To prune or not to prune: Exploring the effects of nodes in neural networks

Master's Thesis Autumn 2020

Lucas Georges Gabriel Charpentier

Thesis submitted for the degree of
Master in Computational Science
(Imaging and Biomedical Computing)
60 credits

Department of Informatics
Department of Physics
Faculty of mathematics and natural sciences

UNIVERSITY OF OSLO

Autumn 2020

To prune or not to prune: Exploring the effects of nodes in neural networks

Master's Thesis Autumn 2020

Lucas Georges Gabriel Charpentier

To prune or not to prune: Exploring the effects of nodes in neural networks

http://www.duo.uio.no/

Printed: Reprosentralen, University of Oslo

Abstract

Over the past decade, computational power has become more accessible, as hardware previously only found in research or military computers has become mainstream. These improvements have led to a vastly increased speed in repetitive calculations, allowing for the widespread adoption of machine learning and, especially, of deep neural networks. Whereas previously large neural networks were impossible to use due to limited memory and computer power, they have now become commonplace in many areas. But, with the increase in popularity of deep neural networks came a need to get a better grasp on their internal behavior. As such, research has deepened the understanding of layers that comprise these neural networks. Researchers now not only rely on fully connected layers but also locally connected layers or temporally connected layers. Further, they solved the mystery of deeper models becoming less performant, with residual layers. However, even though our understanding of neural network layers has considerably improved, our understanding of the components of these layers, the nodes, is still lacking.

In this thesis, we focus on understanding how individual nodes contribute to a neural network's performance. To this end, we try to classify them using a metric that connects the individual nodes to the loss of the model. Furthermore, we also analyze how removing nodes can affect the neural network. Our underlying assumption is that not all nodes contribute to the neural network's performance. We believe that some nodes are redundant and that these nodes both do not actively contribute to the performance of the model and increase its needed resources. By removing them, we reduce wasted resources and speed up the processing time of the model at the training and inference stage.

Our results show that not all nodes contribute to a model's performance. More specifically, some nodes are redundant as they extract the same features as others. Others negatively impact the performance as a whole, while some initially contribute positively to the network, but actually, only correct errors initiated by other nodes. Another important result is the increased stability between models from which we remove nodes. In other words, models of the same architecture trained on the same dataset, but with different initial weights, have a noticeably higher variation in performance before and after node pruning. This can be seen with the pruned convolutional neural network having a 5% reduction in its loss on average and a 19% lower loss variation. While this study makes no claim to provide a full understanding of how nodes impact neural networks, it does suggest promising paths for future works that could enhance our understanding of neural networks.

Acknowledgments

I would like to thank my official supervisors Michael, Pål, Steven and Vajira. Their good advice and feedback helped shape and write this thesis. I am very grateful for the weekly meetings over the summer and autumn especially during at times when the coronavirus required all meetings to be virtual. I feel lucky to have you as supervisors and this thesis would not have been possible without you.

I would also like to thank my parents, Lilian and Bastien as well as my sister Julia. Lilian pushed me hard to work on and complete my thesis, with delicious meals as a carrot during a summer of confinement in Paris. Finally, my dad gave me the resources to complete my work faster in terms of computations. The walks in the forest with my sister and mother helped me decompress and relax by providing a welcomed breath of fresh air in between between confinement spells. Finally, I would like to thank my dad and my godfather Thierry for their feedback on my thesis, especially on the abstract, introduction, and conclusion. Their point of view as outsiders to the field was most helpful.

Finally, I would also like to thank two of my friends. First, Sean for motivating me during these times. Second, my roommate Ådne for motivating me to keep working on this thesis every day while giving me confidence that I would soon finish it.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background and Motivation

Machine learning has increased in popularity over the past few years, with rapid growth in the number of methods and techniques used. It has also become much more common to see machine learning being applied in research and industry. This is not the first time that machine learning (or artificial intelligence) has become popular. The base of most machine learning algorithms is the Perceptron, based on the McCulloch-Pitts Neuron [16] and later enhanced by Rosenblatt [21] in 1958. However, computing power back then was scarce and more expensive, both to produce and to run, especially in terms of cooling. Indeed, aspiring computer engineers were taught to calculate the amount of salt required to cool the computer running their programs, to get an estimate of the cost of running an algorithm. Another major hindrance in the early days of computing was memory which occupied significant space for limited capacity (at most a few Mega-Bytes (MB) rather than today's Tera-Bytes (TB)). Seeing that most machine learning algorithms rely on having large amounts of data, not having an easy way to access this data digitally proved a major hurdle. A final bottleneck was the amount of random-access memory (RAM) available by computers. RAM is used to run programs and store the information used by the program while running. Back in those days, this memory did not exceed a few kilo-bits (kb). However, looking at modern neural networks used in machine learning, we see that they would not be able to run due to running out of memory space. This all led to the hype built by the possibilities of machine learning to die down.

Nowadays, with RAM reaching the TB, large-capacity storage memory becoming physically smaller, and computing power being more readily available at a much-reduced price, machine learning has had the opportunity to flourish and evolve. It started with the Perceptron, then became Multi-layer Perceptrons or Artificial Neural Networks. Those then branched out to different types of neural networks such as Convolutional Neural Networks, Recurrent Neural Networks, and others. We also started to do machine learning with different methods, such as unsupervised

learning, with unlabelled data, or reinforcement learning where we "teach" the computer to imitate or even surpass humans by using a reward system.

During this evolution, we kept making neural networks bigger and more complex. This is especially true in terms of how deep (number of hidden layers) neural networks have become. However, neural networks reached a point where deeper neural networks performed worst than their shallower counterparts. This, however, led to a contradiction since we could prove mathematically that at worst, an extra layer should act as an identity layer and therefore leave the performance of the model unchanged rather deteriorating it. This problem was then solved by residual layers [9], which showed that this contradiction came from layers 'forgetting' what was learned by previous layers.

While the problem of increasing layers in a model has been solved, we have not heavily explored the number of nodes needed in each layer, or how the number of nodes in a model and in each layer affects the results. Therefore, in this thesis, we will explore how nodes affect the network and whether all the nodes contribute to the neural network or if some of them either have no effect or even a negative one. Moreover, we will explore how changing the batch size or adding dropout affects the node's importance of the neural networks. By gaining a better understanding of the nodes, we will be able to remove them. This could lead to improved model performance as well as reducing waste of resources in terms of memory or computing power.

