THESIS

PRUNING VISUAL TRANSFORMERS TO INCREASE MODEL COMPRESSION AND DECREASE INFERENCE TIME

Submitted by

James E. Yost

Department of Computer Science

In partial fulfillment of the requirements

For the Degree of Master of Science

Colorado State University

Fort Collins, Colorado

Spring 2024

Master's Committee:

Advisor: Darrell Whitley

Sudipto Ghosh
Anton Betten

ABSTRACT

PRUNING VISUAL TRANSFORMERS TO INCREASE MODEL COMPRESSION AND DECREASE INFERENCE TIME

We investigate the efficacy of pruning a visual transformer during training to reduce inference time while maintaining accuracy. Various training techniques were explored, including epoch-based training, fixed-time training, and training to achieve a specific accuracy threshold. Results indicate that pruning from the inception of training offers significant reductions in inference time without sacrificing model accuracy. Different pruning rates were evaluated, demonstrating a trade-off between training speed and model compression. Slower pruning rates allowed for better convergence to higher accuracy levels and more efficient model recovery. Furthermore, we examine the cost of pruning and the recovery time of pruned models. Overall, the findings suggest that early-stage pruning strategies can effectively produce smaller, more efficient models with comparable or improved performance compared to non-pruned counterparts, offering insights into optimizing model efficiency and resource utilization in AI applications.

ACKNOWLEDGEMENTS

I would like to acknowledge and thank many people who have helped me in my pursuit of my Master's and with the work of this thesis. First and foremost I would like to thank Dr. Darrell Whitley for his unwavering support and his expertise in this field. Without Dr. Whitley's support, this thesis would not have been possible. I would also like to thank both Sudipto Ghosh and Anton Betten for taking time out of their busy schedules and being members of my committee. Lastly, I would like to friends and family for their encouragement, without which I would not have been able to get where I am.

DEDICATION

I would like to dedicate this thesis to my family who always inspire me to chase my aspirations

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1

Introduction

1.1 Overview

Artificial Intelligence has become an integral part of our daily lives, from its use in facial recognition to its application in thoracic imaging [1]. One of the most notable advancements in AI technology is the development of ChatGPT, a highly capable chatbot that has revolutionized the field of machine learning. ChatGPT-3 is a large language model built on the transformer architecture with a staggering 175 billion trainable parameters.

Recently, there has been a growing interest in larger AI models due to their superior ability to handle diverse tasks compared to smaller models [2]. For instance, GPT-3, with its enormous number of parameters, outperforms GPT-2, which only has 1.5 billion parameters [3]. Moreover, these larger models are now being integrated into our daily lives, such as the Samsung S24 Ultra, which employs AI to assist users in editing photos directly from their camera app. This means users can perform image manipulation and generation tasks, such as moving the subject of their photo around, and the blank space will be automatically generated. However, it requires an internet connection to execute the edit since the models are too large to run locally on the device [4].

Although larger AI models have their merits, smaller models produced by pruning larger models are also worth looking into due to their compactness and cost-effective training capabilities. It is important to note that it is necessary for these models to start off large to begin the training, then be pruned away to become more compressed. Since they are more compressed, they may be able to run on mobile devices without requiring internet connectivity to execute requests. Their integration into edge computing and handheld devices enables local real-time processing, enhancing privacy, energy efficiency, and personalized experiences.

Inference time is a metric that is used to judge how efficient a model is, measuring the time it takes for data to run through the model. Smaller models can lead to significant energy savings due

to a shorter inference time. The environmental cost of training a single large language model is estimated to be equivalent to 125 round-trip flights from New York and Beijing, or 300,000 kg of carbon dioxide [5]. The computational savings from smaller models will not only be seen during training but also after deployment. Around 90% of the computation and energy consumption of these AI models come from their post-training use [6]. It is vital for companies to decrease the size of the models they are using from both an economic and environmental perspective.

1.2 Contributions

The following are contributions made by this thesis:

- We show by randomly pruning visual transformers, we can achieve similar accuracy levels as fully trained non-pruned models while reducing the overall model size. This approach proves to be resource-efficient, requiring comparable computation time. Pruning leads to decreased inference time of the model, leading to both economic and environmental savings.
- We explore the effects of pruning a model in relation to the "recovery" time of the pruned model and how it dynamically changes based on how dense the model is. We show that there is a cost of training a model too quickly and that there is a trade-off between how quickly the model is pruned and how much accuracy it retains.
- We show that these models can be pruned from the start of training vs. a fine-tuning step while not degrading performance. This shows that pruning from the start of training is an effective way to prune large models and decrease their size.

Chapter 2

Literature Review

2.1 Visual Transformer

The transformer architecture has become the state of the art for natural language processing (NLP) tasks. However, with some minor modifications, this architecture has also been shown to perform quite well on vision tasks. In order to use this architecture, the images are split into patches, and the sequence of linear embedding of these patches is the input to the Transformer (seen in Figure 2.1).

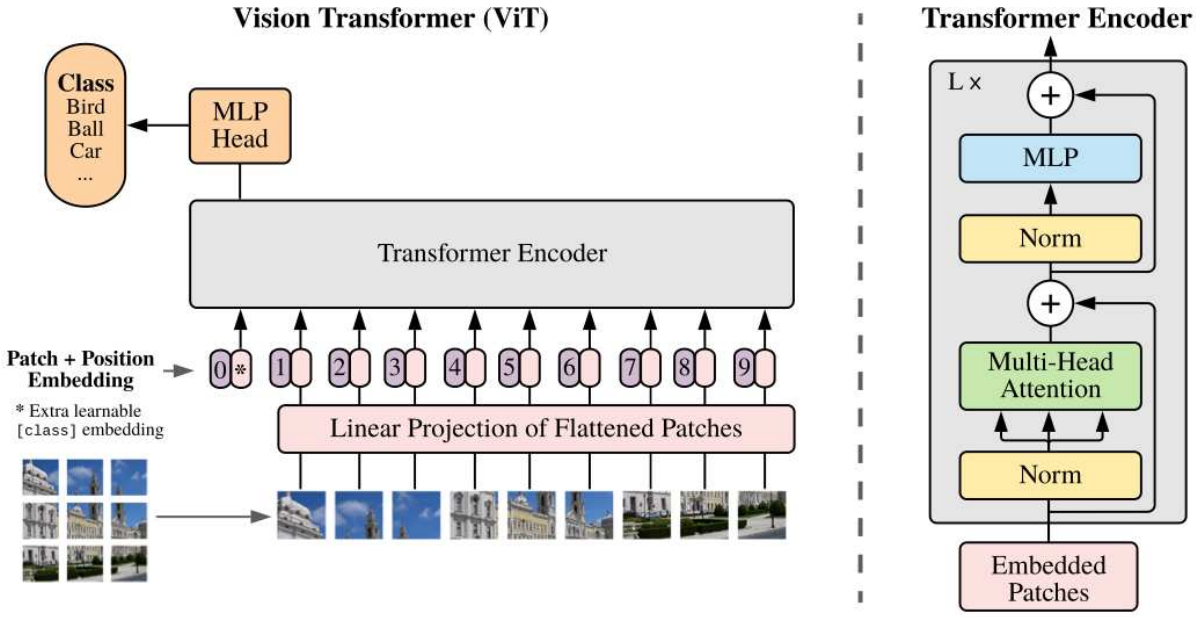


Figure 2.1: Visual Transformer Architecture Overview [7]. Shows the image split into patches. Patches are then linearly embedded, added to position embedding and fed through a sequence of vectors to a standard Transformer encoder. The addition of a "classification token" as an extra learnable embedding allows for classification.

When pre-trained with the public dataset ImageNet-21k and the propriety dataset JFT-300M, this architecture is able to achieve near or beat state-of-the-art image benchmarks. The best model

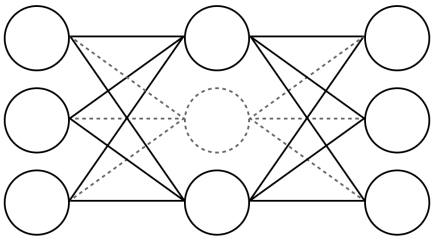
was able to reach 88.55% on ImageNet, 90.72% on ImageNet-ReaL, and 94.55% on CIFAR-100 [7]. The models that performed well are decently large. The models size ranged from 86 million to 632 million parameters. They also had between 12 and 32 encoder layers and 12 to 16 attention heads.

2.2 Types of Pruning

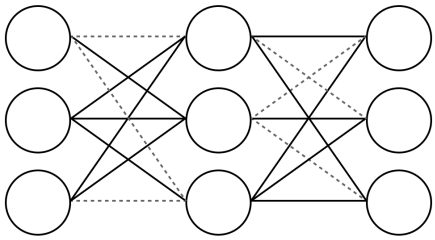
In the literature, there are two types of pruning: structured and unstructured. Unstructured pruning is where individual weights are pruned, not caring where they are in the architecture of a given layer. Structured pruning removes weights in groups, such as removing entire neurons, convolutional channels, or filters [8]. Figure 2.2 shows an example of this. An example of unstructured pruning is Iterative Magnitude Pruning (IMP). IMP prunes the smallest individual weights. The model is then retrained with the weights removed to recover lost performance. This can be done iteratively to get smaller networks [9]. A common example of structured pruning is removing neurons. An example of a neuron that should perhaps be removed is a "dead" rectified linear unit (ReLU). A "dead" ReLU is a neuron with the ReLU activation function that never activates or turns on. This can happen due to the unique ReLU activation function. If the signal into the neuron is a negative value, then the output of the ReLU is zero. ReLU is a piece-wise function; below zero, it is equal to $y = 0$, and above zero, it is $y = x$. With this type of activation, if the signal is negative, the derivative is 0, and if it is positive, it is 1. If the signal into the ReLU has a large negative value, then the ReLU may never activate, leading to it being dead [10].

2.3 Uses of Pruning

Pruning techniques, both structured and unstructured, have emerged as compelling methods for accelerating model inference and reducing model size. Optimizing and using these techniques has become an important field of research. An example of a pruning technique developed is Edgar Pruning (EP) [11], which is a type of unstructured pruning that removes the smallest (insignificant) weights for each prune interval. It also has a rollback function that can bring the network back to its previous state if the loss exceeds a certain threshold. This technique has been used in deep



(a) Structured Pruning Example (Dashed lines are pruned weights).



