-Norm, we decided to use this technique since it is computationally efficient while providing competitive performance in comparison with other techniques [30]. The L1-Norm will be computed over the channels of each layer to estimate their importance. The channels with the least importance will be removed. This process happens iteratively and after each step, metadata will be saved as a dictionary, containing information about what has been pruned along with the associated indexing information. The smallest descendant network, or core network, is what remains after the pruning stage. This means that if the network is pruned in 3 steps with a 20% prune ratio each step, the resulting core network has a size of: original size $* 0.2^3$.

The reconstruction stage is an iterative process comprised of fine-tuning and partial weight freezing stages. This process is illustrated by Figure 3.2. The reconstruction phase initializes with the retraining of the core network. After that, the next descendant network is initialized with the weights from the core

network and the channels that were pruned in the compression stage pertaining to this level. The weights of the retrained core network are then frozen, and the reintroduced filters are retrained. This process continues until the pruned network is reconstructed in an identical amount of steps as the pruning stages, reconstructing the pruned network to exactly the same size as the original network.

The pruning and reconstruction process is illustrated by Figure 3.1.

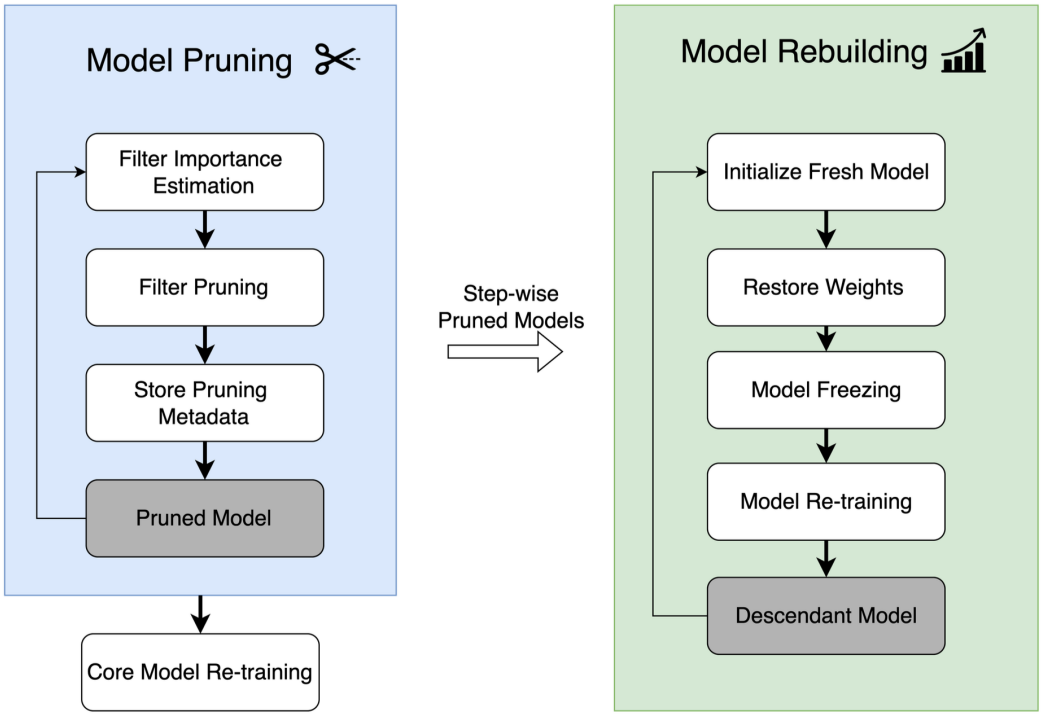


Figure 3.1: Iterative pruning and reconstruction. Source: [3]

3.1.1 Partial Weight Freezing

As stated above, reconstruction is a two-step process. After the compression stage, the last descendant network is called the core network, and the size of the core network is based on the desired prune ratio and network architecture specifications. The first step in the reconstruction process is the retraining of the core network, which can be done with any loss function (e.g. cross-entropy loss, knowledge distillation, or LWD). Retraining the core network is the only reconstruction stage that does not utilize partial weight freezing. The core network is the lowest level descendant network and every descendant network after that is constructed using two components:

- The weights that were retained and retrained in the previous descendant network level W_{L-1}.
- The additional weights W_l that were pruned in iteration L that are now being reintroduced into the descendant network.

So the previously trained weights W_{L-1} are used directly in the new network but kept frozen, and only the new weights W_l are being updated during training, as illustrated by Figure 3.2. The partial weight freezing is applied through gradient masking. Specifically, a binary mask G is used to zero out the gradients of the shared weights during backpropagation:

$$W_L \leftarrow W_L - \eta(G \odot \nabla \mathcal{L}) \tag{3.1}$$

where $G_{ij} = 0$ for all entries belonging to W_{L-1} (shared weights), and $G_{ij} = 1$ otherwise. This ensures that only the reintroduced weights are trainable, preserving compatibility with lower level descendant networks.

3.2

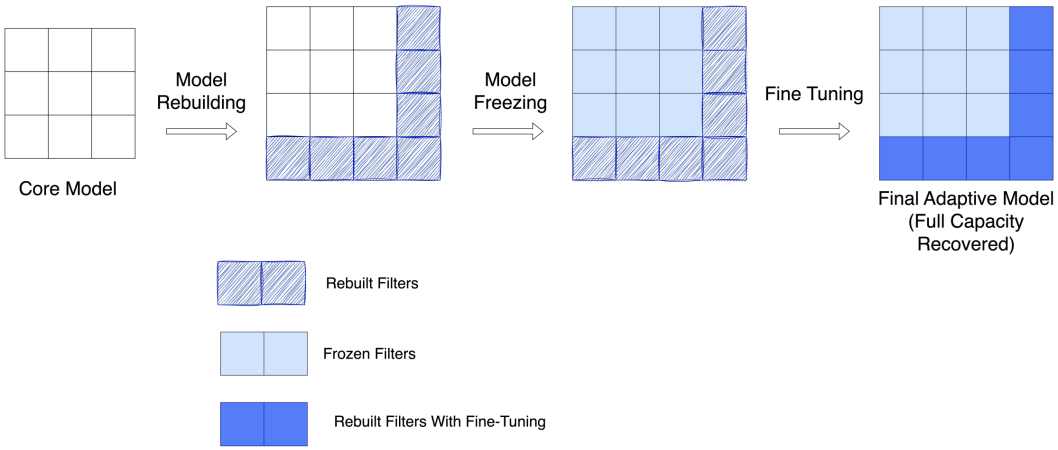


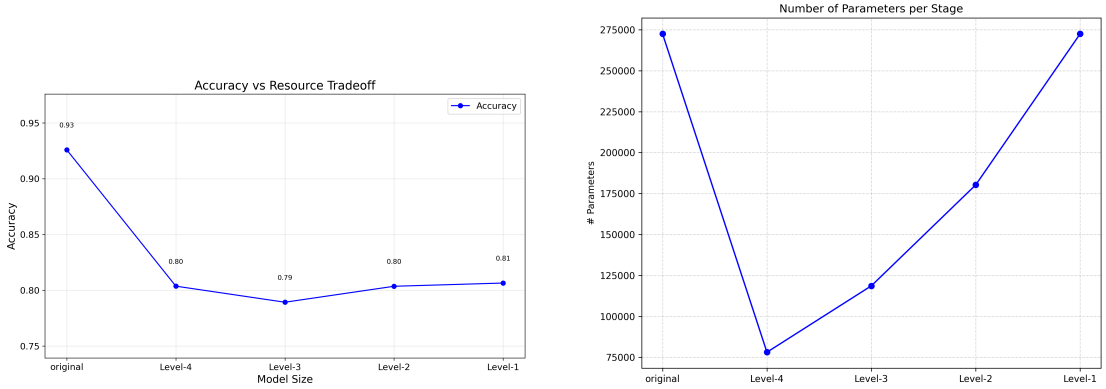
Figure 3.2: Network partial weight freezing and rebuilding illustration. Source: [3]

This mechanism is the backbone of adaptive neural networks, and it allows for multiple nested descendant networks to coexist. When weights get retrained in a descendant network that utilizes partial weight freezing, the accuracy of the network saturates or starts showing overfitting, as discussed in Section 3.2. We could attribute this to the partial weight freezing.

3.2 The Saturation Problem

A major challenge that is introduced as a direct consequence of partial weight freezing is the saturation or overfitting of network performance during reconstruction.

When using the regular cross-entropy loss function on our adaptive ResNet-18 with partial weight freezing, we observe saturating performance, this is illustrated along with the associated network parameters at each stage in Figure 3.3.



(a) Adaptive ResNet-18 accuracy (b) Adaptive ResNet-18 parameter count

Figure 3.3: Network parameters and associated network performance during network reconstruction.

For all three rebuilding stages, the ratio of frozen to non-frozen parameters increases. Consequently, the severeness of the saturation increases for every step, as indicated by the growing discrepancy between training and validation accuracy in Figure 3.4. The core network is intentionally left out since there is no weight freezing implemented at this level.