1.2 Problem Statement

As stated at the end of the above section, our goal in this thesis is to explore the effects of nodes in neural networks. To achieve this, we will prune (i.e. remove progressively nodes from) our neural networks and see how their performance changes after pruning. We will also categorize the nodes based on their usefulness to the network. This will provide us an initial idea of how nodes contribute to neural networks and if they do all contribute, or if some have little or a negative impact on the model's performance.

The question we are trying to answer in this thesis is the following;

> *How do the nodes in a neural network contribute to the overall performance?*

Our research question is motivated by our hypothesis that some nodes might be redundant and, therefore, do not contribute to the performance of the model. With this in mind, we also want to investigate how different neural networks with different depth and layer types are affected when pruning them. To better answer our research question, we split our work into three objectives. These objectives will help us navigate the different facets of our question.

- **Objective 1:** Exploring the effects of pruning neural networks and developing different pruning techniques for both reproducibility and time-consumption.

- **Objective 2:** Classifying the nodes into different classes based on how they affect the neural network if removed.
- **Objective 3:** Seeing how different parameters and types of neural networks affect the nodes impact on neural networks.

1.3 Scope and Limitations

This thesis is concerned with fully connected neural networks and convolutional neural networks. For this exploratory investigation on node importance of nodes in neural networks, we will only look at how a single hyper-parameter (batch size during training) affects the node importance of nodes and a technique applied to the layers (Dropout [26]) affect the node importance of nodes. Limited computation power and time constraints prevented us from exploring different hyper-parameter. In the pruning section, we only test a decreasing number of networks as their complexity increases. This is again due to the time it takes to prune a single network, and at times, the lack of graphics card memory available and memory corruption, making the testing fail halfway through. The same reasons also led us to look into faster ways to prune the networks. Finally, we also look at a more complex network. However, we will only do this once since the time it takes to prune a large complex network is consequential because we have not yet found a way to parallelize the pruning algorithm, and the limited time available.

For the scope, we consider four different datasets, MNIST [13], Fashion MNIST [28], CIFAR-10 [12], and Kvasir [19]. The first three were chosen because they are popular datasets when exploring neural networks and each is more complex than the previous. The last dataset is employed for a more real-world example investigation, when we also consider a substantially more complex neural network (a VGG-16 model [24]).

1.4 Research Method

While there are various ways to conduct research, we followed the method developed in 1989 by a task force assembled by the Association for Computing Machinery Education Board. Their task was to define the core ideas of computer science and computer engineering in a detailed report. Its article "Computing as a discipline" [2] splits the discipline into three paradigms:

1. theory;
2. abstraction;
3. design.

This thesis's research is conducted in compliance with this methodology and we now describe how we adhere to each paradigm.

1.4.1 Theory

The theory paradigm is based on mathematics. It consists of four steps:

1. characterize objects of study (definition);
2. hypothesize possible relationships among them (theorem);
3. determine whether the relationships are true (proof);
4. interpret results.

These four steps help us in the development of a coherent, valid theory.

We follow this paradigm by analyzing how different methods of node pruning affect the reduction in size and performance of the pruned neural networks. We also measure how changing certain parameters changes the importance of nodes in neural networks. Specifically, the presence of dropout and the size of the batch size during training. Finally, we analyze how pruning affects differently various types of neural networks.

1.4.2 Abstraction

The abstraction paradigm is based on the experimental scientific method and consists in four stages:

1. form a hypothesis;
2. construct a model and make a prediction;
3. design an experiment and collect data;
4. analyze results.

These four stages help us in the investigation of a phenomenon.

Our experiments follow this paradigm. We start by hypothesizing that removing nodes from a network can affect it. We then test this out, based on the results obtain, we refine this hypothesis, and test out new corollaries. Finally, we take all our results, form a more refined hypothesis and test it out on a more practical experiment to see if the hypothesis holds in the practical realm.

1.4.3 Design

The design paradigm is rooted in engineering. It consists of four steps:

1. state requirements;
2. state specifications;
3. design and implement the system;
4. test the system.

These four steps describe the construction of a system to solve a given problem.

We follow this paradigm by constructing different pruning algorithms. We start by pruning randomly before turning to a metric to decide whether or not to prune a node. For each pruning algorithm, we specify what is needed (be the number of nodes to prune or a method to select which node to prune) and evaluate the algorithm before either refining it or using it as our final algorithm.

1.5 Main Contributions

This thesis explores how nodes and their removal affect neural networks. To guide our exploration, we split our research questions into three different objectives, couched as follows:

Objective 1: *Exploring the effects of pruning neural networks and developing different pruning techniques for both reproducibility and time-consumption.*

Through the thesis, we will explore different facets of this objective. We will go from a random pruning technique (which is hardly reproducible) to one that works based on unchanging node parameters and metrics but is rather time-consuming. Finally we settle for a technique that keeps the reproducibility of the latter but reduces its time-consumption by at least a factor of two. We will also investigate how different parameters are affected by pruning and will be able to conclude that pruning a model has a noticeable impact on neural networks by generally reducing the loss, or in some cases, not changing it or slightly improving it.

Objective 2: *Classifying the nodes into different classes based on how they affect the neural network if removed.*

To this end we will develop a metric, called "node importance", to classify these nodes based on their effect on the loss of the model. The node importance is as the change in the loss of the model generated by this removal of a node. If the loss of the model increases, then the node is classified as an important node. If the loss does not change or the change is insignificant, then the node is considered to be a zero node. Finally, if the removal of the node decreases the loss, then it falls in the worse node category. To avoid pruning nodes randomly, we iteratively remove worse and zero nodes from the network till all the nodes left are important. We will do this both in an "exhaustive" fashion (finding the node with the highest node importance value and removing it) and in a "greedy" fashion (removing the first zero or worse node we come across).

Objective 3: *Seeing how different parameters and types of neural networks affect the nodes impact on neural networks.*

This thesis will use three different types of neural networks to explore their reaction to pruning. We will also modify one hyper-parameter (batch size during training) and see how the inclusion of dropout to our purely fully connected networks affects performance.

We will be able to achieve our three objectives and form hypotheses on how pruning affects neural networks. A metric based exclusively on the loss of the model, arguably the most important metric of model performance, provides a way to evaluate whether an individual node can impact the model positively (important nodes) and negatively (worse nodes). This will lead us to the hypothesis that some nodes are redundant or even at times unhelpful for the model overall. To be able to strengthen this hypothesis further, we should continue exploring these effects by broadening our metric to how nodes impact other model metrics such as accuracy and precision, etc. Further, looking at the individual effects of nodes is a good starting point in giving us intuition on the impact of nodes on the model. However, looking at how one node impacts another node, would help us better understand how nodes interact with each other and ultimately how these interactions impact the neural network as a whole.