(b) Unstructured Pruning Example (Dashed lines are pruned weights).

Figure 2.2: An example of Structured Vs. Unstructured pruning (a) Structured pruning: the weights are removed systematically so as to remove the whole neuron. (b) Unstructured pruning: the individual weights are removed in a manner not related to the architecture of the network.

neural networks to decrease training time. Cai [12] shows that structured pruning is beneficial as well with large convolutional neural networks (CNNs). They propose a structured and hardware-friendly pruning technique apart of the family Pruning at Initialization (PAI), called PreCrop. This works by pruning channels of the convolution layers before training, this is done until a optimal layer-wise density is met. With this technique, EffcientNetB0 improved its accuracy by 0.3% on ImageNet with 80% of the parameters.

Movement Pruning, a basic pruning technique, prunes parameters based on a score or rank of the weights [13]. This score is arbitrary and defined by the person implementing it, and based on the score, the weight is either removed or kept. For example, IMP could be considered Movement Pruning if the score is how close the weight is to zero, and we want to remove the weight closest to zero.

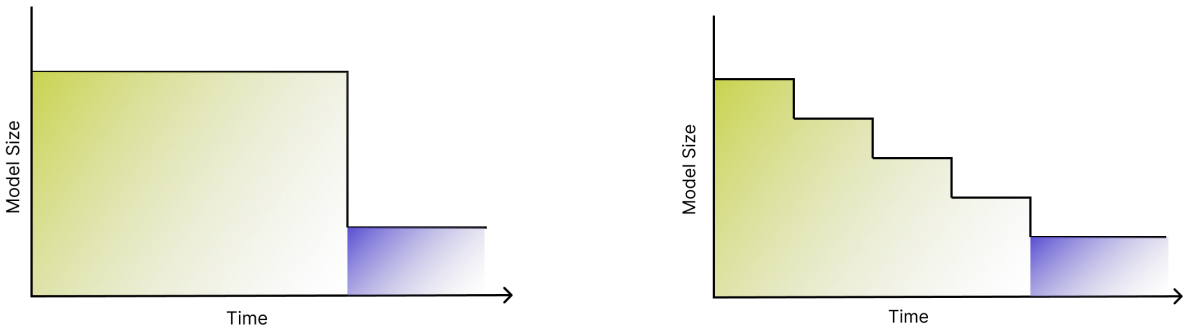
A different type of structured pruning, Block Movement Pruning, which builds off movement pruning, has also been used to decrease the inference time of pre-trained large language models or transformers. Block Movement Pruning has been used to achieve a 2.4x faster and 74% smaller Bidirectional Encoder Representations from Transformer (BERT) model [14, 15]. This technique involves organizing the model's parameters into fixed-sized blocks, which can take advantage of sparse matrix-vector products, and pruning these blocks.

Block pruning has also been shown to be an effective way of pruning pre-trained large transformers. For instance, Li [16] achieved a further 1.79x average compression of DistilBERT with only minor accuracy degradation. It has become common practice to start with pre-trained transformers and show how they can be made smaller and faster with various pruning techniques rather than start from scratch.

2.4 Pruning Scheduling

There are a few different ways to prune when trying to prune from a larger model; the first way is to train the model to convergence and then prune the unwanted weights. The weights that are removed are typically considered to not contribute much to the model's performance. The last step is to re-train the smaller model to regain any lost performance from the pruning. This is the most common way to prune and has no extra computation happening during the initial training [17].

A second way to prune is during training. The model will train in a standard fashion for a certain amount of time and then is pruned by a set amount, typically a small percentage of the model size or a set number of weights. How often the model is pruned is extremely important to this method [17]. If it is pruned too quickly, the model will not have time to "recover" and converge to a high accuracy.



(a) Train then Prune

(b) Prune while Training

Figure 2.3: Different type of pruning schedules Figure is based on [17] (a) Training the model, then prune it, followed by fine-tuning it (b) Prune the model while training, in the end, allows for some time to converge.

Chapter 3

Methodology

We adopted a manual approach to building and pruning a visual transformer. We used the PyTorch framework [18], which enabled us to perform true pruning, not just pseudo-pruning (see section 3.1.1). We had fine-grained access to its models and classes, which allowed us to manipulate them for the purpose of evaluating different pruning strategies.

3.1 Pruning

3.1.1 Pseduo Pruning

PyTorch provides pruning utilities that allow users to prune their models by creating a binary mask that is multiplied by the model weights during the forward pass. This method sets pruned weights to zero, which does not contribute to downstream learning, which is shorthand for forward feed signal propagation and backward unit propagation. The original weights are saved, and this pruning can be reversed. However, if the user wants to make the pruning permanent, they can remove the weights by calling the "remove function", which resets the weights to the product of the binary mask and the original weights.

Weight elimination is just setting the weight to zero. In the forward pass multiplication, it behaves as if the weight is not here because no new information is given to the next neuron. If a whole row of the weight matrix is zero, this is the same as pruning a neuron, and that whole dimension of the matrix does not contribute to any learning. That dimension can be removed from the matrix, reducing the size of the matrix.

PyTorch makes it easy to prune individual weights of a model, but it has a drawback. If we want to prune a whole neuron, i.e., all the weights going out of it, and reduce the dimensionality of the model, we cannot use this method. It just sets that dimension (row/column) of the matrix

to zero, meaning there is no reduction in the model's dimensionality, and there is no saving on computation in the forward pass (see Figure 3.1a).

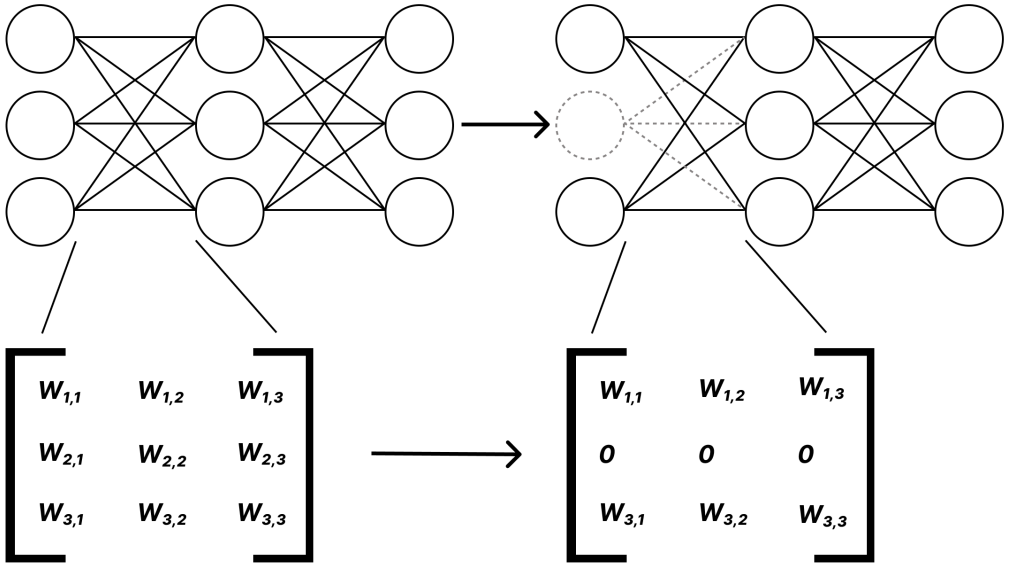
3.1.2 True Pruning

True pruning physically reduces the dimension of a matrix in the model, making it genuinely smaller, as if the neuron was plucked from the model. This leads to faster forward passes because smaller matrices are multiplied together, leading to less computation. Reducing the dimensionality of the matrix also makes the model size smaller, as there are fewer parameters to store.

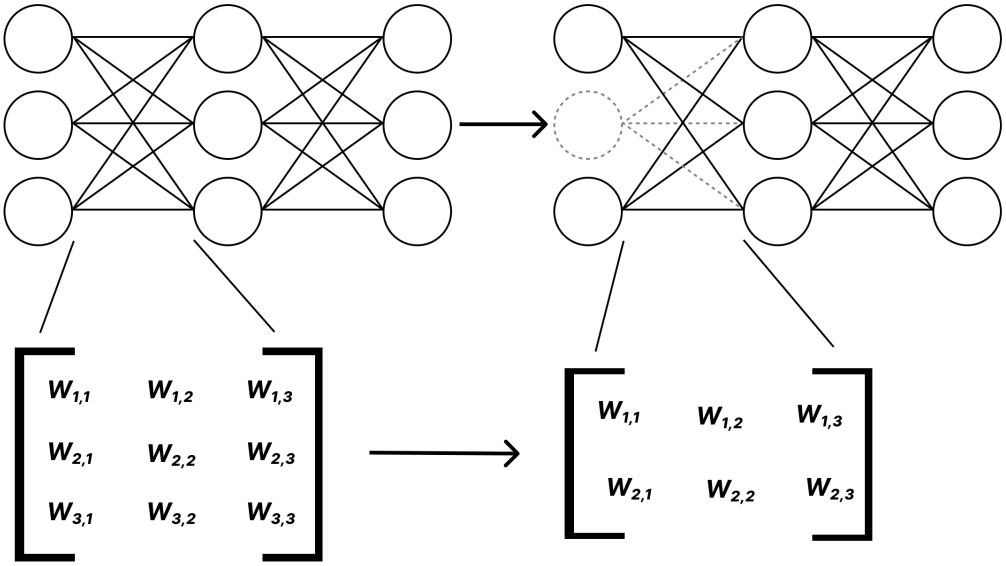
PyTorch does not directly support this type of pruning. Therefore, we had to build a custom way to prune the model. PyTorch provides low-level access to the models, enabling us to prune them.