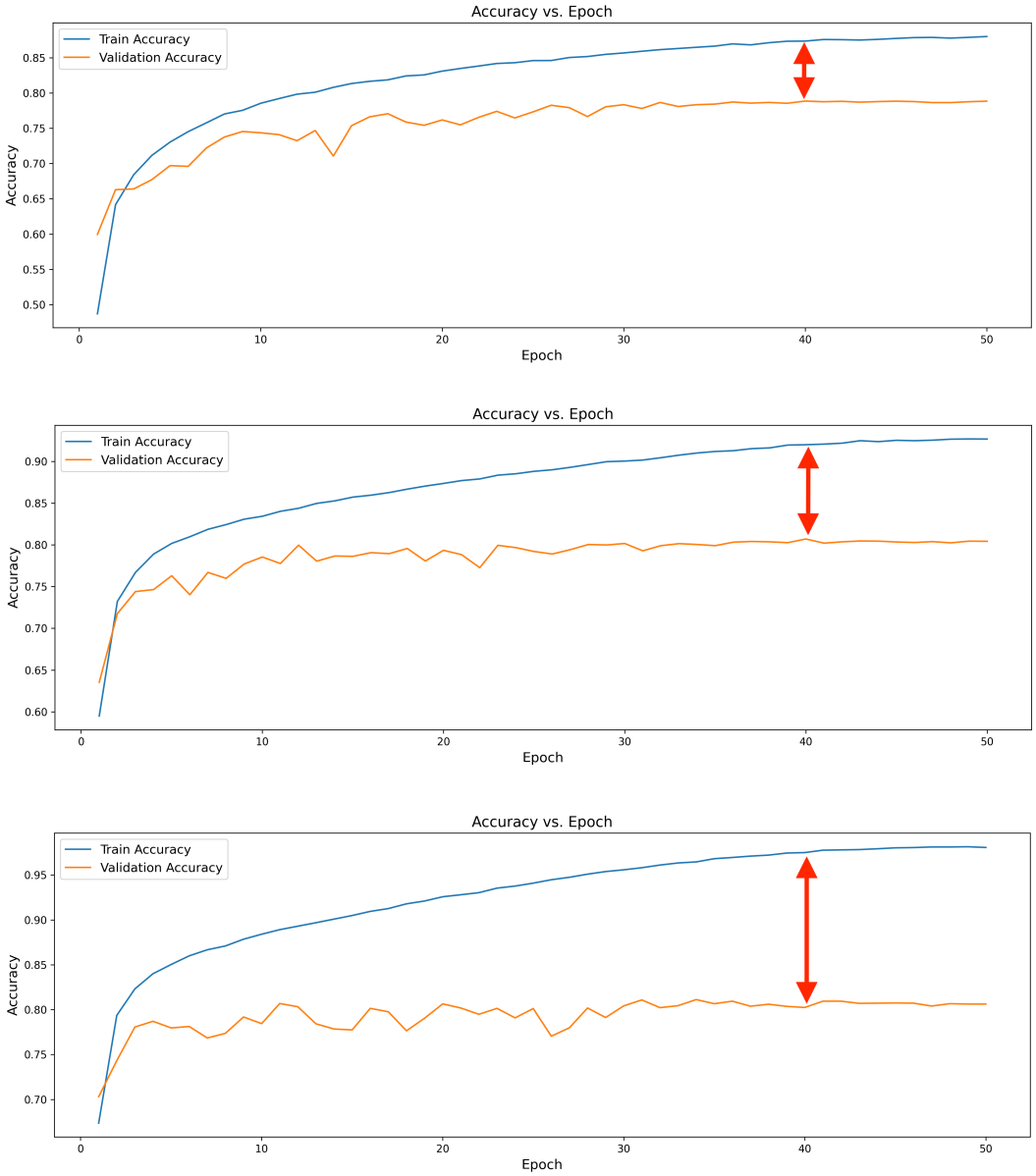


Figure 3.4: Saturating network performance in the third, second, and first reconstruction stages.

Adaptive ResNet-18 utilizing cross-entropy loss has not overcome the saturation problem, as indicated by the red arrows in Figure 3.4. This thesis aims to investigate the effects of LWD and architecture size on the reconstruction performance. We specifically want to explore if LWD in ResNet-18 and ResNet-56 can overcome the saturation problem.

3.3 Layer-wise Distillation Implementation

In order to verify whether LWD can provide a solution to the saturation problem, we implemented it for the custom framework outlined above. The teacher network is initialized with a set of pretrained weights. Furthermore, the network has additional intermediate output layers which output the feature maps of several intermediate convolutional layers. These outputs get aligned manually with the intermediate output layers of the student network. These intermediate outputs are used to calculate the layer-wise loss, which is a summation of the mean squared error between all intermediate student and teacher outputs. The loss function is a combination of cross-entropy loss, distillation loss, and layer-wise loss:

$$\begin{aligned} \ell = & \alpha \cdot \text{classification\_loss} + \\ & (1-\alpha) \cdot (\omega \cdot \text{distillation\_loss} + (1-\omega) \cdot \text{layer\_loss}) \end{aligned} \tag{3.2}$$

Implementing LWD requires some considerations. In order for the student to be able to learn from the teacher, there needs to be a degree of alignment between the teacher and student architecture such that the intermediate outputs are meaningful. We only use pretrained teacher networks with the same architecture as the student. Other architectures would have different features altogether.

If the networks have a meaningful architecture with respect to each other, then the LWD can be implemented by providing a list of intermediate teacher channels and student channels. These channels will be used to indicate where the intermediate layers are located within the network. The output feature maps of the student and teacher networks are then aligned by a transformation layer which takes the student channels as input and teacher channels as output and computes a convolution with kernel size 1 to align the tensor dimensions of teacher and student feature maps. These aligned dimensions are necessary since the layer-wise loss is constructed as a sum of mean squared error losses between the respective teacher and student intermediate output feature maps.

LWD is illustrated in Figure 3.5.

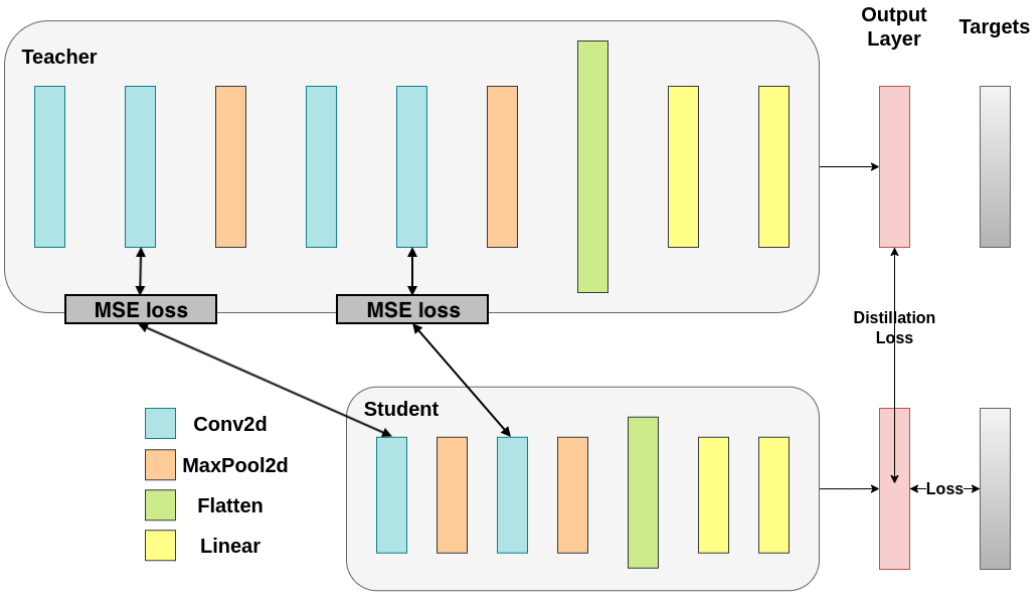


Figure 3.5: LWD

Chapter 4

Experimental Setup

4.1 Datasets and Preprocessing

4.1.1 Datasets

CIFAR-10
The CIFAR-10 dataset was used for the experiments described in Sections 5.1, 5.2, and 5.3. The CIFAR-10 dataset contains 10 classes, each with 6000 RGB images. The total dataset thus contains 60000 32x32 images, of which 10000 are part of the test set and 50000 are part of the train set [31]. CIFAR-10 was used for initial experimentation and the hyperparameter search since it is a widely recognized benchmark dataset and computationally efficient due to the small image size. CIFAR-10 was chosen over more complex benchmark datasets like ImageNet for the experiments in Section 5.3. The reasoning behind this was that solving the saturation problem is fundamentally different from benchmarking the network on a state-of-the-art dataset, and it would speed up our experiment significantly to utilize CIFAR-10.
The Standard Platform League Ball Dataset
The Standard Platform League (SPL) Ball Dataset is a dataset containing more than 100,000 48x48 grayscale images of Robocup SPL scenes, with or without a ball in them. This data can be used to train a binary ball classifier. This classifier can then be deployed on the NAOv6 robots to assist them in playing autonomous robot soccer. This dataset is used in Section 5.4.

4.1.2 Preprocessing

Data Augmentation
In order to reduce overfitting, data augmentation was applied for CIFAR-10. This

creates a more diverse set of data, which helps the network generalize better.

- Horizontal flip: 50% probability,
- Vertical flip: 20% probability,
- Affine transformations: 50% probability to apply a random amount of shear, rotation, scale, or translation,
- Crop: 50% probability to crop some of the images by 0-10% of their height/width.

Normalization
The pretrained networks used in our ResNet experiments use these normalization parameters:

- Mean: [0.4914, 0.4822, 0.4465]
- Standard deviation: [0.2023, 0.1994, 0.2010]

To ensure that the input images are similar to what the pretrained networks were trained on, the same normalization parameters are applied.

4.2 Networks and Architectures

4.2.1 ResNet

In order to facilitate training deeper neural networks, the ResNet architecture was created. It introduced residual blocks that utilize skip connections in order to overcome the vanishing gradient problem. This method argues that it is easier for a network to learn the residual instead of the entire representations from scratch [4].

We used ResNet-18 since its more compact network size is ideal for smaller datasets like CIFAR-10. We used the pretrained PyTorch network from [32]. This architecture was used in Sections 5.1, 5.2, and 5.3.

To research whether a deeper architecture could alleviate the saturation problem, ResNet-56 was utilized in Section 5.3. ResNet-56 uses the same concepts as ResNet-18 but with more layers.

4.2.2 Ball Classifier

The binary ball classifier can be deployed on the NaoV6 robots in order to participate in SPL robot soccer. We created both a student and teacher network, and their architectures can be found in Appendix A: Figures 1 and 2 respectively.