This thesis we will show that pruning frequently has a positive effect. In most cases, the model loss will diminish and the accuracy will either stay similar or will increase slightly. Even though the model loss will not change much for the larger network, we will be able to significantly reduce the size of our network (reduction in excess of 35% in the number of filters and nodes). This in itself is a positive since using a much smaller network will require less memory and computation. This will also lead us to a new hypothesis where a network, through pruning, could be overfitted to a different dataset from the one it was trained on. Further researching this new path could lead us to a better understanding of neural networks. Moreover, by logging the evolution of the model loss through the pruning process, we will see that the loss of our model did start by decreasing before increasing due to overfitting. This will help us ascertain our views that not all nodes in a network directly benefit it.

Our thesis will start to answer our research question on how nodes contribute to the overall performance of neural networks. As mentioned in the two previous paragraphs, we will see and hypothesize some of the effects of nodes on neural networks. We will also show that removing nodes frequently improve our models. These experiments have open new paths into better understanding neural networks and specifically their base element, the node. Continuing the exploration of these previously unknown new paths could lead us to better answer our research question and truly understand the impact of nodes on neural networks.

The code for this thesis can be found at https://github.com/lgcharpe/Masters (MIT License).

1.6 Thesis Outline

The rest of this thesis is structured as follows:

- **chapter 2: Background**: We will provide background knowledge on the techniques and methods employed in this thesis.

- **chapter 3: Methodology**: We will explain our methodology. We will describe the three different neural networks we will use. We will then describe the main metric, node important, employed to gauge the node's usefulness. After that, we will explain the different ways we prune our networks. From randomly pruning the networks to using the node importance to decide which nodes to remove. Finally, we will describe all the algorithms we programmed to prune networks and estimate the node importance.

- **chapter 4: Exploring node pruning and node importance in simple neural networks**: We will explore how we estimate the node importance and we will see how two different hyper-parameters affect it. We will also evaluate different pruning techniques and their effectiveness at ameliorating the performance of networks. Finally, we will try to use node importance to improve initial conditions. In section 4.1, we will prune the networks randomly and analyze how effective it is. In section 3.2, we will introduce our node importance metric and classify nodes into three different importance categories. In section 4.3 and section 4.4 we will analyze how changing the batch size during training and adding dropout to the fully connected networks, changes the node importance of the neural networks. In the section 4.5, section 4.6, and section 4.7, we will test pruning our networks based on the node importance. We will start by pruning the networks based on pre-calculated node importance. We will then move to pruning iteratively, calculating the node importance of the nodes, removing a node, and restarting till the networks are fully pruned. Finally, we will suggest another technique to prune networks similar to the previous one but has for objective to be faster than the previous one. In all the sections we will also explore whether pruning on the validation set instead of the training set makes a difference. In section 4.8, we will look at how pruning affects the class accuracy of the networks. We will also analyze how it changes the regular accuracy and loss. In section 4.9, we will attempt to use node importance to improve the initial weights before training.

- **chapter 5: Case study: Reducing a VGG-16 model trained on the Kvasir dataset**: The techniques developed in the previous chapter are used on a more complex neural network (VGG-16) trained on a more complex dataset (Kvasir) to estimate the node importance and prune the network. We will then evaluate how effective the node pruning is and how it affects accuracy, class accuracy, and loss.

- **chapter 6: Conclusion**: We will sum up all of our results and draw conclusions from them. We will also suggest topics of interest for future research.

Chapter 2

Background

2.1 Machine Learning

In this chapter, we will give some background to machine learning techniques used nowadays and more specifically, techniques used in this thesis. For this reason, we will focus on supervised learning, specifically multilayer perceptron, and convolutional neural networks. These techniques have become very popular in both research and applications to real-world problems.

2.1.1 Supervised Learning

Supervised learning is used in a wide area of applications, from image recognition to text translation to stock predictions. To be able to do supervised learning, we need labeled data. We need the labels to be able to tune our algorithms to do its specified task. This can be loosely paralleled to teaching kids what an object is or how a word is pronounced. We let the algorithm tell us what it thinks the data is. If it is wrong, we update the weights in the direction of the correct label, while if it is correct, we do not change any weights. We do this till it converges or starts learning the noise of the data. Some examples of supervised learning are linear regression, logistic regression, and neural networks. In this project, we will only look at supervised learning, specifically neural networks.

2.1.2 Unsupervised Learning

Unsupervised learning is commonly used to separate data into clusters. Contrarily to supervised learning, the data fed to an unsupervised algorithm is not labeled. Therefore, we are not able to judge the performance of the model with respect to ground truth. Nonetheless, unsupervised learning is still very useful. Once the machine cluster the data into different groups, we can analyze these groups and try to determine similarities between items in the group. This is possible because these algorithms usually cluster data by how close they are to each other (where the distance between the data points is determined by a pre-defined metric).

2.2 Artificial Neural Networks

Artificial Neural Networks, also called multi-layer perceptrons, are the simplest form of neural networks. They contain one or more hidden layers of perceptrons which are all fully connected in between layers. This was the first type of neural network used to do supervised machine learning and is an extension of linear/logistic regression since they are neural networks with no hidden layers. These computational models were inspired by the neural networks found in the human brain. However, since their introduction, they have diverged from their biological roots and have become a staple in the search for optimizing machine learning results.

Three important features strongly affect the architecture of ANN. The two first are the number of hidden layers and the number of nodes in each of these layers. These, when increased, generally improve the performance of the model but also significantly increase computational resource usage both in the form of computing time and hardware requirements. The last feature to highly affect the architecture of ANNs is the activation function applied to each layer. It is used to introduce non-linearity to the model. If the activation functions were linear then we would in essence be performing an over-engineered linear regression since the layers could all be collapsed into a single layer. Therefore having the possibility for non-linearity will help us solve problems with non-linear decision boundaries that are otherwise unsolvable for a linear model.

ANNs were more common in the past being used for every type of application. However, we have since created more advanced types of neural networks to solve different tasks. For images, we now use convolutional neural networks, while natural language processing frequently makes use of recurrent neural networks.