To prune each layer, we created a fresh layer with the dimensions we wanted. We then copied over the old layer's biases and weights, excluding the dimensions we were pruning. We carefully kept track of the previous parameters' output dimension as that would be the new layers' input dimension. This approach took care of the physical model's pruning. However, we also needed to update the parameters the optimizer would use. To do this, we created a new optimizer and transferred over the state of the old optimizer for each parameter, excluding the dimensions of each parameter that had been pruned. To test our pruning method and ensure that it was working as expected, we carefully tested it by watching the size and values of the tensors and comparing them to what we expected. We also tested the pruning technique by pruning no neurons, which would reproduce the exact same model. An example of Pseudo vs. True Pruning is shown in Figure 3.1.

The number of dimensions removed at once was equivalent to the number of patches, allowing for a mathematically simple reduction of the embedding dimension. The number of patches was eight, so the embedding dimension $8 * 48 = 384$. This reduction was also convenient as it was also the same number of attention heads used.



(a) Pseudo Pruning



(b) True Pruning

Figure 3.1: An example of what weight matrices will look like in (a) Pseudo vs. (b) True pruning (a) The dimension of the matrix is set to zero, therefor pruning that neuron, but keeping the total dimension the same, (b) The dimension is removed from the matrix and dimension three becomes the new dimension two.

3.2 Learning Rate Scheduler

A learning rate scheduler dynamically changes the learning rate as training takes place based on certain criteria or a predefined schedule. This is useful as it improves convergence speed, stability,

and generalization performance. We faced difficulties using off-the-shelf schedulers from PyTorch because they did not support updating each parameter the way we needed. To overcome this issue, we created our own scheduler that updates the learning rate of the optimizer. Our scheduler has five warm-up epochs with a linear scheduler; this linearly increased the learning rate from $1 \cdot 10^{-8}$ to the initial learning rate of $1 \cdot 10^{-3}$. This allowed the model to slowly warm up. After the warm-up period, we used a cosine annealing learning rate scheduler for the decay of the learning rate for the remaining epochs. We used traditional approaches to both types of schedulers. We started with an initial learning rate of $1 \cdot 10^{-3}$, which was then reduced to a final learning rate of $1 \cdot 10^{-5}$ during training; after testing, we found this to be the best learning rate. This decay allows the model to search quickly at the start of training and gradually converge towards the end. You can see an example of this learning rate curve for 500 epochs in Figure 3.2.

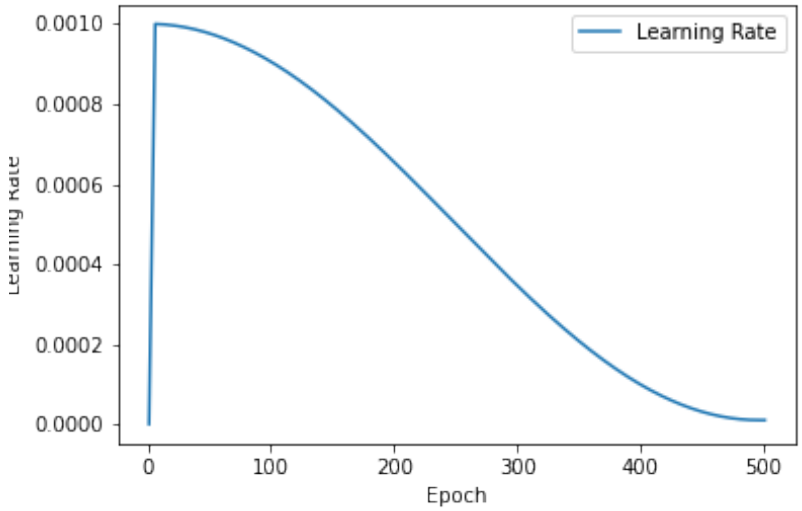


Figure 3.2: Learning Rate Scheduler Example, initial learning rate being $1 \cdot 10^{-3}$, and final learning rate $1 \cdot 10^{-5}$. Five warm-up epochs, and then cosine annealing decay.

3.3 Pruning Scheduling

As discussed in Section 2.4, there are two main methods for pruning large models to make smaller models. We decided to prune while we trained the models (see Figure 2.3b) as this allows us to reap the reward of a smaller network and faster inference time during training. This method of pruning introduces a new hyperparameter, which determines how often we prune the model. We tested various hyperparameter values in our experiments to see which works best (see Table 3.1.)

3.4 Model Architecture

We employed a visual transformer model inspired by the design available on GitHub, originally developed by Omiita [19]. The model was built off the architecture seen in section 2.1 due to its great performance and marginal changes from the original transformer architecture. Our model used eight attention heads for each transformer encoder, with seven encoder layers. We set the input dimension of the multi-layer perception (MLP) to 384, which is a multiple of both the patch size for image processing and the head size of the encoder. It scaled to 1536 dimensions internally. Overall, our model has 1.2 million trainable parameters.

3.5 Dataset

Our model was trained on the Cifar-10 dataset using Pytorch datasets [20]. This is a standard dataset that consists of 32 by 32 color images and ten prediction classes. Some include airplanes, birds, cats, deer, dogs, ships, and more. We split the dataset into a training and test set. We randomly cropped and flipped the training set horizontally and then normalized it. We then augmented it with randomly chosen two of twenty-five sub-polices on Cifar-10, ranging from inverting the image's colors, rotating the image on a point, shearing around the X or Y axis, changing brightness by a magnitude of 0.9, and image translation along the Y axis. The test set was normalized.

3.6 Optimizer

We used the AdamW optimizer for these experiments because we found that it worked better in practice than just the Adam optimizer. Adam works by adjusting the learning rate for each parameter based on the gradients' past behavior, combining techniques like momentum and RMSprop(Root Mean Squared Propagation) to update parameters efficiently [21]. AdamW is a variation of Adam that incorporates weight decay directly into the update step for the model parameters [22].

When pruning, we would create a new AdamW optimizer and transfer over the old optimizer's state while excluding the state's dimension for a given parameter that we wanted to prune. This was to keep the optimizer from "restarting" every time we pruned.

3.7 Hyperparameters

We adjusted many hyperparameters through training and experiments; Table 3.1 is a list of the parameters and their use.

Table 3.1: Hyperparameters Table

Parameter	Use
epoch	Number of epochs to train
desired accuracy	Train to a desired accuracy (overrides epoch)
train time	Train for a set amount of time (overrides epoch)
prune	If we want to prune
prune delay	Initial wait before pruning (in epochs)
prune frequency	How often to prune the model (in epochs)
prune frequency decay	How much to change the prune frequency by
prune frequency decay frequency	How often to decay prune frequency

3.8 Experiments

3.8.1 Random Pruning

Epoch Training

In this type of experiment, we use the standard approach of training for a certain amount of epochs. We would train the model for a set number of epochs and keep the data for the best-performing model based on the test set. We trained with a batch size of 128 images and used the ReLU activation function. We altered the number of epochs the model was training for and how often it was pruned to get a better understanding of how pruning affected the model. We would keep a constant seed per attempt and for each pruning frequency.

Training for a Fixed Time Budget

In this type of experiment, we would train the models for 3 and 5 hours. This allowed us to see if the pruned model could achieve a similar accuracy as a non-pruned model when time, not epochs, was the limiting factor. We needed to do this because each epoch would execute faster as the pruned model trained and was pruned. This means that the pruned model could complete more epochs than the non-pruned model during the same amount of time, potentially allowing it to learn more.

Training for a Desired Accuracy

In this type of experiment, we allowed the model to train to the desired accuracy. This was to get a more direct comparison between the pruning and non-pruning and see if there was a speed-up of training time, not necessarily just a change in the number of epochs. We put some boundaries on training to ensure that this would not run forever. If the pruned model were to get too small, somewhere in the range of 95% pruned, or trained too long, 1000 epochs, then we stopped training. We chose 0.88 and 0.89 for the desired accuracy as most models converged to 0.88, and 0.89 was a good way to see how the pruned behaved with higher accuracy.

3.8.2 Dead ReLU Pruning

We thought about running experiments where we pruned dead ReLUs in the model. However, in the architecture, seen in Figure 2.1, the only place where there could be dead ReLUs is in the MLP. This is because this section in the transformer architecture is the only one with activation functions. If we wished to prune the edge, or outer dimension, of the MLP, we would need to prune the whole dimension of the encoder, both of the norms and the multi-head attention, to keep the dimensions between the encoders consistent due to the residual connections. We would also have to prune only the dead neurons that are dead across all the encoders to keep the dimensions consistent, as the input and output dimension of each encoder needs to be equivalent due to the residual connection in each encoder block.

The residual connection between the output of the MLP and the concatenated multi-head attention can lead to the neuron turning back on because gradient information can be passed around the dead ReLU. This can potentially restart the dead ReLU. Due to the constraints of pruning dead ReLUs, we opted not to continue with this type of pruning.

Chapter 4

Results

4.1 Standard Epoch Training

One of the primary questions we wanted to address was if pruned networks could achieve accuracy similar to that of the non-pruned models. We trained the models for 200 and 500 epochs, with pruning steps of 15, 20, 25, and 30 epochs. We repeated each of these experiments five times to ensure we saw a trend. The models were trained using a Titan V GPU and used the learning rate scheduler as discussed in Section 3.2.

Looking at the test accuracy for the 200 epoch experiment (see Table 4.1), we get similar performance in terms of accuracy, but we do notice that the non-pruned model does perform slightly better. This can be seen in the cropped graph in Figure 4.1. We can also see that the average pruning of the models ranges from 29% to 15%. We were able to achieve a similar performance as the non-pruned model in less time while also getting a smaller model. We can see that we save around five minutes of training time during these 200 epochs when pruning every 30 epochs. This is expected as there is no reduction in the weight matrices in the non-pruned model.