4.3 Hardware and Software Setup

The hyperparameter search was conducted on the monsterfish workstation, using a NVIDIA Titan RTX. This experiment was conducted on monsterfish since the time limit for jobs on the Snellius supercompute cluster was not enough to run the program. The remainder of the experiments was run on the Snellius supercomputer provided by SURF. All experiments that were run on Snellius used the A100 GPUs. In order to construct and train our neural networks, we used the PyTorch deep learning library [33].

4.4 Training and Fine-Tuning Setup

The reconstruction of the networks was done using three methods: cross-entropy, KD, and LWD. We used the Adam optimizer [34] in combination with a cosine annealing learning rate scheduler [35]. For the ResNet-18 experiments, a learning rate of 1e-3 was used. However, this learning rate showed significant overfitting in the ResNet-56 experiments, so we used a learning rate of 1e-4 there to reduce the overfitting.

4.5 Evaluation Metrics

The primary performance metrics used to track and showcase the saturation problem are F1-score and accuracy. Furthermore, the parameters, network size, and MACs were used to showcase the computational resources that were used at various reconstruction stages.

Chapter 5

Experiments and Results

5.1 Experiment 1: Initial Comparison

After implementing LWD, an initial experiment was conducted that set out to explore how the core network after a one-shot prune of 20% would perform for the various types of loss functions: cross-entropy, KD, and LWD. The results can be seen in Figure 5.1.

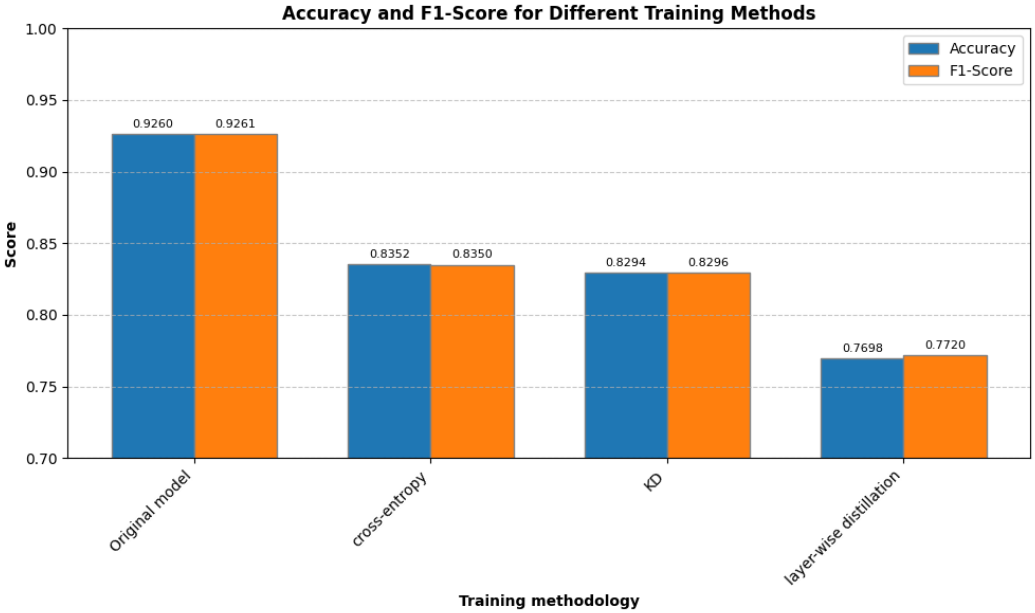


Figure 5.1: Accuracy and F1 score of a 20% pruned network for various types of training methods.

Both the F1-score and accuracy decreased for each method that increases the amount of transferrable information. This was opposite to our initial expectation, which was that the accuracy and F1-score would improve when increasing the amount of transferrable information to the student. In order to explain this result, we hypothesized that due to the increase in transferrable information, the network might need more epochs and optimized hyperparameters in order to utilize the increase in distilled information. As a result, a hyperparameter search was conducted.

5.2 Experiment 2: Hyperparameter Search

The hyperparameter search needed to be computationally feasible, and hence only the alpha and omega parameters, as described in equation 3.1, were considered for the LWD hyperparameters on ResNet-18. Each experiment was run for 100 epochs. The results are shown in Figure 5.2 and 5.3. These heatmaps show the average of the performance over all stages and which parameter combination performs optimally.

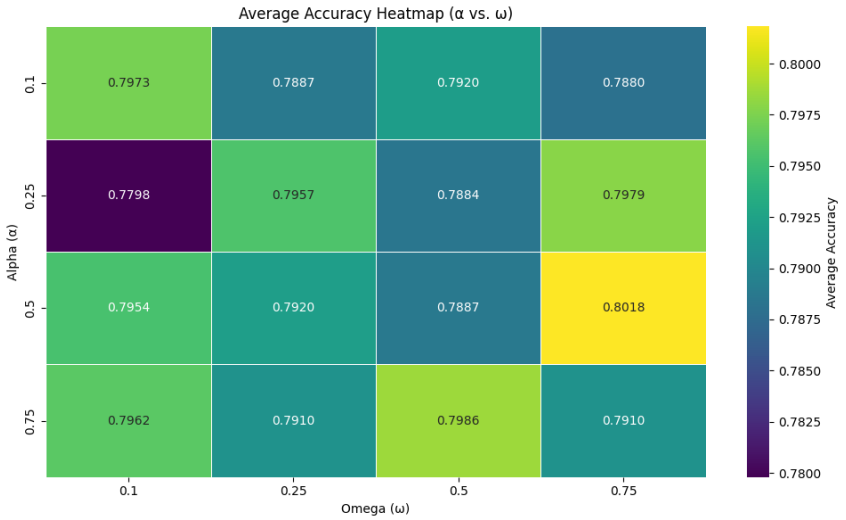


Figure 5.2: Accuracy heatmap for LWD.

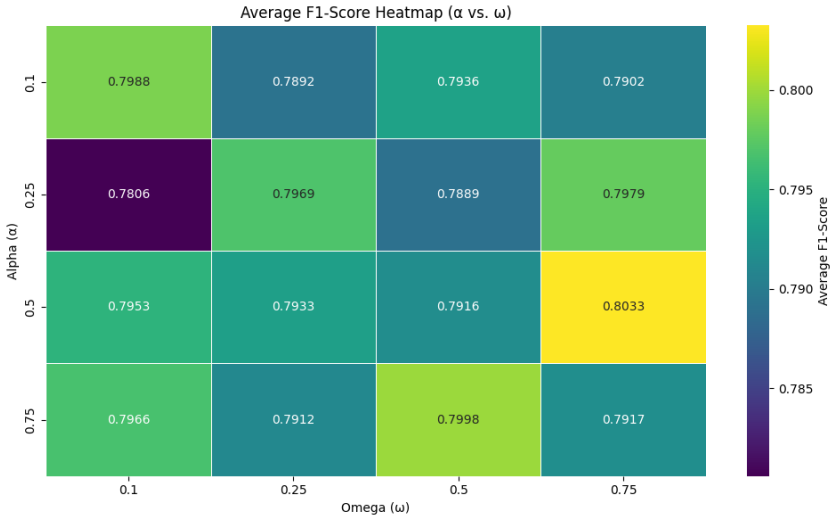


Figure 5.3: F1 score heatmap for LWD

From these figures, we can conclude that $\alpha = 0.5$ and $\omega = 0.75$ are the optimal hyperparameters, and these will be used for the LWD experiments described in experiments 3 and 4. What these values mean intuitively is that the LWD loss function is comprised of 50% cross-entropy loss, 37.5% distillation loss, and 12.5% layer-wise loss.

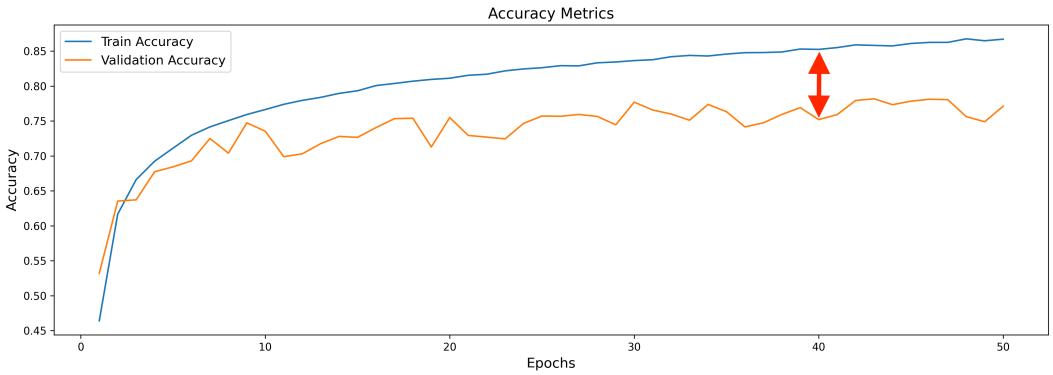
5.3 Experiment 3: CIFAR-10 on Adaptive ResNets

This experiment was conducted to research the effects of LWD and architecture size on the reconstruction performance of adaptive ResNet-18 and ResNet-56 architectures. The aim of this experiment is twofold. First, we would like to know if LWD is able to better mitigate the effects of saturation due to partial weight freezing in comparison to KD and cross-entropy. Second, we are interested in the effects of increasing architecture size with respect to the saturation problem. For each architecture, three experiments will be conducted: one using regular cross-entropy, one using KD, and one using LWD.