2.2.1 Perceptron

The base element of an ANN is the perceptron. The perceptron is inspired by the McCulloch-Pitts Neuron [16], which is a simplified version of the human neuron. A M-P neuron takes inputs $(x_1, x_2, ..., x_n)$ that are then summed up. If the sum is bigger than a threshold, then the neuron 'fires' (outputs one) otherwise the neuron outputs zero. The perceptron has a few differences from an M-P neuron. First proposed by Rosenblatt [21], a perceptron works by taking inputs $(x_1, x_2, ..., x_n)$ multiplying them by set of weights $(w_1, w_2, ..., w_n)$ and then summing them up. As for the MP neuron, if the sum is bigger than a threshold, then the neuron 'fires' (outputs one) otherwise the neuron outputs zero. This is also shown in Figure 2.1.

$$f = \begin{cases} 0 & \text{if } \sum_{i=0}^{n} w_i x_i \leq 0 \\ 1 & \text{if } \sum_{i=0}^{n} w_i x_i > 0 \end{cases} \tag{2.1}$$

The identity function was replaced by different functions (named activation functions) in more recent years. Nowadays there is a wide range of different activation functions used in neural networks. The ones used in

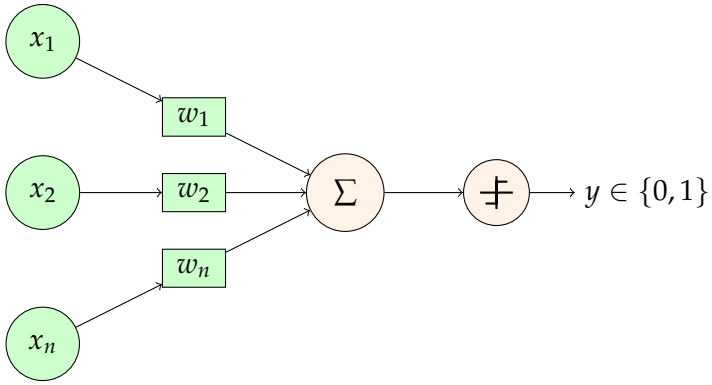

Figure 2.1: Perceptron

this paper are described in subsection 2.4.5. A bias term b was also added to the sum making Equation 2.1 become Equation 2.2.

$$f = \begin{cases} 0 & \text{if } \sum_{i=0}^{n} w_i x_i + b \leq 0 \\ 1 & \text{if } \sum_{i=0}^{n} w_i x_i + b > 0 \end{cases} \tag{2.2}$$

While excitement for the uses of the perceptron started high, they diminished with Minsky and Papert's paper [17] which showed that while the perceptron could replicate linearly separable behaviors such as AND and OR. If the behavior was not linearly separable, such as XOR, no amount of tuning could make a perceptron mimic it.

2.2.2 Multilayer Perceptron

The solution to the non-linearly separable problem (such as XOR) was to have a layer of perceptron between the inputs and the outputs. This layer became a hidden layer where every perceptron (now called a hidden node) had a bias and took all inputs in. They then all contributed to the output. However, each hidden node's contribution was weighted, with the weights being tuned during training in a similar fashion as the weights applied to the input nodes. This is also referred to as a feed-forward architecture, where each previous layer contributes to the next layer. A visual representation can be seen in Figure 2.2. By having multiple perceptrons together, we approximate any function as proven by the universal approximation theorem [3]. If we look at the connections between each layer, we can see that each node, whether it be a hidden, input, or output node, is fully connected to all the nodes in the previous and next layer when they exist. This is referred to as fully connected layers or dense layers.

2.2.3 Training a Neural Network

As mentioned in the two previous sections, the outputs passed by either the input nodes or hidden nodes are weighted. These weights have to be finely-tuned for the model to be performant. This procedure of finely-tuning the

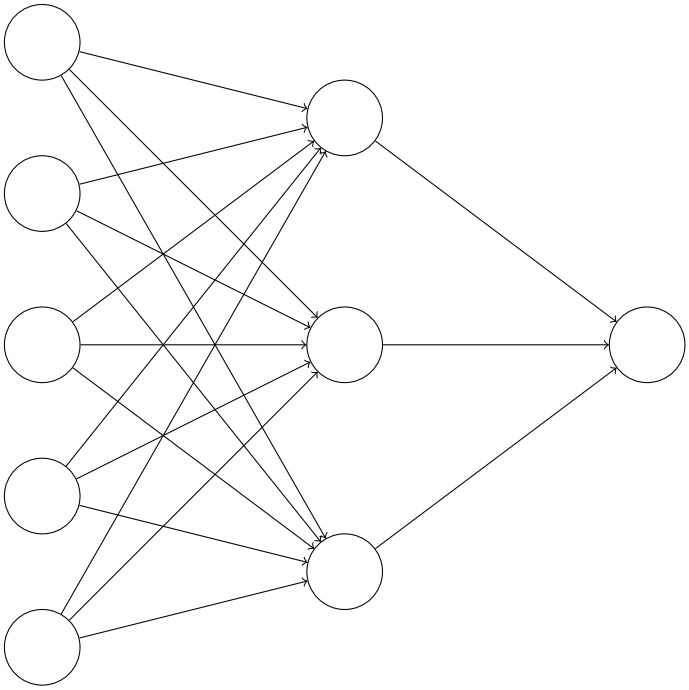

Figure 2.2: A multilayer perceptron with 5 input nodes and a hidden layer of 3 nodes.

weights of the model is known as training. A general explanation of how training is performed for each learning type is presented in section 2.1. Since this thesis focuses on supervised learning, we will describe more carefully its training procedure in this section. A mentioned before the main idea is to train our model using labeled data. This can be paralleled to how humans learn, however, our model will require a substantial amount of labeled data to perform similarly to humans.

The training of a model consists of three main components. The first is a loss function. Once we have obtained an output from our model, for a given input, we need to compare it to the ground troop. To be able to objectively compare them we need a function that compares our model's output to the ground truth. This function is called the loss function (also referred to as error function or cost function). Now that we can compare our model's output to the ground truth, we need to be able to pass those errors to the weights of the model. By passing the errors to the weights of the model, they will be able to learn from them and hopefully recognize patterns in the data such that when it encounters similar inputs it will be able to produce the correct output.

The method to pass the errors to the weights is our second training component and is called backpropagation [22], where the gradients of the loss function with respect to each weight are passed back to each layer. Since, the gradients dependent both on the loss function and weights, as we go back in the layers, we need to apply a series of chain rules since the layers learn both from the loss function and the layers that succeed

it. To avoid calculating gradients multiple times, we save the gradient calculations of a layer before moving to the next.