Table 4.1: Test accuracy after training for 200 Epochs

		Frequency of Pruning			
Run	No Prune	15 epochs	20 epochs	25 epochs	30 epochs
1	0.8877	0.8766	0.8766	0.8795	0.8813
2	0.8816	0.8764	0.8748	0.8825	0.8781
3	0.8823	0.8731	0.8789	0.8821	0.8829
4	0.8823	0.8767	0.8765	0.8789	0.8824
5	0.8874	0.8799	0.8792	0.8823	0.8847
Avg:	0.8844	0.8765	0.8772	0.8812	0.8819
σ :	0.003	0.0024	0.0018	0.0018	0.0024
Avg.% Pruned:	0%	29.01%	23%	19.63%	15.62%
Avg. Time:	02:18:47	02:03:48	02:08:42	02:12:16	02:13:27

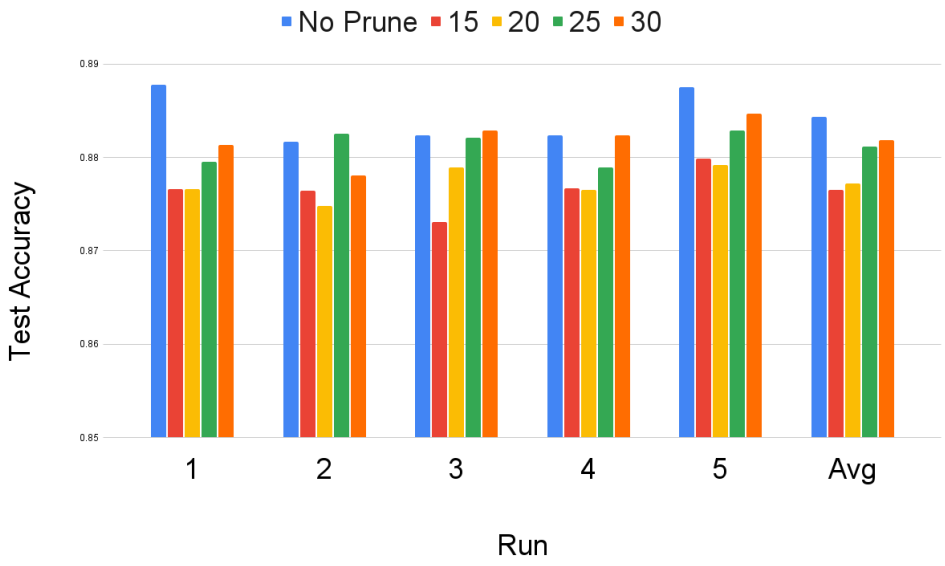


Figure 4.1: Accuracy of each try/ after training for 200 epochs, (blue) no pruning, (red) pruning every 15 epochs, (yellow) pruning every 20 epochs, (green) pruning every 25 epochs, (orange) pruning every 30 epochs.

Looking at the model being trained for 500 epochs in Table 4.2, we again see that we also get similar performance between the pruned and non-pruned models. Here, we see that we also get a higher pruned percentage ranging from 54% to 36%. This makes sense because the model has more time to train, and therefore, it has more pruning steps. We also see that it achieves similar performance with increased pruning.

Another advantage of pruning while training is the decreased inference time, which leads to reduced epoch time. This is apparent in the average train times in Table 4.4. The non-pruned model took the longest at 5 hours and 46 minutes to finish, and the model that was pruned every 15 epochs was the fastest at 4 hours and 4 minutes. It took 141% longer to train the non-pruned model in this case.

Table 4.2: Test Accuracy after Training for 500 Epochs

		Frequency of Pruning			
Run	No Prune	15 epochs	20 epochs	25 epochs	30 epochs
1	0.8944	0.8884	0.8895	0.8957	0.8981
2	0.8989	0.8818	0.8864	0.8865	0.8903
3	0.8971	0.8868	0.8881	0.8896	0.8904
4	0.8974	0.88568	0.8898	0.8914	0.8925
5	0.9067	0.8931	0.89550	0.8960	0.8987
Avg:	0.8989	0.88725	0.8899	0.8918	0.8939
σ :	0.004	0.004	0.003	0.003	0.003
Avg. % Pruned:	0%	54.04%	45.73%	40.41%	35.84%
Avg. Time:	05:46:04	04:04:19	04:30:07	04:45:50	04:57:19

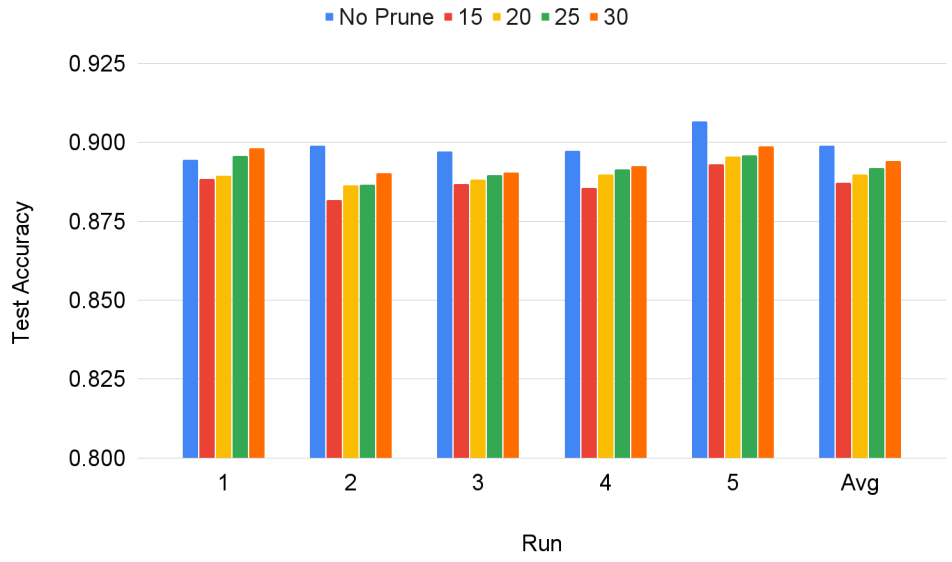


Figure 4.2: Accuracy of each try/ after training for 500 epochs, (blue) no pruning, (red) pruning every 15 epochs, (yellow) pruning every 20 epochs, (green) pruning every 25 epochs, (orange) pruning every 30 epochs.

We can see the drastic decrease in training time from around 41 seconds to below 28 seconds in Figure 4.3. This model in Figure 4.3 got to around 36% pruned. Training time per epoch is directly tied to how large the model is. With the training time per epoch decreasing, we know that the pruning while training will finish training faster than a non-pruned model.

The model in Figure 4.3 finished the 500 epochs in 4 hours and 57 minutes, whereas the non-pruned model, with the same seed, took 5 hours and 46 minutes. The non-pruned model took 116%

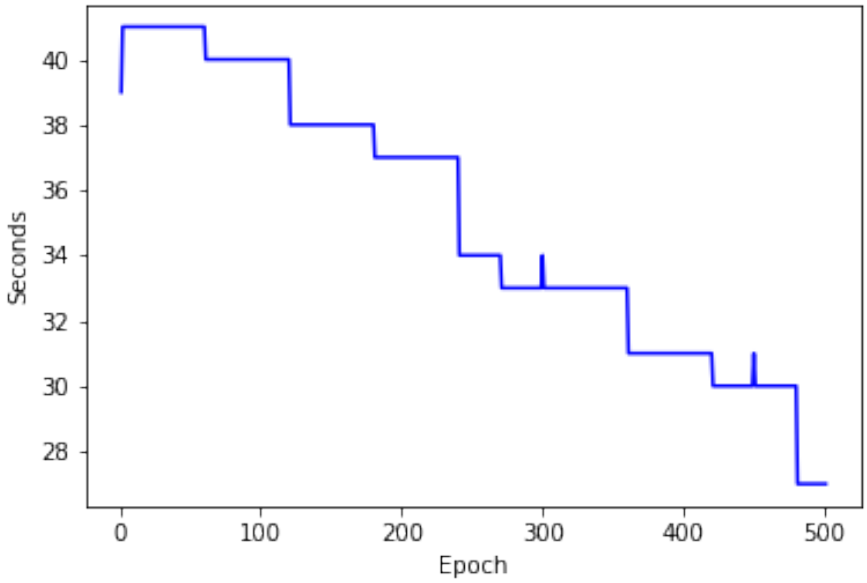


Figure 4.3: How long the model takes to complete each epoch, including the testing step, pruning every 30 epochs.

of the time that the pruned model did to train. The non-pruned model does perform slightly better than the pruned model, but the pruned model is limited because of the recovery time. A more fair comparison between these models would be to allow them to train for the same amount of time.

4.2 Training for a Fixed Time Budget

For this experiment, we trained the models for a set amount of time: three and five hours. Each iteration was run on machines with the same hardware. Each device had no other processes running on the GPU, as this would impact how much it could train in the set amount of time. We omitted the 15-epoch prune step for the five-hour training, as this would end up pruning the whole model.

When training for three hours, the accuracy between pruned and non-pruned is very close, as seen in Table 4.3. Figure 4.4 shows that the pruned models beat the non-pruned model in some cases and got beaten in others. The differences in the accuracy of the pruned vs. non-pruned models are marginal, with 0.4% being the largest seen in Table 4.3 run number two.

Table 4.3: Accuracy after 3 hours of training

		Frequency of Pruning			
Run	No Prune	15 epochs	20 epochs	25 epochs	30 epochs
1	0.8829	0.8808	0.8819	0.8831	0.8846
2	0.8858	0.8834	0.8844	0.8873	0.8893
3	0.8867	0.8834	0.8882	0.8857	0.8883
4	0.8924	0.8881	0.8926	0.8904	0.8922
Avg.	0.8869	0.8839	0.8868	0.8866	0.8886
σ	0.004	0.003	0.004	0.003	0.003
Average Pruned:	0%	49.66%	36.19%	30.3%	24.93%

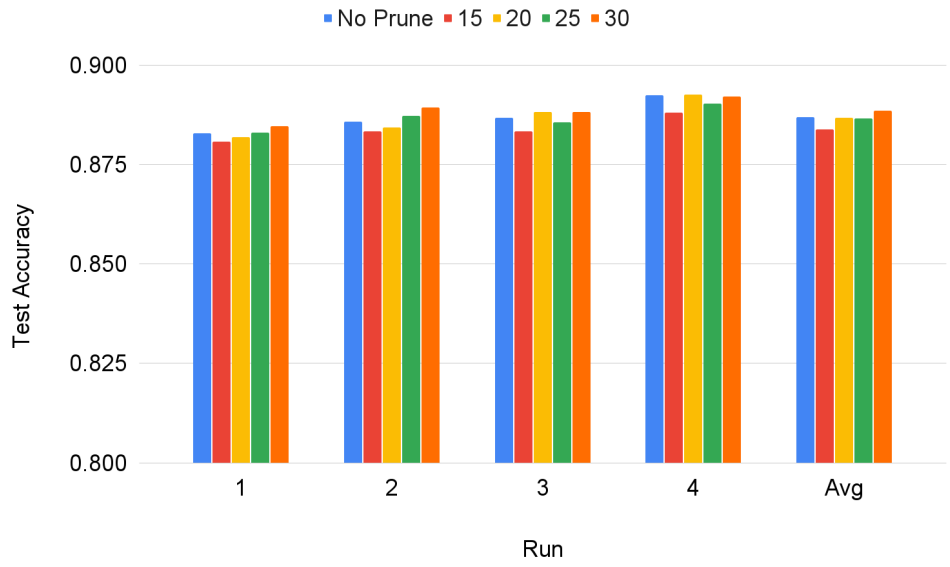


Figure 4.4: Accuracy of each try after 3 hours of training, (blue) no pruning, (red) pruning every 15 epochs, (yellow) pruning every 20 epochs, (green) pruning every 25 epochs, (orange) pruning every 30 epochs.