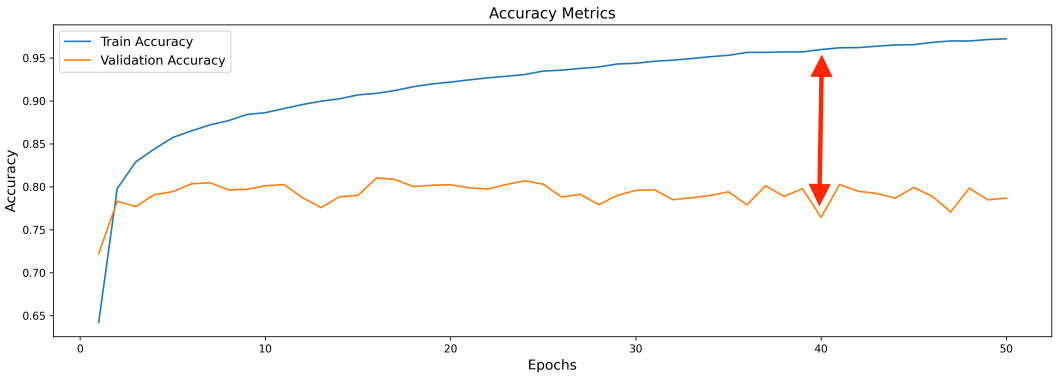
Each experiment has four stages: core, stage 3, stage 2, and stage 1. Only stage 3, 2, and 1 utilize partial weight freezing during reconstruction because the core network has no frozen parameters during retraining. Hence, only stage 3, 2, and 1 are considered when we discuss the partial weight freezing problem. In order to present the results in a more concise manner, we left the figures for stage 2 out.

5.3.1 ResNet-18

We are interested in exploring whether LWD can overcome the saturation problem. The saturation problem for KD and cross-entropy can be seen in Figure 5.5 and Figure 5.6 respectively.



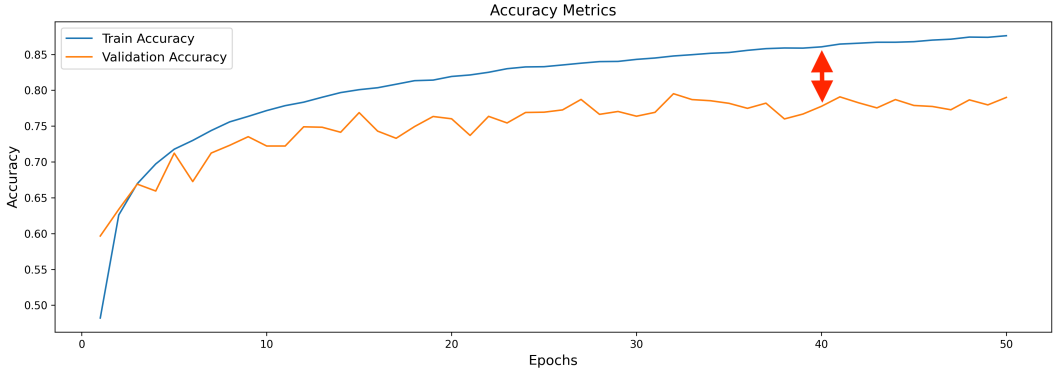
(a) Reconstruction stage 3 network performance



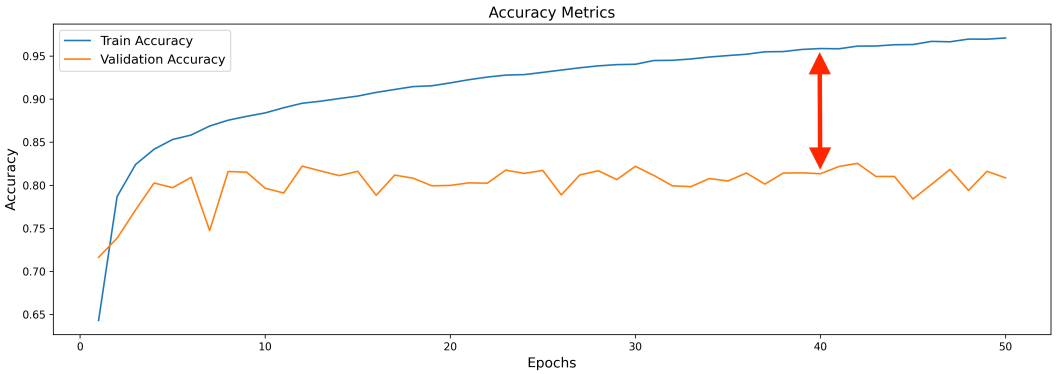
(b) Reconstruction stage 1 network performance

Figure 5.4: Reconstruction performance - LWD - ResNet-18

From Figure 5.4 it can be observed that there is a similar amount of discrepancy between the validation and training accuracy in the stage 1 network performance as in Figure 5.5 and Figure 5.6. Furthermore, all methods seem to be experiencing saturation at an accuracy of around 0.8. Here, partial weight freezing prevents the validation accuracy from exceeding this limit. This indicates that LWD is not able to overcome the saturation problem. This is discussed in more detail in Section 5.3.3.

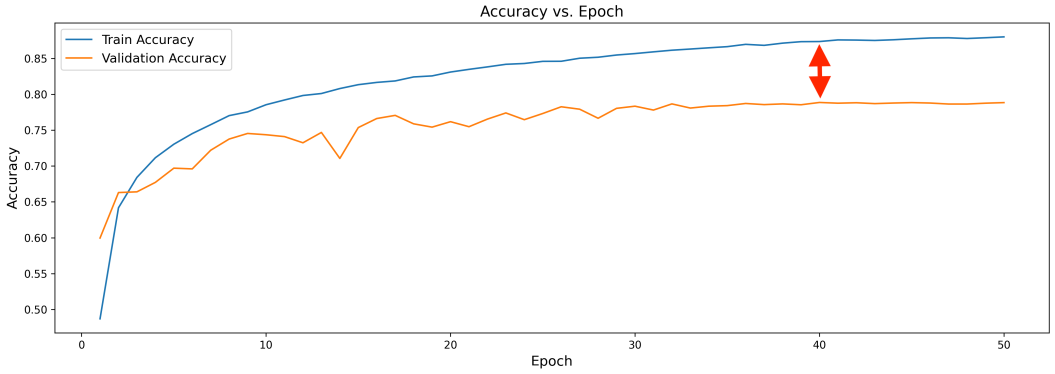


(a) Reconstruction stage 3 network performance

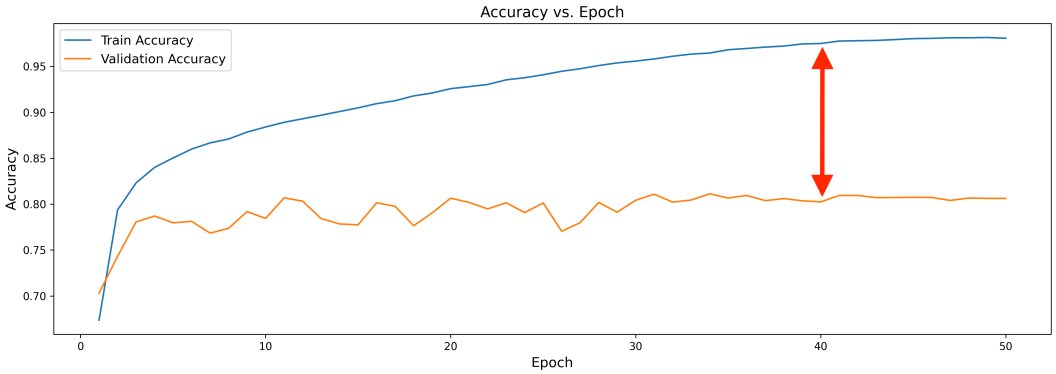


(b) Reconstruction stage 1 network performance

Figure 5.5: Reconstruction performance - Knowledge distillation - ResNet-18



(a) Reconstruction stage 3 network performance



(b) Reconstruction stage 1 network performance

Figure 5.6: Reconstruction performance - Cross-entropy - ResNet-18

Figure 5.7 shows the accuracy for KD, cross-entropy, and LWD methods for ResNet-18. Layer-wise seems to be performing slightly worse or similar compared to the other two methods. In the core network, this might be due to the smallest network not being able to incorporate the teacher's complexity or the network being over-constrained by having to learn a lot of information with limited parameters. In the ResNet-56 experiment, the methods all had equal performance on the core network, as illustrated by Figure 5.11.

Except for the architecture size, there was no difference in implementation, further hinting that the cause of KD and layer-wise under-performing in the core network of ResNet-18 might be due to the smaller network size of 75.000 parameters instead of the 250.000 parameter core network of ResNet-56, as illustrated by Figures 3 and 4 in Appendix A. Furthermore, LWD is being outperformed at stages 3, 2, and 1 as well, and this could be attributed to the partial weight freezing. This is discussed in more detail in Section 5.3.3.