The final training component ties in directly with the backpropagation method since it determines how we apply the gradients to the weights such that they can learn from the ground truth. For this, we use an optimizer. The most basic optimizer used is gradient descent where we apply the calculated gradients to the weights. To not move the weights by too large of a margin, we usually multiply the gradients by a learning rate. While this is the simplest optimizer, it is not frequently used. In subsection 2.4.1, we describe the optimizers used in this thesis.

With these components, we can train our models to perform correctly on labeled data. The training procedure is performed a given number of times, where each time we complete training on the whole dataset being called an epoch. The parameters used in the optimizer, such as the learning rate, the optimizer used, the maximum number of epochs of training to do, and other parameters that can be altered are referred to as hyper-parameter. Fine-tuning these parameters is also important to achieve highly performant models. For example, if the learning rate is high, we might train faster but we run the risk of overshooting the global minimum and therefore not be able to train the model to its full potential. On the other hand, a learning rate that is too small risks getting stuck in a local minimum or taking very long to converge.

A final note on training the neural networks is how we prepare our data. To make sure that our neural network is not fitting the noise of the data we have instead of learning the general patterns of the data, we split our data into two or three different sets:

1. **Training set**: This is usually the largest set and the one we use to train the data. This is the set that the model learns from.

2. **Test set**: This set is unseen by the model, till the very end, where we use it to test our model performance. This set will tell us if our final model understands the general underlying pattern in the data or if it learns background noise from the training set.

3. **Validation set (optional)**: This set is usually found during the training set, it is used to validate whether the model is learning the general pattern or if it overfitting (learning the noise of the training set). If the loss on this set starts increasing while the loss on the training set decreases, then we are overfitting the model. In other words, the larger the difference between the loss of the validation and training set is, the more overfitted the model is.

Once we have separated our data into these two or three sets, then we are ready to start training our models.

2.3 Convolutional Neural Network

As mentioned in the previous section, MLPs have become less common depending on the problem needed to be solved by the machine. This is especially true for problems involving images. With all the layers in an MLP being fully connected, as the number of inputs increases the parameter space rapidly increases. This causes the search for the global minima of the loss function to become increasingly complex. This is called the curse of dimensionality [1].

Another weakness of MLPs is that they do not take into account the spatial structure of the input images directly. Since there is no sense of distance between nodes, whether a pixel is close to another or they are far apart, they will be treated similarly. Since images usually contain useful spatial information, and having a notion of those helps greatly in classifying an image. Therefore, not being able to learn from the spatial structure hinders an MLP greatly.

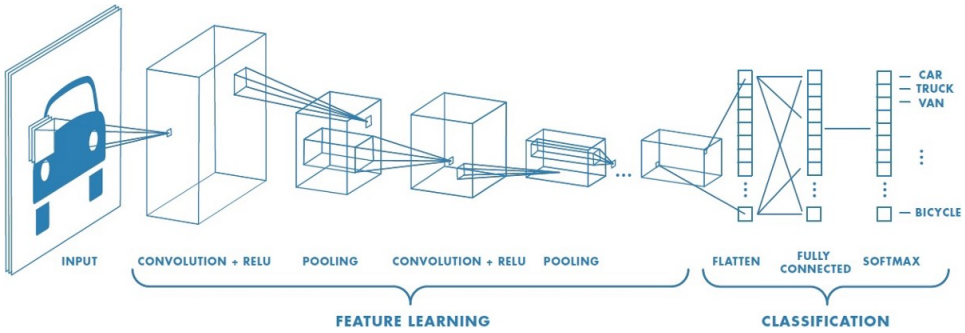

Figure 2.3: The typical structure of a CNN[1].

To solve these weaknesses, a new type of neural network was developed, a Convolutional Neural Network. Similarly to an MLP, a CNN is a feed-forward network, however, instead of all being fully connected between layers, it is instead locally connected between some layers. In other words, instead of a node in a hidden being connected to all the nodes in the previous layer, it is only connected to a small subset of nodes. By doing this, we greatly reduce the parameter space of the network, even if we have a deeper network. This also solves the problem of distance since the nodes only depend on nodes that are near each other in the previous layer.

The hidden layers of a CNN can be separated into two parts. The first is a series of convolutional and pooling layers (described in the next section). These layers take advantage of local connectivity and share weights to both reduce the total parameter space and extract spatial features from the images. The next part is a series of fully connected layers to classify the extracted spatial features and images. This is shown in Figure 2.3.

[1]Credit goes to author Mathworks.com: https://www.mathworks.com/solutions/deep-learning/convolutional-neural-network.html

2.3.1 Convolutional Layers

Convolutional layers are at the heart of convolutional neural networks. Not only do they do most of the work, but they are also the main reason CNNs are so effective on image data. The idea for these layers comes from image analysis, specifically, convolutions, such as the Sobel operator [25] used for edge detection or Gaussian blur, where a Gaussian function is applied as a filter to an image, used to smooth images, where we could extract features from images that help us classify.

The basic functioning of a convolutional layer is very similar to convolutions. We have a filter, usually of small square dimensions such as $3 \times 3 \times D$ or $5 \times 5 \times D$ (where D represents the number of channels (depth) the image contains, for example, RGB images have three channels; R, G, and B, while grayscale images only have one channel), we then slide this filter across the whole image, this gives us a new 2-dimensional image. The sliding consists of an element-wise multiplication of the values in the filter with the image values the filter is currently covering. Figure 2.4 gives a visual representation of a convolution. However, instead of defining the convolution function before, convolutional layers start with a set of filters with random weights that are then fine-tuned by training those weights with the ground truth label of the data.

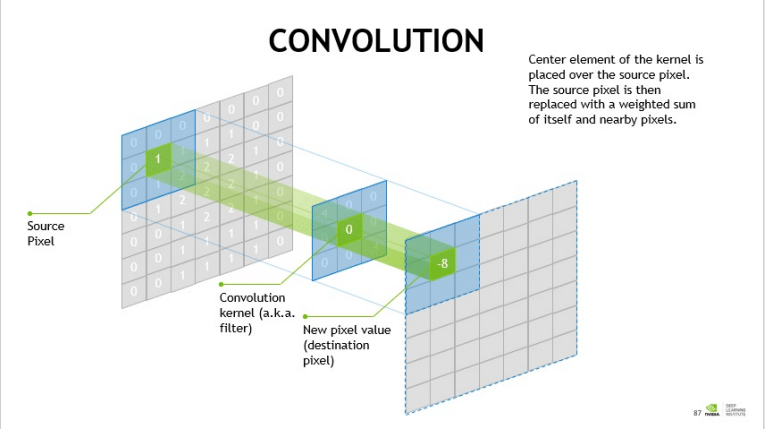

Figure 2.4: A convolution with a 3 by 3 filter[2].