This shows us that given the same limited amount of time, the pruned model can perform just as well, if not better in some cases, than the non-pruned model, with the added benefit of getting a smaller model. The average amount the models were pruned ranged from 15% to nearly 50%.

Table 4.4: Accuracy after 5 hours of training

		Frequency of Pruning		
Run	No Prune	20 epochs	25 epochs	30 epochs
1	0.8989	0.8857	0.89184	0.8914
2	0.8968	0.8879	0.8908	0.8917
3	0.8973	0.8893	0.8909	0.8953
4	0.9067	0.8938	0.8976	0.8981
Avg.	0.8999	0.8892	0.8928	0.8942
σ	0.005	0.003	0.003	0.003
Avg. % Pruned:	0%	48.53%	38.77%	37.51%

If the models had five hours to train, we see in Table 4.4 that the results are close. However, the non-pruned model does beat the pruned model in all cases. These pruned models are approaching 50% the size of the model pruned in some cases.

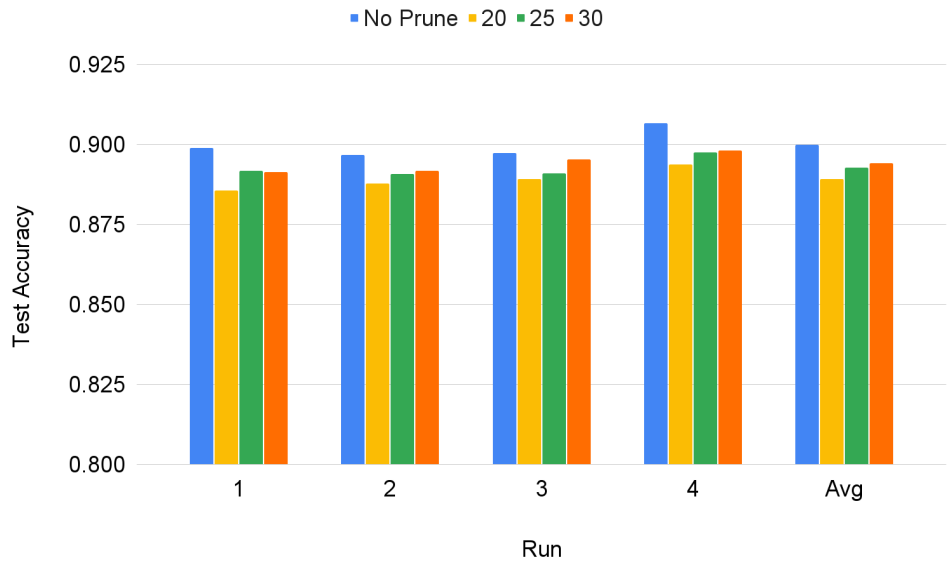


Figure 4.5: Accuracy of each try after 5 hours of training, (blue) no pruning, (yellow) pruning every 20 epochs, (green) pruning every 25 epochs, (orange) pruning every 30 epochs.

If we want a very compressed model and maintained accuracy, it has to be trained and pruned slowly. We give up the speed up in training time for extra compression of the model. For example, with run four and the prune interval of 20 epochs, Figure 4.6, it seems that the model starts to level

off, then slowly loses accuracy as the model becomes smaller, and the pruning frequency is too fast for the model to recover. There is a major drop off in performance toward the end of training. We talk more about model compression and side effects of pruning in Section 4.4 and 4.5 respectively.

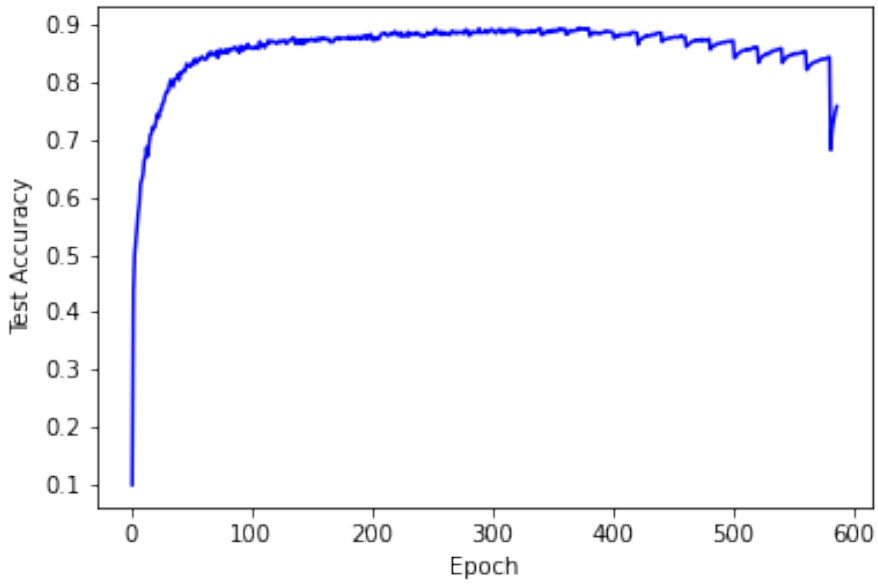


Figure 4.6: Test Accuracy of pruning every 20 epochs for five hours. Showing that the model slowly loses accuracy, as it does not have time to recover.

4.3 Training for a Desired Accuracy

Training until we get the desired accuracy gives us a great way to directly determine if there is some speed up in total training time. For this experiment, we decided that for a model to be finished, the desired accuracy is 88%. We chose this as it was a good balance between high accuracy and lower training time.

We re-ran the experiments seven times with different seeds (1 to 7). We ran the prune frequencies of 20, 25, 30, 40, and 50 epochs epochs. We did try lower frequencies, such as 5 and 10. We found that these tended to prune too quickly and did not converge to the desired accuracy very often. We can see the results of these experiments in Table 4.5.

Table 4.5: Desired Accuracy Accuracy 0.88 (500 epoch LR curve)

		Frequency of Pruning				
Run	No Prune	20 epochs	25 epochs	30 epochs	40 epochs	50 epochs
1	2:20:58	2:24:52	2:32:50	2:18:29	2:11:02	2:04:01
2	2:18:47	2:22:21	2:23:43	2:16:06	2:20:15	2:12:25
3	2:20:04	2:07:42	2:07:27	2:14:09	2:21:42	2:14:08
4	1:56:45	1:54:05	2:01:46	1:59:39	1:54:50	1:52:43
5	2:07:42	2:11:03	2:17:28	2:00:14	2:13:29	2:11:02
6	2:03:43	2:05:17	2:09:19	1:55:14	2:02:42	2:09:34
7	2:31:18	2:16:30	2:32:10	2:27:29	2:34:29	2:45:06
Avg.	2:14:11	2:11:41	2:17:49	2:10:11	2:14:04	2:12:43
Avg. % Pruned:	0%	28.22%	23.14%	17.51%	13.37%	9.58%

Across the rows in Table 4.5, there is no extreme difference in the time it took to converge. We can also see from the table that sometimes pruning is faster to converge, sometimes not, but we always get a smaller model from pruning, which will significantly reduce the inference time of the model once it is deployed. On average, we got a faster time to converge when pruning every 30 epochs, and it beat its non-pruned counterpart six out of the seven times.

In run five, the non-pruned model took 185 epochs to converge, whereas when pruning every 20 epochs, it took 205 epochs to converge. Pruning may take more epochs to converge, which is why there is not always a speed-up. If it were a constant number of epochs, there would always be a free speed-up. However, the model needs time to recover from pruning because the search space changes every time it is pruned.

From Figure 4.7, we see no clear winner in terms of which training method is faster. On some occasions, the non-pruned model wins, and on others, the pruned model wins. We see that, on average, pruning every 30 epochs yields a faster convergence time. The average savings is around four minutes, which is around three percent, but we also get a smaller model during the training as well.

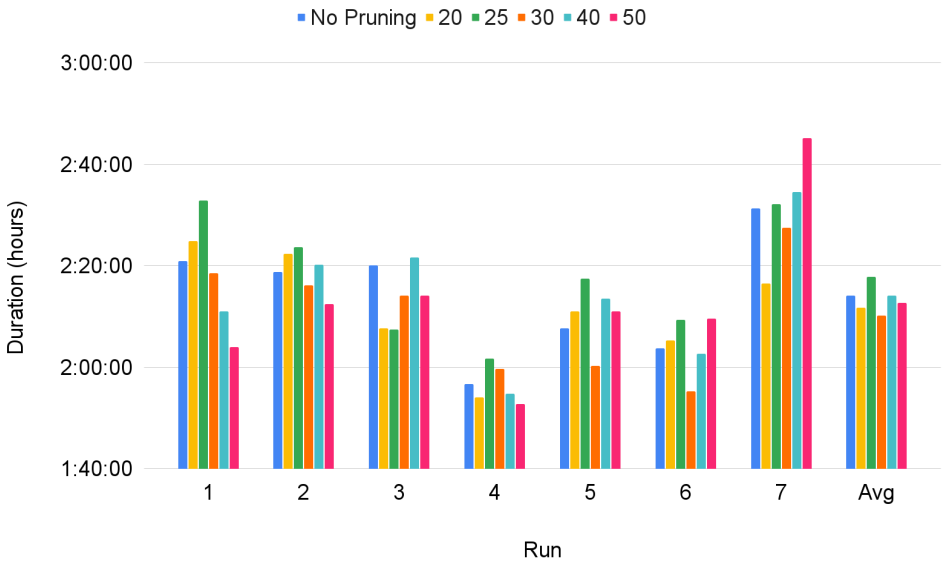


Figure 4.7: Time it took for each try to get to the desired accuracy, of 88%, lower value is faster. (blue) no pruning, (yellow) pruning every 20 epochs, (green) pruning every 25 epochs, (orange) pruning every 30 epochs, (light blue) pruning every 40 epochs, and (magenta) pruning every 50 epochs.