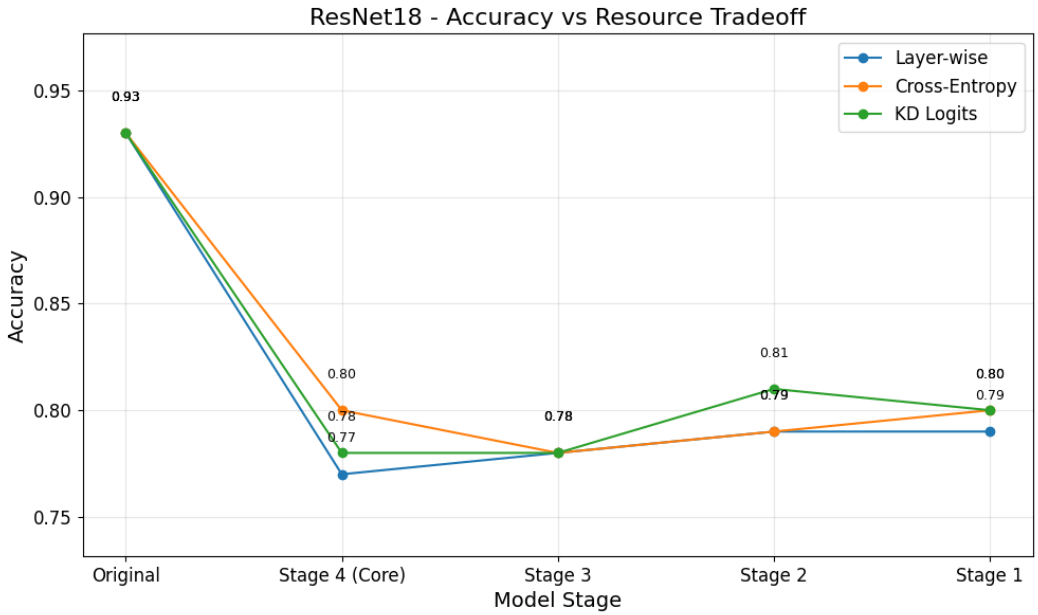
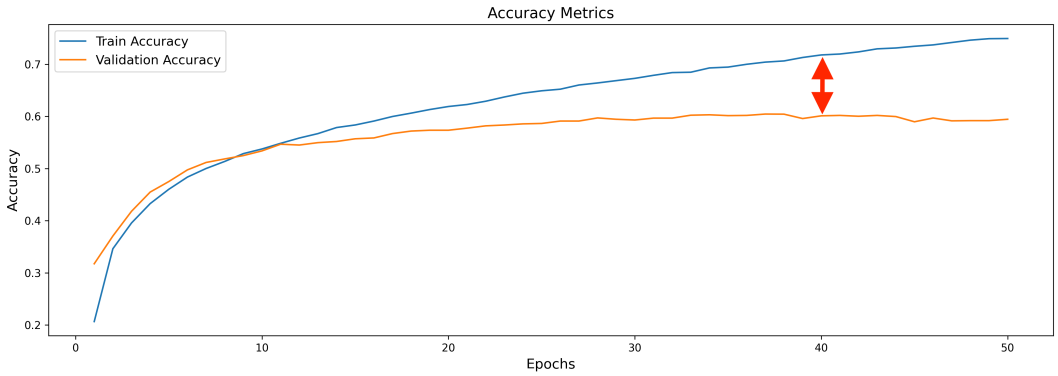


Figure 5.7: ResNet-18 accuracy across methods

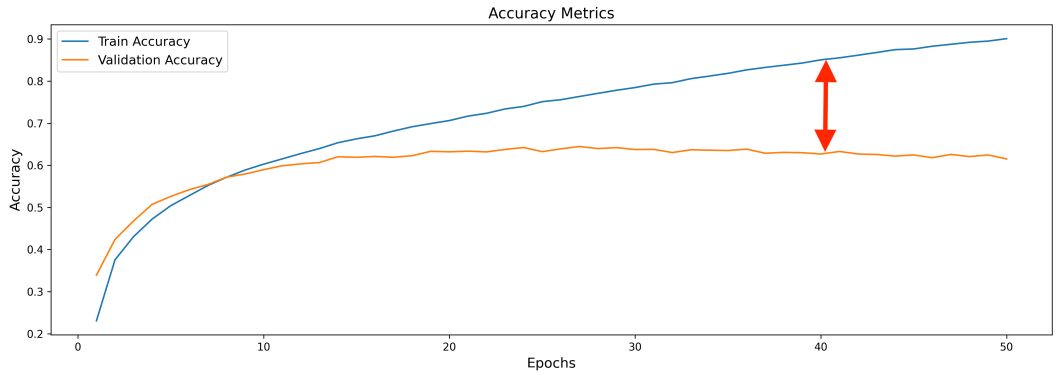
5.3.2 ResNet-56

The aim of incorporating ResNet-56 in the experiments is to explore whether a larger architecture in itself can overcome the saturation problem due to partial weight freezing. If a significant change in saturation occurs, it indicates that a larger network size might inherently be able to solve the saturation problem.

Figures 5.8, 5.9, and 5.10, show the saturation problem is still prevalent. This suggests that the partial weight freezing mechanism is causing a saturation in performance similar to ResNet-18. The saturation does happen at a different accuracy. Where the ResNet-18 saturation was capping validation accuracy at 0.8, ResNet-56 seems to be capping between 0.6 and 0.65. Through experimentation, we observed that ResNet-56 will also cap at 0.8 if the learning rate is the same as the one used for ResNet-18, although this also led to significant overfitting that was not due to partial weight freezing, obscuring what impact partial weight freezing had in comparison to regular overfitting. This was the reason why we decided to use a lower learning rate for ResNet-56, as described in Section 4.4. The result is that the performance of ResNet-56 is significantly lower than ResNet-18, but this is not very relevant for our investigation of the partial weight freezing problem.

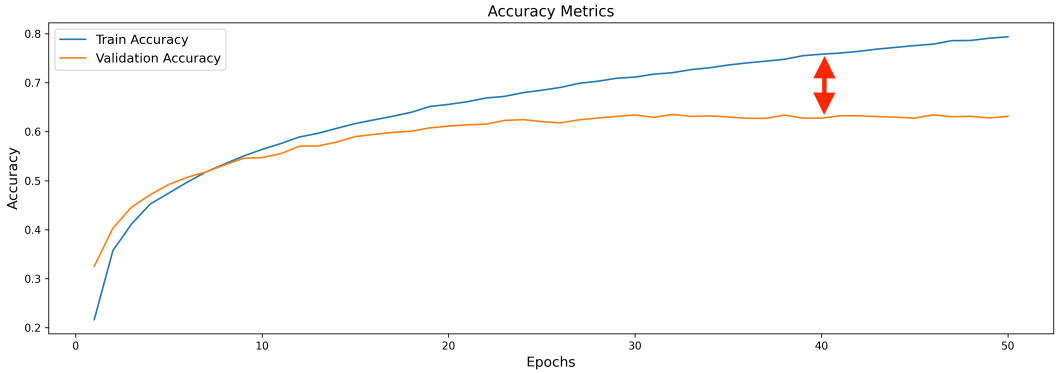


(a) Reconstruction stage 3 network performance

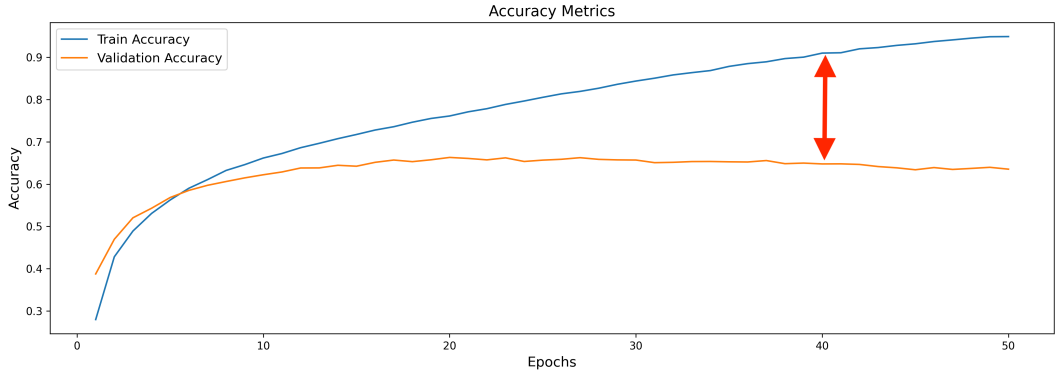


(b) Reconstruction stage 1 network performance

Figure 5.8: Reconstruction performance - Layer-wise - ResNet-56

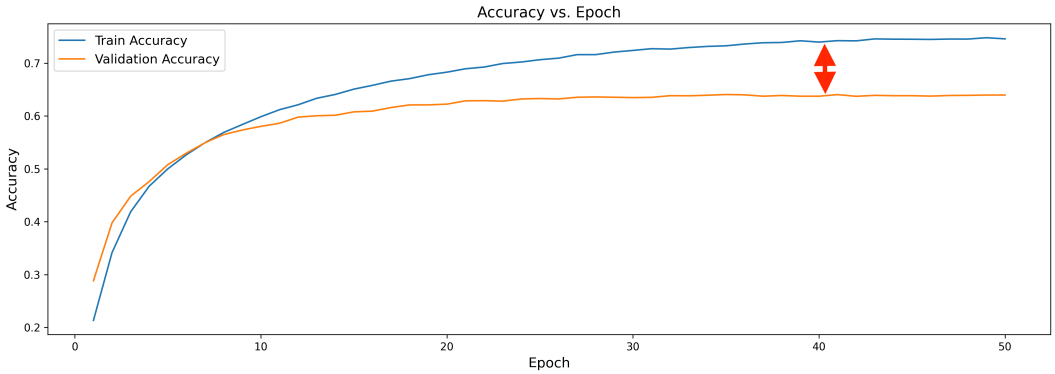


(a) Reconstruction stage 3 network performance

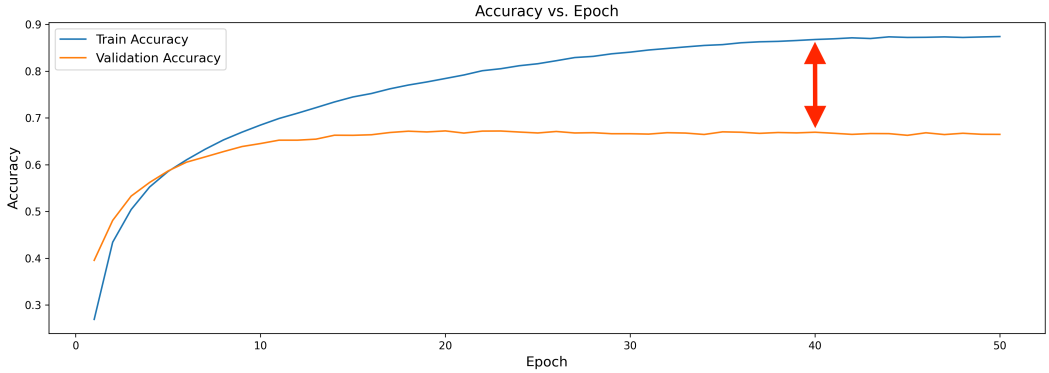


(b) Reconstruction stage 1 network performance

Figure 5.9: Reconstruction performance - Knowledge distillation - ResNet-56



(a) Reconstruction stage 3 network performance



(b) Reconstruction stage 1 network performance

Figure 5.10: Reconstruction performance - Cross-entropy - ResNet-56

Figure 5.11 shows the accuracy for KD, cross-entropy, and LWD methods for ResNet-56. The figure does not differ significantly from Figure 5.7 which is expected since increasing the network size has proven ineffective in solving the saturation problem. It is noteworthy that the accuracy of the core network is equal for each method, which is different from what we observed in the case of ResNet-18. As discussed above, this is likely due to the increase in parameter count between the ResNet-18 and ResNet-56 core networks which can be seen in Figures 3 and 4 in Appendix A.