Another difference to normal convolutions is that in convolutional layers, we use multiple filters for each layer. Once each of these convolutions is executed the resulting two-dimensional images are stacked together to make a three-dimensional image with the number of filters being the depth of the image. This is visually represented by Figure 2.5.

Using these convolutional layer provides two big advantages:

1. Local Connectivity

[2]Credit goes to author nVidia: https://blogs.nvidia.com/blog/2018/09/05/whats-the-difference-between-a-cnn-and-an-rnn/

[3]Credit goes to author Brilliant.org: https://brilliant.org/wiki/convolutional-neural-network/

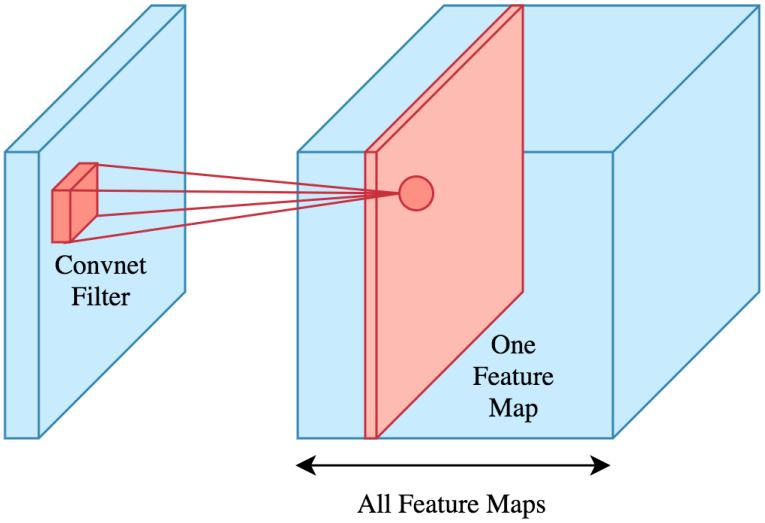

Figure 2.5: A convolutional filter from a convolutional layer gives one of the depth layer of the output layer[3].

2. Parameter Sharing

As mentioned before, when all nodes are connected, we end up having no explicit distinction between nodes that are close or far spatially. This is a problem especially in images where neighboring pixels unusually share similar features. Further, by using local connectivity instead of full connectivity, we dramatically reduce the number of total connections and therefore parameters in the model. This both reduces our risk of running into high-dimensionality problems and increase our training speed.

Parameter sharing is another big positive to using convolutional layers. This again reduces the total number of parameters. By sharing parameters we assume that a particular feature found in one place can also be found in different parts of the image. Both of the advantages together, help the network classify images into the same class even if the image is manipulated (shifted, rotated, etc.). With the help of local connections and shared parameters, the positioning, rotation, or even the color of an object/animal does not influence the final classification as much.

In practice to create these convolutional layers we need to set four parameters:

1. Number of filters (K);
2. Filter dimensions (F) (in terms of width and length since depth is always equal to the depth of the image);
3. Stride (S): the number of pixels moved between each convolution calculation. In other words, if we have a stride of one, we slide our filter one pixel at a time, while a stride of two skips every other pixel.
4. Amount of zero padding (P): the number of zeros to add to each direction (width/length). By zero padding, we can decide on the

output image size.

Once we have these four parameters, then assuming that our initial image has dimensions $W_1 \times H_1 \times D_1$. Our output image will be of dimensions:

- $W_2 = \frac{W_1 - F + 2P}{S} + 1$
- $H_2 = \frac{H_1 - F + 2P}{S} + 1$
- $D_2 = K$

2.3.2 Pooling Layers

In addition to the convolutional layer, CNNs commonly also include pooling layers. These have for function to reduce both the spatial size of the outputs and reducing the total number of parameters. This speeds computation and helps us control overfitting. Usually, we follow either a convolutional layer or a series (normally no longer than three) of convolutional layers by a pooling layer. The most common type of pooling layer is a max-pooling layer of size two and stride two. A max-pooling layer returns the maximum of the values considered in the window. Figure 2.6 shows how a pooling layer works, and Figure 2.7 shows how the max pooling operation works.

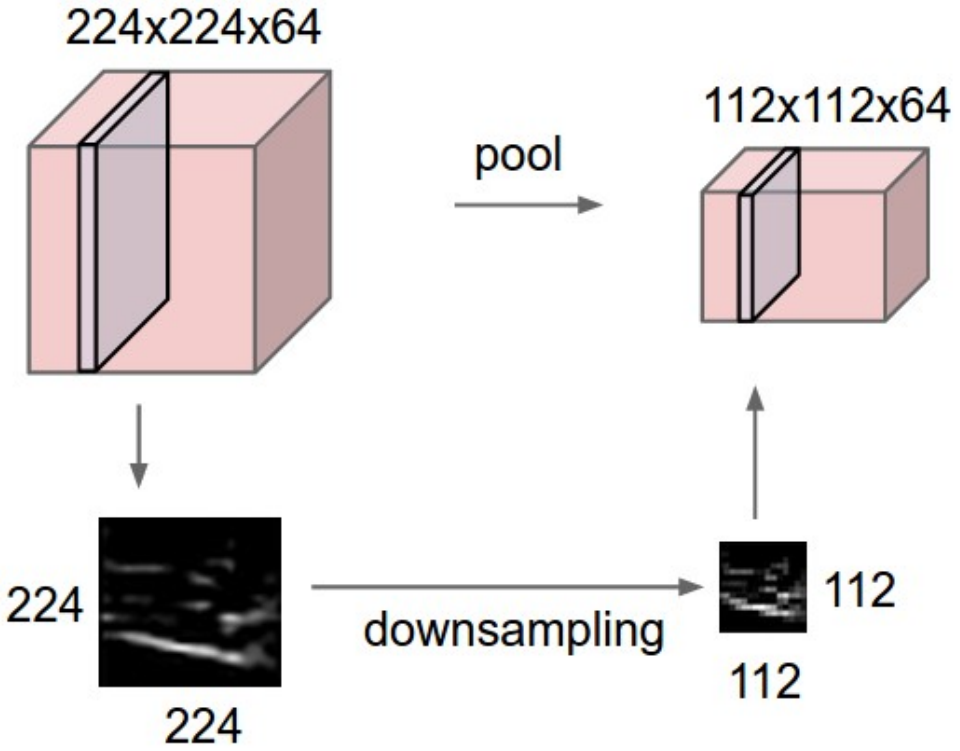

Figure 2.6: A pooling layer of size 2 by 2[4].