To see if the models converge to a higher accuracy of 0.89, we re-ran the experiment with pruning frequencies of 30, 40, and 50 epochs. We did not run smaller pruning steps, as they had difficulty converging to the desired accuracy.

Table 4.6: Desired Accuracy Accuracy 0.89 (500 epoch LR curve)

		Frequency of Pruning		
Run	No Prune	30 epochs	40 epochs	50 epochs
1	3:50:14	3:17:49	3:01:43	3:33:48
2	3:39:31	3:10:49	3:42:05	2:57:08
3	3:22:33	3:24:20	3:20:42	3:10:31
4	2:50:49	2:38:13	2:21:57	2:57:26
5	3:11:33	2:54:06	3:02:02	2:56:28
6	2:55:57	3:08:21	2:59:55	3:08:32
7	3:39:05	3:18:46	3:53:39	3:40:51
Avg.	3:21:23	3:07:29	3:11:43	3:12:06
Avg. % Pruned:	0%	26.69%	18.71%	15.87%

In Table 4.6, we see that the average models per pruning frequency are smaller than their 0.88 accuracy counterpart; this makes sense as they have to train for longer to get to the higher accuracy. The models range from 26% pruned to 16% pruned, which is a significant reduction in model size. We see a similar pattern in Figure 4.8 as we see in Figure 4.5, where on average, the pruning is faster, sometimes by a significant margin, but it is dependent on the run. Overall, when training to a desired accuracy, we had a slight speed up on average while getting denser models for no extra computational cost.

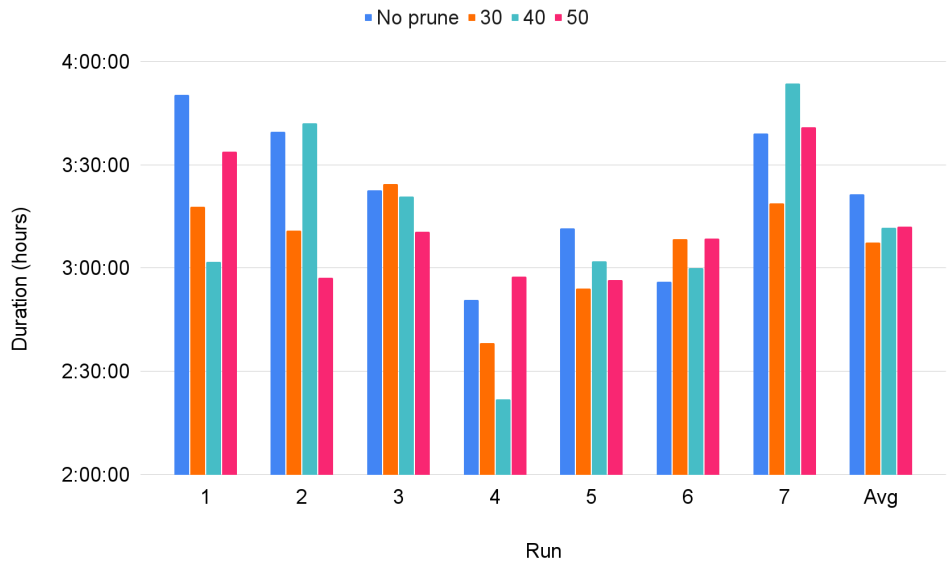


Figure 4.8: Time it took for each try to get to the desired accuracy, of 89%, lower value is faster. (blue) no pruning, (orange) pruning every 30 epochs, (light blue) pruning every 40 epochs, and (magenta) pruning every 50 epochs.

4.4 Model Compression

There is a trade-off between how fast a model can be pruned and how much accuracy is retained. We have seen this in previous experiments. In Figure 4.6, we do see that the accuracy starts to trail off, and the recovery seems to take more time. To show how far we can take compression, we also introduced hyperparameters that decay the pruning frequency and alter how often to decay

the pruning frequency. We trained the model for 2000 epochs for this experiment to allow it plenty of time to recover. We pruned the model every 40 epochs, with a prune delay of 50 epochs, meaning the first epoch it was pruned was epoch 90. We decayed the prune frequency by five epochs every five prunes. This means that every five prunes, the prune step decreased five epochs, from 40 to 35 to 30, and so on. When the prune frequency went negative, it stopped pruning the model.

Ultimately, this model achieved 0.8989 test accuracy while being pruned by 84.26%. We can see the test accuracy in Figure 4.9. The model performed as expected at the start of training; however, as the pruning frequency increased, the model's performance suffered. Once the pruning was finished, the model was able to recover and achieve high accuracy.

It goes up as we expect, and once it is getting pruned very quickly, then the model is not able to recover. Once the prune frequency went negative, it stopped pruning the model and was allowed a long time to recover. We see that it could recover its performance from the extreme pruning.

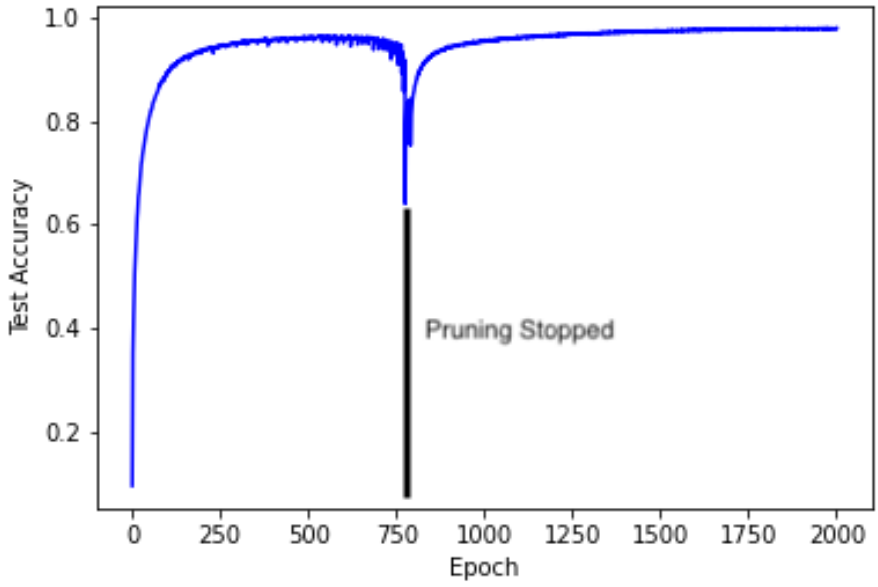


Figure 4.9: Test accuracy for compression, the model was trained for 2000 epochs, with an adjusting prune step, starting at 40, working its way down to 5, and then stopping pruning. A major dip in performances from pruning exponentially faster, not allowing the model time to recover. Once pruning was finished, the model (around 750 epochs) was able to recover.

The model size started off at around 45MB and finished at less than 7 MB (see Figure 4.10). The model size stays constant once we stop pruning. The model took around 12 hours and 8 minutes to complete its training. This is longer than training the original model to completion; however, we get similar accuracy with a much smaller model, leading to a much quicker inference time. The average time per epoch at the start of training was around 41 seconds or 12 iterations per second, whereas once the model was fully pruned, it was 15 seconds or 31 iterations per second. If a user had one million images to run through this model, the non-pruned model would take nearly a whole day to complete the request, at around 23 hours. If we ran it through the pruned model, it would take just a work day, at around 9 hours.

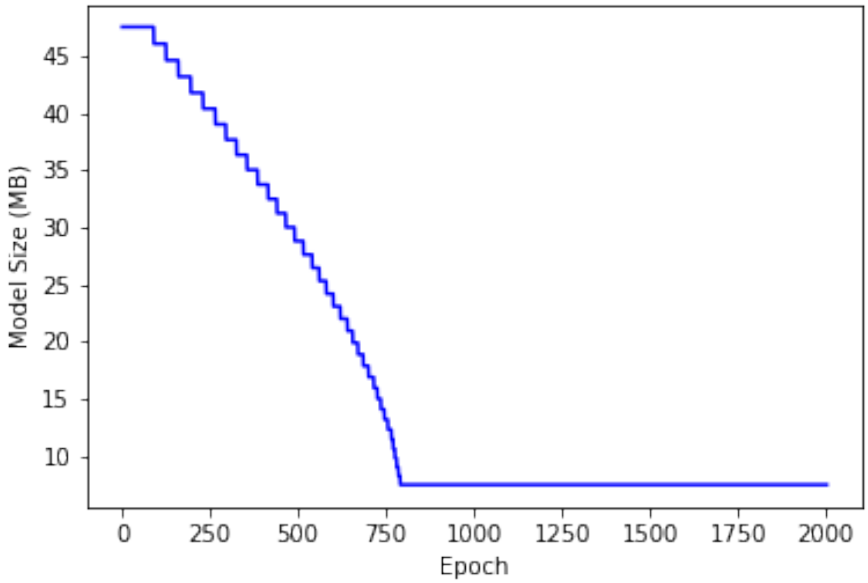


Figure 4.10: Model size while pruning in megabytes, constant once finished pruning.

We trained a model with the exact architecture that the pruned model finished with to see if there would be a difference in performance. We trained for 2000 epochs, with all other things being equal, except this time, we did not prune the model. This model achieved 0.878 test accuracy, which is extremely impressive given its size. This model did take around eight hours to train, which is less time than it took to train the pruned model. However, the pruned model did perform better

than the smaller model by about 0.02 or 2%, which is a significant difference when we get to this high of accuracy.