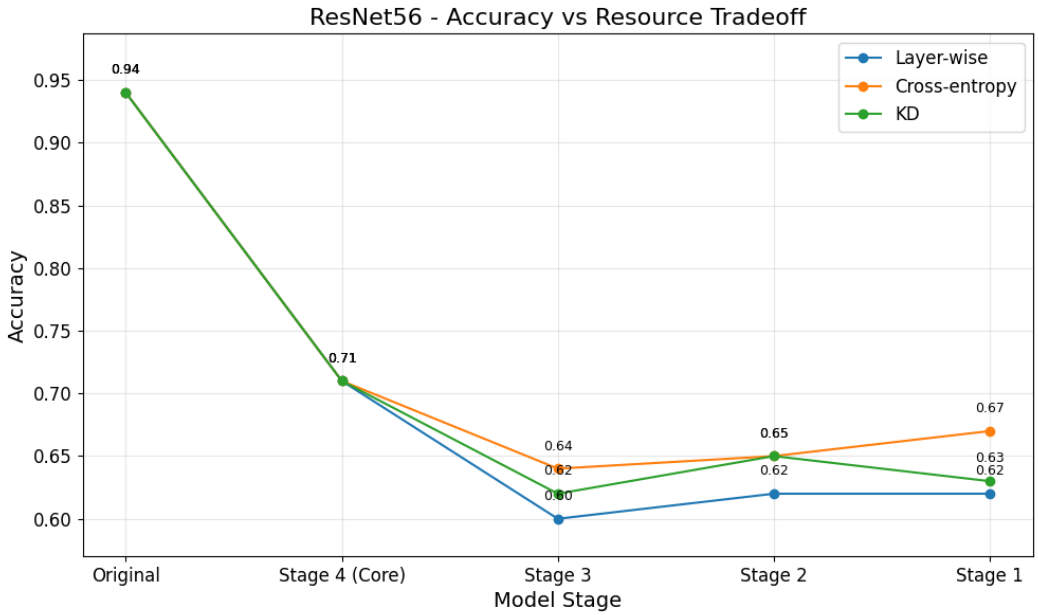


Figure 5.11: ResNet-56 accuracy across methods

5.3.3 Evaluation of the Results

The discrepancy between the training and validation accuracy increases throughout the stages, with stage 1 showcasing the largest gap, as can be seen in the figures above. The performance saturation observed with partial weight freezing is potentially caused by two side effects of partial weight freezing:

- Over-compensation of the trainable weights.
- Increasing constraints on every subsequent descendant network.

The number of parameters increases with each subsequent stage, and so does the number of frozen parameters. With ResNet-18, the network parameters for each stage are displayed in Table 5.1.

Table 5.1: Network Parameters by Reconstruction Stage

Stage	**Total Parameters**	**Frozen Parameters**	**Trainable Parameters**
Core	78,142	0	78,142
Stage 3	118,652	78,142	40,510
Stage 2	180,281	118,652	61,629
Stage 1	272,474	180,281	92,193

The amount of trainable parameters is about half of the frozen parameters for each stage except for the core network. When reconstructing the network, these trainable parameters still receive the same amount of loss as the full network would and thus will be over-compensating for all the frozen parameters. Furthermore, for each subsequent descendant network, the retained and retrained weights of the previous descendant networks are frozen, and this means that the current descendant network has additional constraints on how its weights can be updated while the loss function and hence the optimization of the network does not take into account these additional constraints.

We can only reason from the results that these are the main causes of the saturation problem, but we are not able to prove this due to the black box nature of neural networks.

LWD seems to perform even worse than KD and cross-entropy because it is potentially even more impacted by overcompensation due to partial weight freezing. This is because of the way the loss is calculated, incorporating the sum of mean squared errors that are calculated over all selected feature maps while only a few channels in those feature maps are unfrozen and hence able to update their parameters. The resulting LWD accuracy and loss can be seen in Figure 5.12 and 5.13.

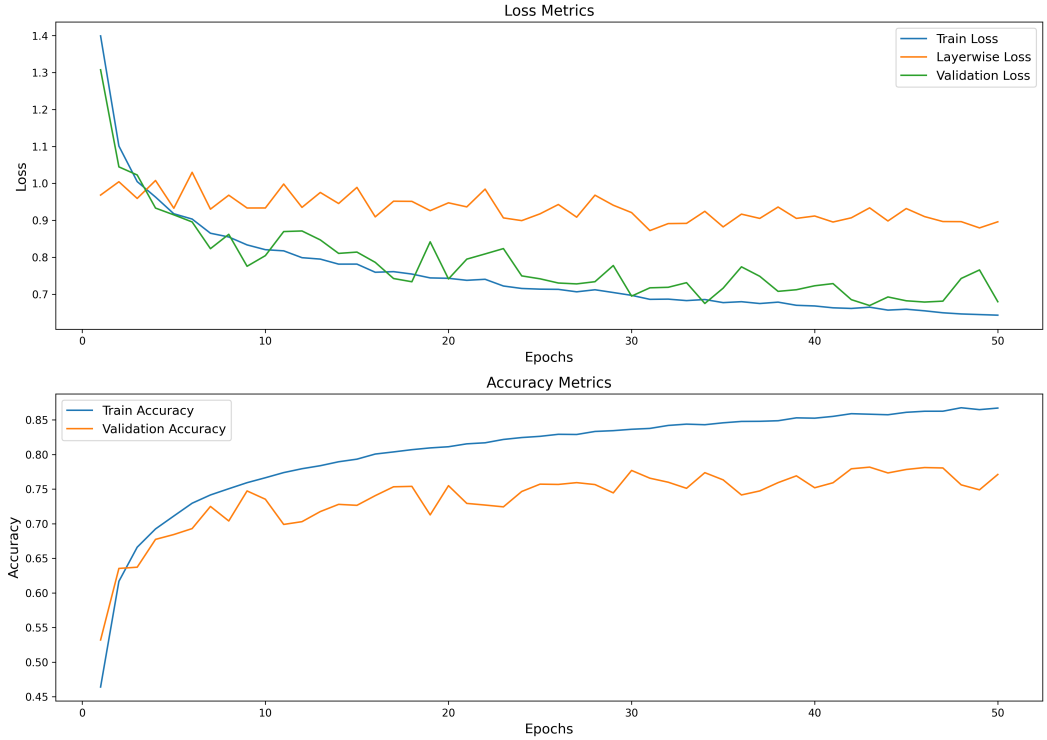


Figure 5.12: Loss and accuracy of ResNet-18 in Reconstruction stage 3

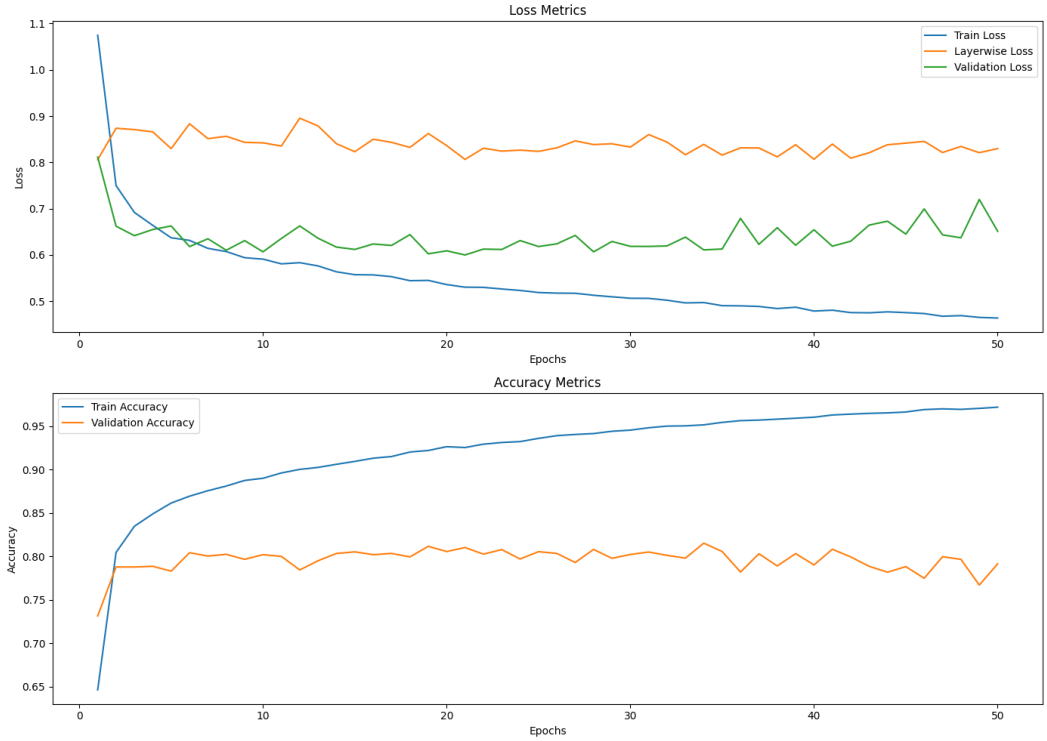


Figure 5.13: Loss and accuracy of ResNet-18 utilizing the layer-wise approach in reconstruction stage 1

Furthermore, the increase of architecture size does not seem to alleviate the performance saturation. While we initially hypothesized that a larger architecture might be more suitable to manage optimizing a model with additional constraints and associated overcompensation of the weights and overfitting of the model, ultimately leading to the saturation problem, we showed that it is not able to do so. This suggests that the core issue lies more with the partial weight freezing mechanism itself and how it is handled in the loss function, rather than simply the capacity of the network to learn.