[4]Credit goes to author CS231n: https://cs231n.github.io/convolutional-networks/

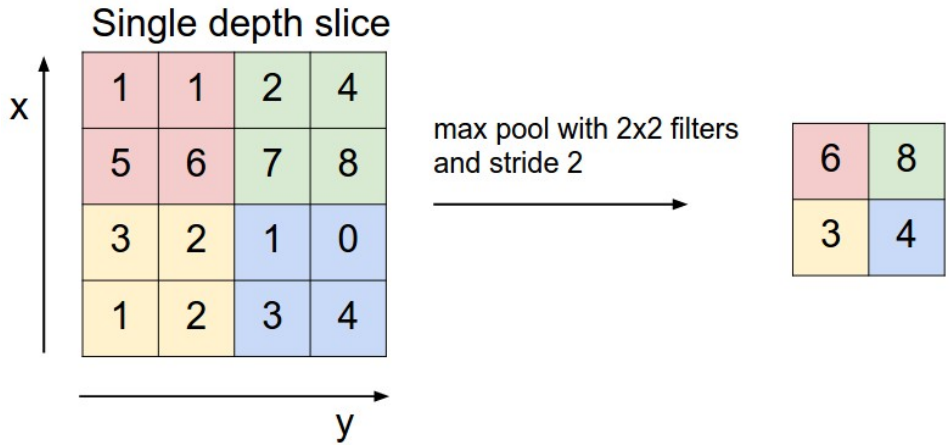

Figure 2.7: A max pooling operation of size 2 by 2[4].

2.4 Neural Network Training Optimization

As mentioned in subsection 2.2.3, adjusting hyper-parameters is one way to optimize the training procedure for neural networks. However, this is not the only technique, we can also optimize our weight initialization, or modify both the feed-forward and backpropagation algorithms.

2.4.1 Optimizers

As mentioned previously, while gradient descent is the simplest optimizer techniques, it is rarely used. For machine learning, more refined optimizers are usually used.

Stochastic Gradient Descent

A variation of gradient stochastic gradient descent. The basic idea for this method comes from the Robbins-Monro algorithm [20]. Instead of doing gradient descent on the whole dataset, we do gradient descent by batches. In other words, during training, we shuffle and divide the training set into batches of user-defined sizes. We then feed-forward a batch before doing backpropagation on that batch. An epoch, in this case, represents multiple feed-forward and backpropagation passes till the whole training set has been passed to the model. By splitting our dataset into multiple batches we reduce the computational burden for each backpropagation iteration. We also increase our chances of exiting a local minimum since we have introduced some randomness to the training process. Finally, with the increased updates per epoch, we end up converging faster than when using gradient descent. A paper by Le Cunn et al [14] showed that stochastic gradient descent is a standard optimizer when training neural networks.

Adam: Adaptive Moment Estimation

Unlike stochastic gradient descent which has a single learning rate for all weight updates, Adam has a learning rate for each weight of the network. Furthermore, these learning rates are adapted during training. Adam was described and created by Kingma and Ba in 2015 [11]. It combines the advantages of two previous optimizers, RMSProp [10] a method that improves performance on noisy data and online problems, and AdaGrad [4] which has a learning rate for each weight, especially useful for problems with sparse gradients. To do this Adam adapts its learning rates based on both the first moment (the mean) and the average of the second moment (the uncentered variance) of the gradients. To do this Adam has four parameters; α representing the learning rate, $\beta_1 \in [0, 1)$ representing the exponential decay rate of the first moment, $\beta_2 \in [0, 1)$ representing the exponential decay rate of the average of the second moments, and ϵ which is used to ensure that there is no division by zero.

The update procedure for the Adam algorithm is as follows [11]:

$$
\begin{aligned}
g_t &= \nabla_\theta f_t(\theta_{t-1}) \quad \text{(Gradients of the loss function)} \\
m_t &= \beta_1 m_{t-1} + (1-\beta_1) g_t \quad \text{(Update of biased first moment estimate)} \\
v_t &= \beta_2 v_{t-1} + (1-\beta_2) g_t^2 \quad \text{(Update of biased second moment estimate)} \\
\hat{m}_t &= m_t / (1-\beta_1^t) \quad \text{(Bias-corrected first moment estimate)} \\
\hat{v}_t &= v_t / (1-\beta_2^t) \quad \text{(Bias-corrected second moment estimate)} \\
\theta_t &= \theta_{t-1} - \alpha \hat{m}_t / (\sqrt{\hat{v}_t} + \epsilon) \quad \text{(Parameter update)}
\end{aligned}
$$

where β_1^t and β_2^t denote β_1 and β_2 to the power t and θ_t represents the loss function at iteration t.

By only using the gradients, the method is computationally efficient and requires little memory. This can be seen in Figure 2.8. Good default values for the Adam parameters are 0.001 for the learning rate, 0.9 for β_1, 0.999 for β_2 and $1e^{-8}$ for ϵ. We initialize the first and second moments to be zero.

2.4.2 Weight Initialization

Weight initialization is a very important factor in determining how well our network will learn. To see this, we will consider three different scenarios:

1. **Initializing all the weights to zero**: This will lead our model to be the same as a linear model. This is because the derivative with respect to the loss will the same for all the weights, meaning that the weights will always have the same values.

2. **Initializing the weights to arbitrarily small values**: This is especially a problem in deep neural networks since small weights will lead to gradients becoming smaller and smaller. Therefore the first layers of

[5]Credit goes to author Diederik Kingma and Jimmy Lei Ba: https://arxiv.org/pdf/1412.6980.pdf