4.5 Pruning Cost

As models get pruned, the space they are searching changes due to the error surface changing as it loses some of the parameters it's trying to optimize. It takes time for the model to adjust to using the parameters that are left and searching the altered search space; this is what we call "recovery of the model". We expect the model's loss to be increased after pruning; the question is how long it takes to recover the added loss. Where the model prunes, we see jumps in the training loss compared to a non-pruned model (see Figure 4.11).

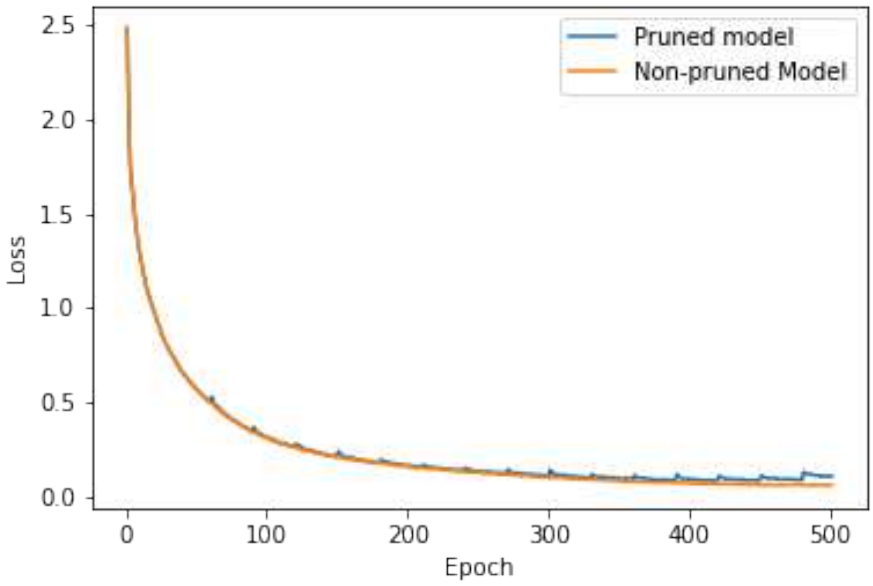


Figure 4.11: Training loss after 500 epochs, prune every 30 epochs and non-pruned model, (blue) is the pruned model, (orange) is the non-pruned model.

Zooming in (Figure 4.12), we get a better idea of how large the changes in loss are and how long it takes for the model to recover. The large peaks are when the model was pruned. Looking at the graph, it seems there is a varying number of epochs that it takes for the model to recover and

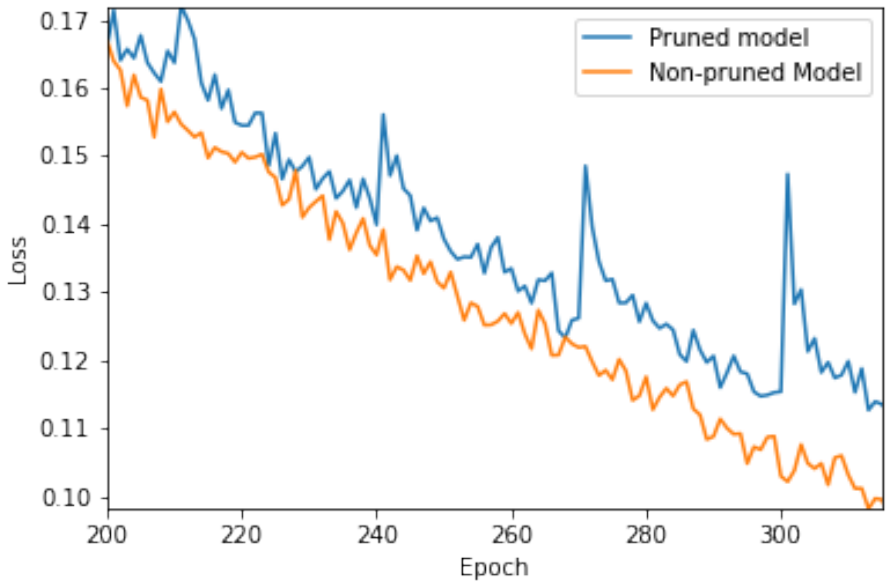


Figure 4.12: Training loss after 500 epochs, prune every 30 epochs, and non-pruned model, zoomed in to show more detail, (blue) the pruned model, (orange) the non-pruned model.

get back to the loss it achieved before the prune. In this instance it ranges between 5-20 epochs. We also see that the pruned model nearly always has a higher loss than the non-pruned model. In Figure 4.12, they have a similar loss; however, gradually, the gap between them becomes larger.

Looking at the cost of a single prune, we ran an experiment that ran for 45 epochs and pruned once at epoch 25, allowing it time to recover. We see that, on average, there is a cost to this pruning. (see Table 4.7). This is a marginal cost, but it would seem, based on past experiments, that the cost can get stacked together and magnify the effect if there is not ample time to recover.

Exploring if a model can recover fully given enough time, we trained three models, where one was not pruned, one was pruned every 30 epochs, and one was pruned every 100 epochs. Each was trained for 500 epochs. From Figure 4.13 we see that the faster pruned model does start to diverge. The gap seems to get larger closer to the 500th epoch. Examining Figure 4.14, We see that there is no divergence between the model's test accuracy; they are all tightly grouped together. This means

Table 4.7: Accuracy after one prune at epoch 25, total training: 45 epoch

Run	No Prune	Pruned
1	0.8291	0.8266
2	0.8368	0.8295
3	0.8336	0.8319
4	0.8279	0.8305
5	0.8302	0.8291
6	0.8339	0.8352
Avg.	0.8319	0.8305
σ	0.003	0.003

that even after a number of prunes, the pruned models are able to recover well and get to the same point as the non-pruned model.

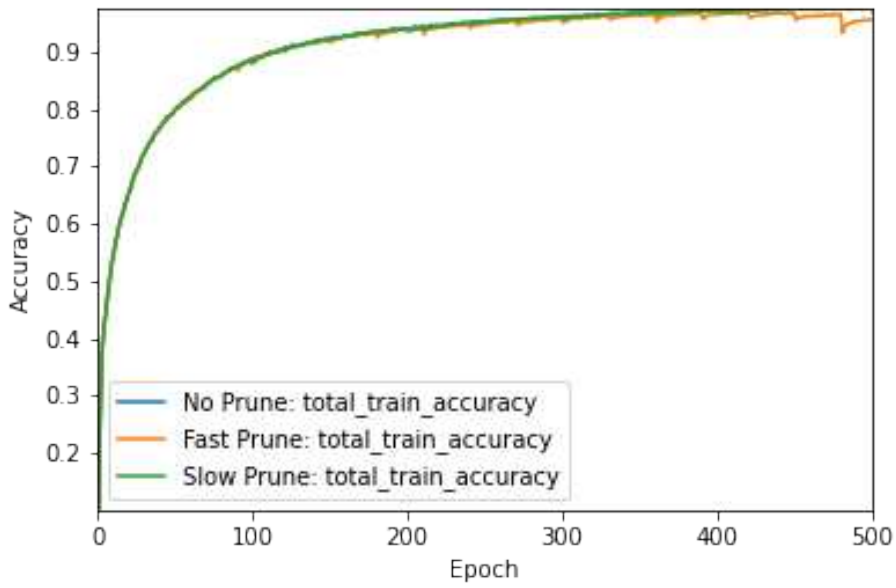


Figure 4.13: Training accuracy to show effects of fast pruning vs. slow pruning, ranged from epoch 0 - 500. Blue is the model not being pruned; orange is the model being pruned every 30 epochs, and green is the model being pruned every 100 epochs.

Figure 4.15 zooms in on the models toward the center of their training. It shows that the faster pruning model does diverge from the other models. Meanwhile, the slower pruning model does not seem to diverge. The fast-pruned model seems to start to diverge around epoch 300 and is very

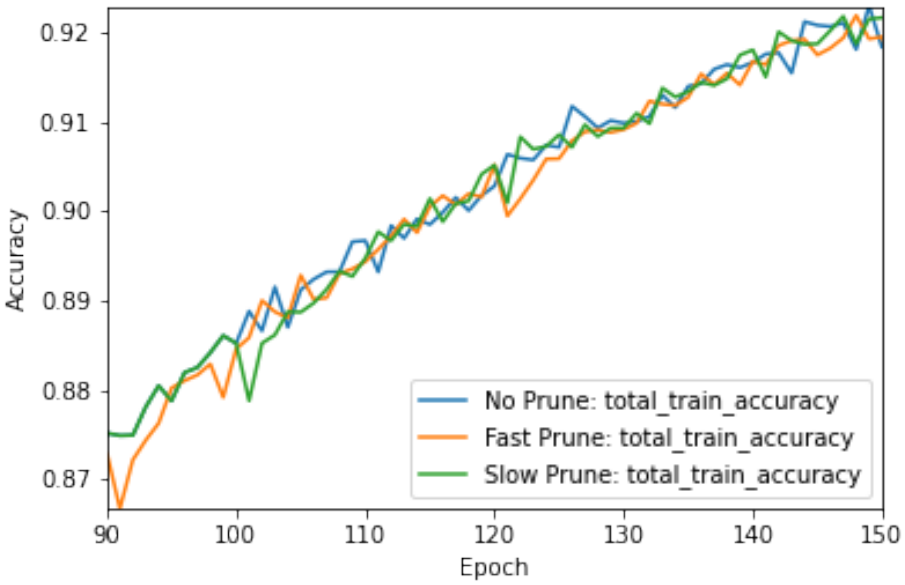


Figure 4.14: Training accuracy to show effects of fast pruning vs. slow pruning, focusing on epochs 90 - 150. This shows that there is not a large divergence between models at the beginning of training. Blue is the model not being pruned; orange is the model being pruned every 30 epochs, and green is the model being pruned every 100 epochs.

noticeable at epoch 350. This model struggles to recover, and its performance starts to suffer. These experiments showed at the beginning of training the fast pruning model pruned slowly enough for the model to recover. From these results, we hypothesize that the required recovery period is not necessarily constant but grows longer as the model becomes more pruned. We speculate this is because the model weights have to "carry more of the burden" of learning when there are fewer of them, so removing them means more knowledge is lost. This leads to a longer recovery period.