5.4 Experiment 4: Method Validation: Ball Classification

We would like to show that LWD is a valid method for network reconstruction. In order to showcase this, we want to experiment with non-adaptive architectures so that there is no influence from the partial weight freezing, pruning, and reconstruction methodologies stated above. This will allow us to experiment in an

isolated environment, uniquely monitoring the influence LWD has on network performance in comparison with logit-based KD, and regular cross-entropy loss. For this experiment, two CNNs were constructed, a teacher network with 80.000 parameters and a student network with 40.000 parameters. They are both trained on a classification task using the SPL ball dataset. The objective is to predict whether there is a ball present in 32x32 grayscale images. As shown in Table 5.2, there is almost no loss in performance when halving the network size. This is because the classification task is relatively simple and both teacher and student networks are able to learn adequate internal representations to achieve around a 90% accuracy. Nonetheless, the results in Table 5.2 show that without the adaptive neural network framework and associated partial weight freezing problem, LWD performs just as well or slightly better than the student network trained with KD or cross-entropy loss. The loss function for LWD is identical to Equation 3.1.

Network Type	**Params**	**Train Acc.**	**Val. Acc.**
Student Layer-wise	40k	0.910	0.913
Student KD	40k	0.909	0.913
Student Cross-entropy	40k	0.909	0.912
Teacher Cross-entropy	80k	0.911	0.914

Table 5.2: Comparison of Network Performance

The architectures for both the student networks and the teacher network are displayed in Appendix A.

Chapter 6

Discussion

This chapter discusses the current limitations of both adaptive CNNs and our implementation of LWD. Furthermore, it discusses further research that could be conducted in order to overcome these challenges.

6.1 Current Limitations

Adaptive CNNs utilize partial weight freezing, which over-constraints the descendant models and causes overcompensation in the weights, leading to overfitting and saturation of the model performance. This thesis did not find a solution to the performance saturation problem that occurs due to the partial weight freezing mechanism used in our adaptive CNN framework. Furthermore, in the current implementation of LWD, the weights might be overcompensating slightly more when compared to the KD and cross-entropy methods, which was due to the calculation of the layer-wise distillation loss, as described in 5.3.3.

6.2 Further Research

The current implementation of LWD utilizes MSE to calculate the loss between the teacher and student feature maps. This method was inspired by hint-based KD, but it has shown to lead to unstable optimization in our adaptive CNNs. Further research could be conducted towards fine-tuning the LWD loss function and loss calculation mechanism such that the overcompensation of the weights during reconstruction is reduced, and the stability of weight optimization improved. In order to achieve this, it would require taking into account which weights are frozen, calculating the loss function solely based on the non-frozen parameters.

Instead of LWD, novel techniques could be researched and implemented that

are specifically designed to overcome the saturation problem. These techniques could be focused on taking into account the different descendant networks and their frozen weights. This could change the optimization problem to one where the additional constraints due to freezing are better considered in the loss function.

Once the saturation problem is mitigated, testing in real-world environments with constrained and fluctuating hardware can be done. This research could potentially provide environment-specific knowledge such as optimal pruning ratios and network architectures.

Chapter 7

Conclusion

This thesis set out to research the effects of LWD and network size on the reconstruction performance of adaptive CNNs in image classification tasks.

We have shown that LWD does not provide a solution to the inherent problem of performance saturation due to partial weight freezing. This is due to the effects of partial weight freezing on our LWD implementation as described in Section 5.3.3.

Furthermore, we showed that increasing the network size from ResNet-18 to ResNet-56 cannot overcome the saturation problem either. This suggests that the core issue lies more with the partial weight freezing mechanism itself, rather than simply the capacity of the network to learn, as described in Section 5.3.

Further research will have to be conducted in order to alleviate the saturation problem and make our adaptive CNNs a viable option for deployment on systems with constrained and fluctuating computational resources that require on-board image processing.

Bibliography

[1] Hongrong Cheng, Miao Zhang, and Javen Qinfeng Shi. A survey on deep neural network pruning-taxonomy, comparison, analysis, and recommendations, 2024. URL https://arxiv.org/abs/2308.06767.

[2] lexandros Chariton. Knowledge distillation tutorial¶, 2023. URL https://docs.pytorch.org/tutorials/beginner/knowledge$_d$*istillation*$_t$*utorial.html*.

[3] Pooja Mangal, Sudaksh Kalra, and Dolly Sapra. Towards adaptive deep learning: Model elasticity via prune-and-grow cnn architectures, 2025. URL https://arxiv.org/abs/2505.11569.

[4] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015. URL https://arxiv.org/abs/1512.03385.

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

[6] Mariusz Bojarski, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence D Jackel, Mathew Monfort, Urs Muller, Jiakai Zhang, et al. End to end learning for self-driving cars. *arXiv preprint arXiv:1604.07316*, 2016.

[7] Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton. Imagenet classification with deep convolutional neural networks. *Neural Information Processing Systems*, 25, 01 2012. doi: 10.1145/3065386.

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

[9] Song Han, Jeff Pool, John Tran, and William J. Dally. Learning both weights and connections for efficient neural networks, 2015. URL https://arxiv.org/abs/1506.02626.

[10] Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Quantized neural networks: Training neural networks with low precision weights and activations. *journal of machine learning research*, 18(187):1–30, 2018.

[11] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network, 2015. URL `https://arxiv.org/abs/1503.02531`.

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

[13] Wenhao Sun, Grace Li Zhang, Xunzhao Yin, Cheng Zhuo, Huaxi Gu, Bing Li, and Ulf Schlichtmann. Steppingnet: A stepping neural network with incremental accuracy enhancement, 2022. URL `https://arxiv.org/abs/2211.14926`.

[14] Biyi Fang, Xiao Zeng, and Mi Zhang. Nestdnn: Resource-aware multi-tenant on-device deep learning for continuous mobile vision. In *Proceedings of the 24th Annual International Conference on Mobile Computing and Networking*, MobiCom '18, page 115–127. ACM, October 2018. doi: 10.1145/3241539.3241559. URL `http://dx.doi.org/10.1145/3241539.3241559`.

[15] Rui Han, Qinglong Zhang, Chi Harold Liu, Guoren Wang, Jian Tang, and Lydia Y. Chen. Legodnn: block-grained scaling of deep neural networks for mobile vision. In *Proceedings of the 27th Annual International Conference on Mobile Computing and Networking*, ACM MobiCom '21, page 406–419. ACM, October 2021. doi: 10.1145/3447993.3483249. URL `http://dx.doi.org/10.1145/3447993.3483249`.

[16] Jiahui Yu, Linjie Yang, Ning Xu, Jianchao Yang, and Thomas Huang. Slimmable neural networks, 2018. URL `https://arxiv.org/abs/1812.08928`.

[17] Surat Teerapittayanon, Bradley McDanel, and Hsiang-Tsung Kung. Branchynet: Fast inference via early exiting from deep neural networks. In *2016 23rd international conference on pattern recognition (ICPR)*, pages 2464–2469. IEEE, 2016.

[18] Yizeng Han, Gao Huang, Shiji Song, Le Yang, Honghui Wang, and Yulin Wang. Dynamic neural networks: A survey. *IEEE transactions on pattern analysis and machine intelligence*, 44(11):7436–7456, 2021.

[19] C. J. Van Rijsbergen. *Information Retrieval*. Butterworth-Heinemann, 2 edition, 1979.

[20] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2023. URL https://arxiv.org/abs/1706.03762.

[21] Yann Lecun, John Denker, and Sara Solla. Optimal brain damage. *Advances in Neural Information Processing Systems*, 2, 05 2000.

[22] Namhoon Lee, Thalaiyasingam Ajanthan, and Philip H. S. Torr. Snip: Single-shot network pruning based on connection sensitivity, 2019. URL https://arxiv.org/abs/1810.02340.

[23] Zhonghui You, Kun Yan, Jinmian Ye, Meng Ma, and Ping Wang. Gate decorator: Global filter pruning method for accelerating deep convolutional neural networks, 2019. URL https://arxiv.org/abs/1909.08174.

[24] Saleh Ashkboos, Maximilian L. Croci, Marcelo Gennari do Nascimento, Torsten Hoefler, and James Hensman. Slicegpt: Compress large language models by deleting rows and columns, 2024. URL https://arxiv.org/abs/2401.15024.

[25] Xinyin Ma, Gongfan Fang, and Xinchao Wang. Llm-pruner: On the structural pruning of large language models, 2023. URL https://arxiv.org/abs/2305.11627.

[26] Xin Men, Mingyu Xu, Qingyu Zhang, Bingning Wang, Hongyu Lin, Yaojie Lu, Xianpei Han, and Weipeng Chen. Shortgpt: Layers in large language models are more redundant than you expect, 2024. URL https://arxiv.org/abs/2403.03853.

[27] Yihui He, Xiangyu Zhang, and Jian Sun. Channel pruning for accelerating very deep neural networks, 2017. URL https://arxiv.org/abs/1707.06168.

[28] Adriana Romero, Nicolas Ballas, Samira Ebrahimi Kahou, Antoine Chassang, Carlo Gatta, and Yoshua Bengio. Fitnets: Hints for thin deep nets, 2015. URL https://arxiv.org/abs/1412.6550.

[29] Chen Liang, Simiao Zuo, Qingru Zhang, Pengcheng He, Weizhu Chen, and Tuo Zhao. Less is more: Task-aware layer-wise distillation for language model compression. In *International Conference on Machine Learning*, pages 20852–20867. PMLR, 2023.

[30] Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. Pruning filters for efficient convnets. *arXiv preprint arXiv:1608.08710*, 2016.

[31] Alex Krizhevsky. Learning multiple layers of features from tiny images. *University of Toronto*, 05 2012.

[32] Yaofo Chen. Pytorch cifar models. `https://github.com/chenyaofo/pytorch-cifar-models` . Accessed: 2025-5-17.