We can see an example of this, where we dynamically slowed down the pruning frequency while training in Figure 4.16. We allowed the model increasing time to recover, we trained it on the pruning schedule of pruning on epochs: 30, 70, 120, 180, 250, 370. We see the dynamic pruning schedule does perform just as well as the slower pruned, and the non-pruned model. With this dynamic pruning frequency, we were able to prune more times and get the model down to 19% pruned, whereas the slowly pruned model was only able to get to 12%. This was done while

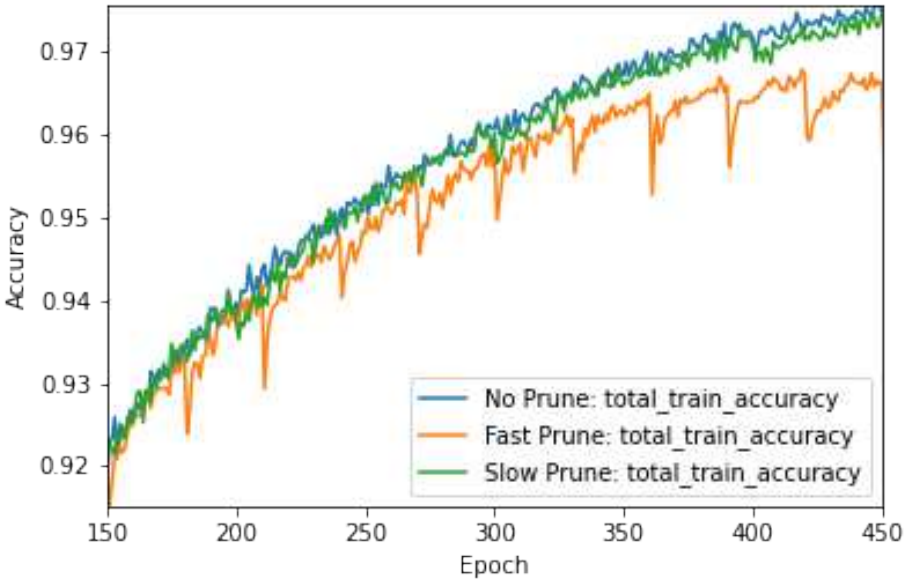


Figure 4.15: A graph that shows training accuracy to highlight the effects of fast pruning vs. slow pruning, focusing on epochs 150 - 450. This showed that the divergence between models starts around epoch 300 and picks up at 350. Blue is a model not being pruned, orange is the model being pruned every 30 epochs, and green is the model being pruned every 100 epochs.

also increasing test performance from 0.8951 on the slow-pruned model to 0.8961 on the dynamic-pruned model.

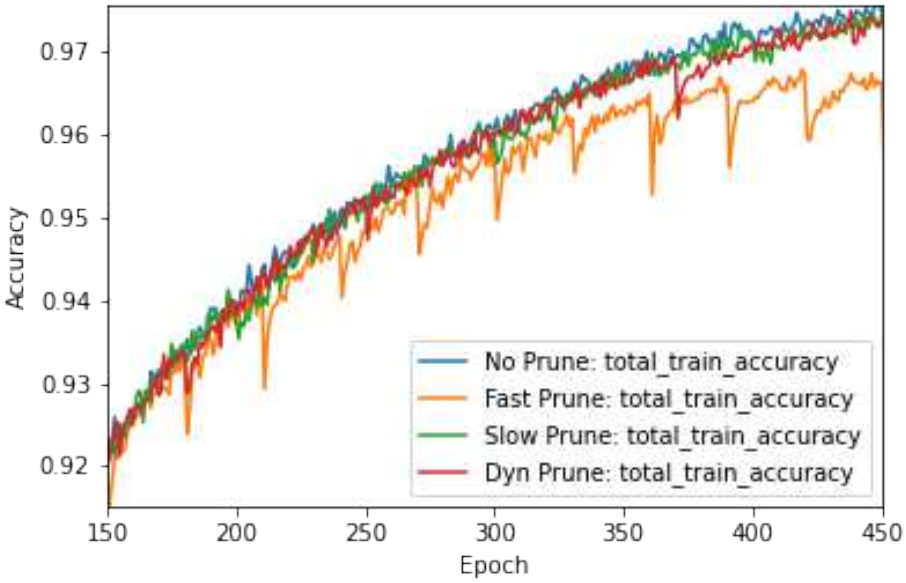


Figure 4.16: A graph that shows training accuracy to highlight dynamically slowing down the pruning frequency, focusing on epochs 150 - 450. Blue is a model not being pruned, orange is the model being pruned every 30 epochs, and green is the model being pruned every 100 epochs. Red is pruned on epochs 30, 70, 120, 180, 250, 370.

Chapter 5

Conclusion

5.1 Summary

Overall, this work shows that it is possible to prune a visual transformer that is being trained from scratch. We showed that pruning is an effective method of decreasing the inference time while maintaining accuracy, and this can be done from the very start of training, not just as a fine-tuning effort. We explored different training techniques to verify and test the model; this ranged from the generic training per epoch to training for a set amount of time or until we hit a certain accuracy.

We found that for epoch training, our pruning gave a significant speed up in the training time while also keeping a similar accuracy as the non-pruned model and having a smaller model. We found that when training for a restrictive fixed amount of time, the pruning effectively reduced the model size while keeping similar, if not better, performance as the non-pruned model. The non-pruned models performed slightly better if the allotted time was not restrictive. When training for a certain accuracy, we found that there was a tie between the non-pruned model and the pruned model. It was a toss-up who won or got better accuracy.

We also showed that there is a trade-off between how fast we prune and how compressed we can make the model. We can prune quickly to get faster training, but it will not converge to a higher accuracy, whereas if it was trained slower and pruned slower, it can be significantly compressed.

We explored the cost of a prune and how long the prune takes to recover. We showed that pruning more slowly leads the model to keep pace with the non-pruned model for longer, whereas pruning faster leads to more rapid divergence. We also explored slowing down the rate of pruning as the model got more dense, showing that it is an effective way to manage the dynamic recovery time as the model gets more pruned. We ended up with a smaller, more accurate model, than with just slow pruning.

5.2 Open Questions

Working on this research leaves some open questions. For example, are there more sophisticated ways of pruning that would lower the recovery time, perhaps removing neurons or dimensions that the attention mechanism deems unimportant across all samples rather than randomly? Is there a way to use the activation of the neuron in an intelligent way rather than just looking at the dead neuron? Is there a way to get a better pruning frequency? For example, is there a dynamic algorithm that can tell us when we should prune, based on the density of the model, that can lead to better results?

Bibliography

[1] Guillaume Chassagnon, Maria Vakalopolou, Nikos Paragios, and Marie-Pierre Revel. Deep learning: definition and perspectives for thoracic imaging. *European radiology*, 30:2021–2030, 2020.

[2] Alon Brutzkus and Amir Globerson. Why do larger models generalize better? a theoretical perspective via the xor problem. In *International Conference on Machine Learning*, pages 822–830. PMLR, 2019.

[3] Tianyu Wu, Shizhu He, Jingping Liu, Siqi Sun, Kang Liu, Qing-Long Han, and Yang Tang. A brief overview of chatgpt: The history, status quo and potential future development. *IEEE/CAA Journal of Automatica Sinica*, 10(5):1122–1136, 2023.

[4] Samsung. Galaxy s24 ultra, Jan 2024.

[5] Payal Dhar. The carbon impact of artificial intelligence. *Nat. Mach. Intell.*, 2(8):423–425, 2020.

[6] Guglielmo Tamburrini. The ai carbon footprint and responsibilities of ai scientists. *Philosophies*, 7(1):4, 2022.

[7] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. In *International Conference on Learning Representations*, 2021.

[8] Alex Renda, Jonathan Frankle, and Michael Carbin. Comparing fine-tuning and rewinding in neural network pruning. In *International Conference on Learning Representations*, 2020.

[9] Marco Zullich, Eric Medvet, Felice Andrea Pellegrino, and Alessio Ansuini. Speeding-up pruning for artificial neural networks: Introducing accelerated iterative magnitude pruning.

In *2020 25th International Conference on Pattern Recognition (ICPR)*, pages 3868–3875, 2021.

[10] Isac Arnekvist, J Frederico Carvalho, Danica Kragic, and Johannes A Stork. The effect of target normalization and momentum on dying relu. *arXiv preprint arXiv:2005.06195*, 2020.

[11] Jiaqi Zhang, Xiangru Chen, Mingcong Song, and Tao Li. Eager pruning: Algorithm and architecture support for fast training of deep neural networks. In *2019 ACM/IEEE 46th Annual International Symposium on Computer Architecture (ISCA)*, pages 292–303. IEEE, 2019.

[12] Yaohui Cai, Weizhe Hua, Hongzheng Chen, G Edward Suh, Christopher De Sa, and Zhiru Zhang. Structured pruning is all you need for pruning cnns at initialization. *arXiv preprint arXiv:2203.02549*, 2022.

[13] Victor Sanh, Thomas Wolf, and Alexander Rush. Movement pruning: Adaptive sparsity by fine-tuning. *Advances in Neural Information Processing Systems*, 33:20378–20389, 2020.

[14] François Lagunas, Ella Charlaix, Victor Sanh, and Alexander M Rush. Block pruning for faster transformers. *arXiv preprint arXiv:2109.04838*, 2021.

[15] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[16] Bingbing Li, Zhenglun Kong, Tianyun Zhang, Ji Li, Zhengang Li, Hang Liu, and Caiwen Ding. Efficient transformer-based large scale language representations using hardware-friendly block structured pruning. *arXiv preprint arXiv:2009.08065*, 2020.

[17] Torsten Hoefler, Dan Alistarh, Tal Ben-Nun, Nikoli Dryden, and Alexandra Peste. Sparsity in deep learning: pruning and growth for efficient inference and training in neural networks. *J. Mach. Learn. Res.*, 22(1), jan 2021.

[18]

[19] Omiita. Vit-cifar. https://github.com/omihub777/ViT-CIFAR, 2023.

[20] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009.

[21] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[22] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017.