[33] A Paszke. Pytorch: An imperative style, high-performance deep learning library. *arXiv preprint arXiv:1912.01703*, 2019.

[34] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017. URL `https://arxiv.org/abs/1412.6980`.

[35] Ilya Loshchilov and Frank Hutter. Sgdr: Stochastic gradient descent with warm restarts, 2017. URL `https://arxiv.org/abs/1608.03983`.

Appendix A: Complementary Figures

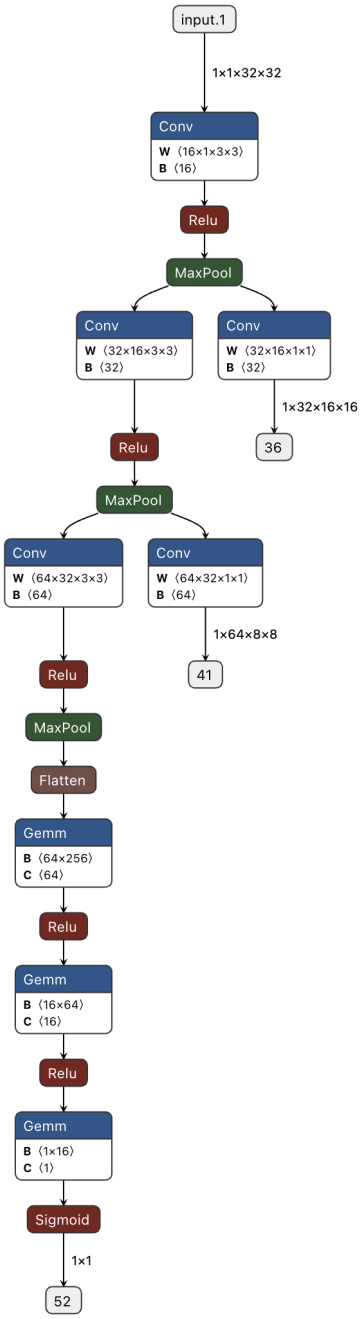


Figure 1: Student network architecture

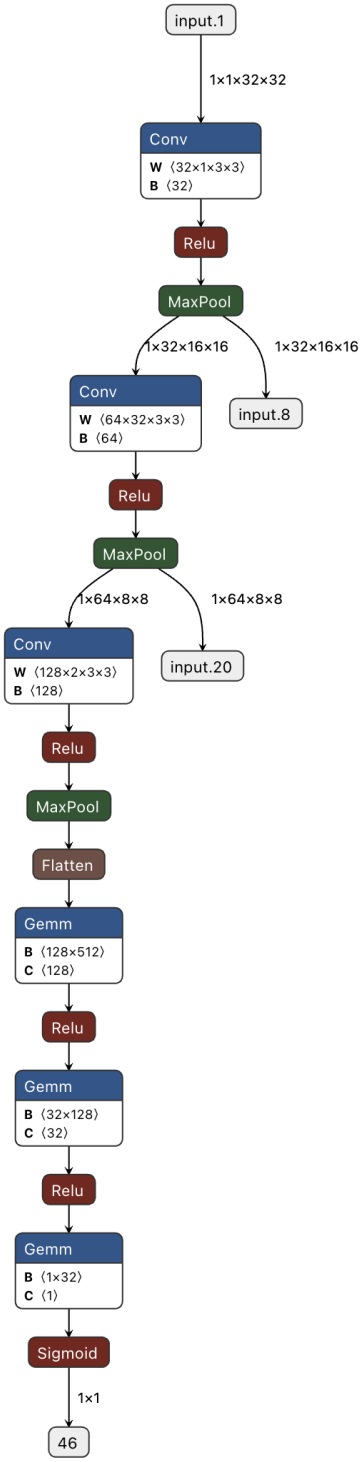


Figure 2: Teacher network architecture

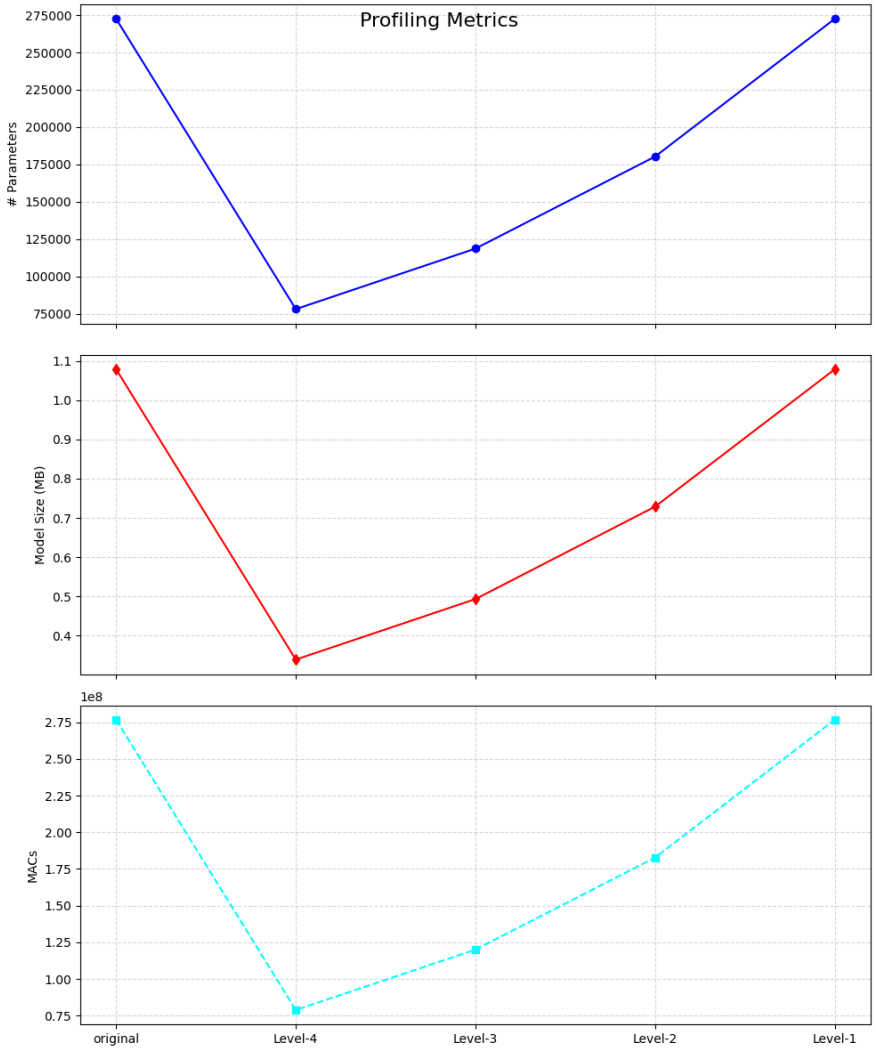


Figure 3: ResNet-18 computational resources per reconstruction stage

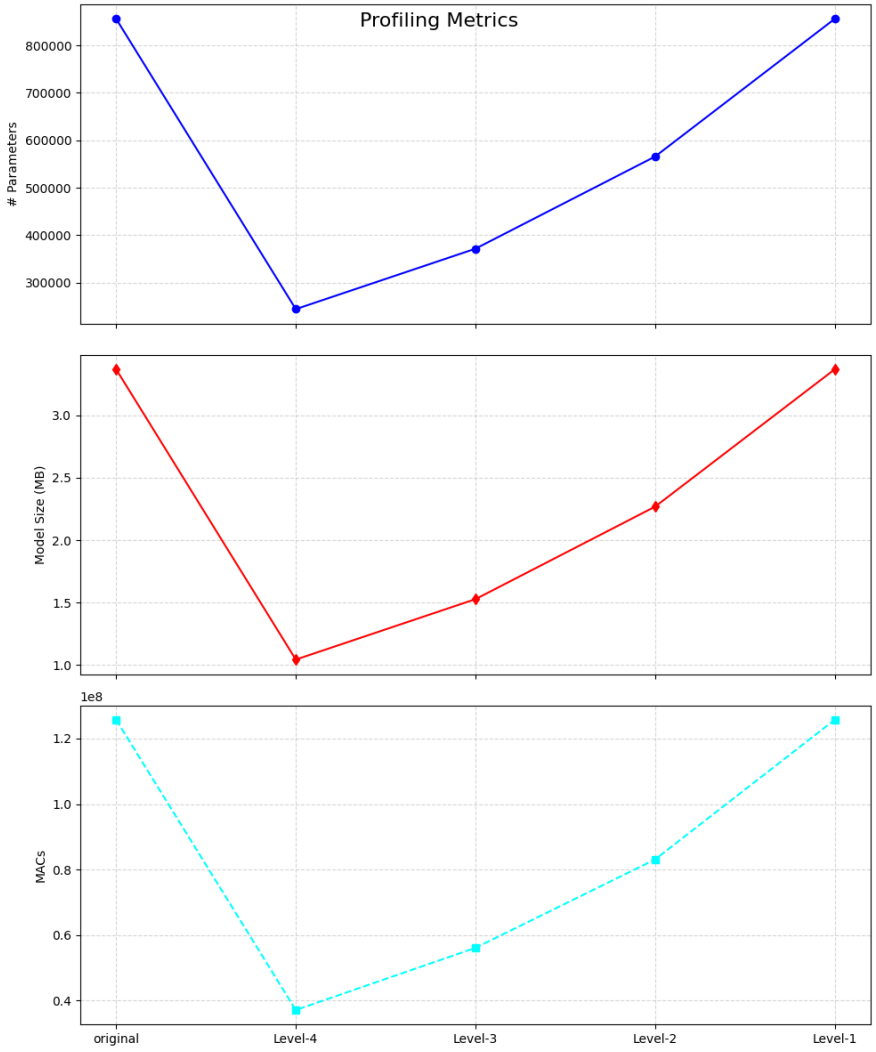


Figure 4: ResNet-56 computational resources per reconstruction stage