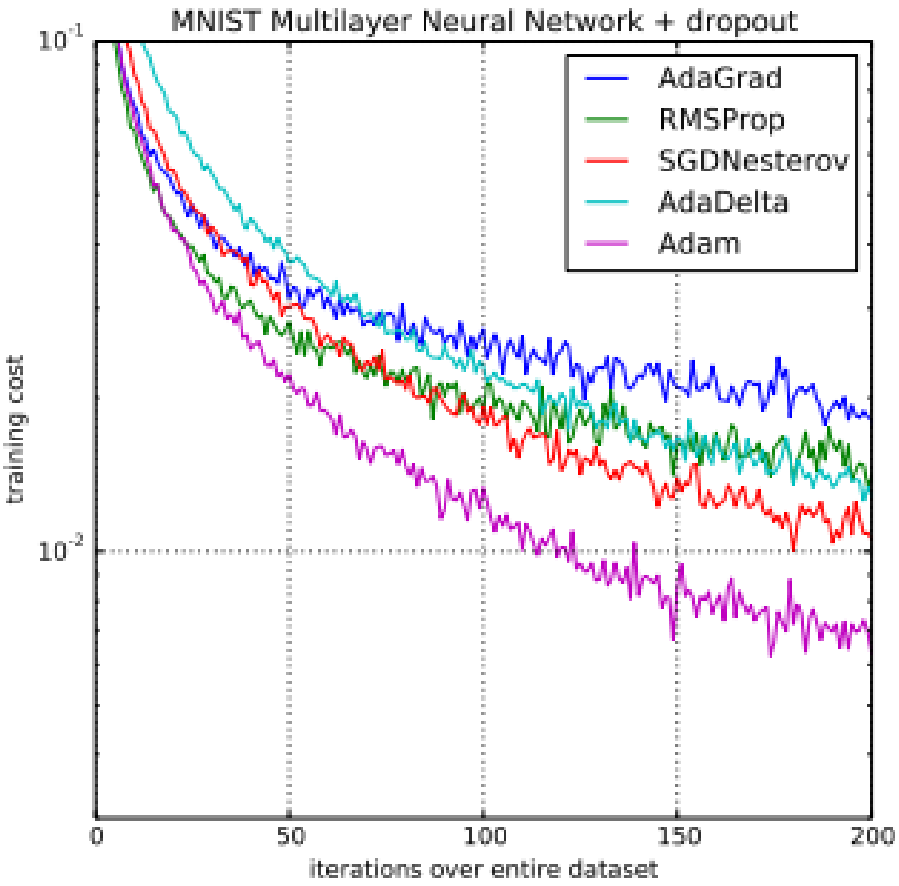

Figure 2.8: Comparison of the efficiency of the Adam optimizer versus other optimizers[5]

the model will only slowly improve if at all. This problem is called vanishing gradients.

3. **Initializing the weights to arbitrarily large values**: This leads to the opposite problem to the previous scenario, where the gradient becomes larger and larger to the point where there is a possibility of them becoming too big for the computer to assign a numerical value. This is called exploding gradients.

To solve all these problems, Glorot, and Bengio [5] came up with several initialization schemes. The one used in this thesis is called glorot uniform initialization. This initialization scheme uses the number of incoming nodes and the number of outgoing nodes to define a set from which the weights are uniformly initialized. Mathematically this comes to:

$$\mathcal{W} \in \mathcal{U}(-\sqrt{\frac{6}{n_{in}+n_{out}}}, \sqrt{\frac{6}{n_{in}+n_{out}}})$$

2.4.3 Training Batch Size

As seen in subsection 2.4.1 while training our models, we split our models into batches. The sizes of these batches are considered a hyper-parameter that can be optimized to better fit our model to the data. If we use a batch size, then we introduce too much randomness which might lead the model to take too long to train or in the worst cases not converge. On the other hand, if it is too large then we risk getting stuck in a local minimum. Therefore finding the right balance is important to achieve the best results for our model.

2.4.4 Dropout

Instead of affecting the initial weights or the hyper-parameters, we could modify the training procedure itself. If during training we decided to consider only a subset of the nodes of a particular layer, then we might be able to reduce the co-adaption problem that happens in fully-connected layers. This is when multiple nodes in a layer extract the same features, this typically occurs when nodes have very similar weights. By only considering a different subset of the nodes at each training iteration, we try to make the nodes in the network learn more about the data instead of trying to correct errors from previous layers. This method is called Dropout and was developed by Hinton, et al [10]. Explicitly, for a given hidden layer, we define a probability that a node be dropped or not. Then to make sure that mean of the nodes is the same whether nodes are dropped or not we multiply the values of the nodes by $\frac{1}{1-p_{drop}}$.

2.4.5 Activation functions

A final way to optimize a neural network is to change the activation functions used in the hidden layers. In this thesis, we only use the Rectified Linear Unit (ReLU) and softmax functions in our models.

Rectified Linear Unit

The ReLU function is the most commonly used activation function in modern machine learning models. This function was first introduced by Hahnloser et al. in 2000 [7] before being shown, in 2011, to train deeper neural network by Glorot et al. [6]. The ReLU function is the identity function for all positive values and zero otherwise. Mathematically:

$$f(x) = \begin{cases} x & \text{if } x \geq 0 \\ 0 & \text{else} \end{cases}$$

This function has a few advantages, it reduces computations since it either does not change the variable or sets it to zero. Also by having a constant gradient of one, it better avoids the vanishing gradient problem. Finally, it leads to sparser models since all nodes with values below zero,

do not contribute to the model. This however can lead to nodes 'dying' since a node could output zero for all inputs which means that it will never be able to learn.

Softmax

Once we have passed the input through all the hidden layers, we want to represent our outputs in such a way as to be comparable to our ground truths, which are usually vectors of zeros with a one for the position of the class of the image. Therefore we use the softmax function on the output layer. This makes sure that the output value for each class is between zero and one and that the sum of the outputs equals one. Mathematically the softmax function comes out to:

$$\theta(\mathbf{z})_i = \frac{e^{z_i}}{\sum_{j=0}^{K} e^{z_j}} \quad \text{for } i = 1, \ldots, K \text{ and } \mathbf{z} = \{z_1, \ldots, z_K\} \in \mathbb{R}^K$$

By doing this we ensure that all the values are between zero and one and that they add up to one.

2.5 Network Pruning

The main focus of our thesis is getting a deeper understanding of how nodes affect the neural network. To do this, we prune our networks and see how the model reacts to it by assessing the change in performance. While this is relatively unexplored, network pruning has been done. This can be seen in [18] and [8]. However, the goal in these papers is not as much to understand the roles of nodes but to find a method to compress the model to make them more resource-efficient. As such, they employ a series of network pruning followed by re-training of the model, while in this thesis, we only prune without re-training. Furthermore, we use a stricter metric to prune our networks. In [18], the metric we used is referred to as oracle-loss. While it is explored quickly, it is considered too costly to compute. This leads to different methods to estimate the impact of nodes to be used. While this is more computationally efficient, it leads to slightly less accurate results when trying to determine how nodes affect networks. Therefore, we will stick to a stricter metric since we want to better understand the impact of nodes on the network.

2.6 Datasets

In this thesis, we consider four different datasets. The first three, MNIST, Fashion MNIST, and CIFAR-10 are simpler and cleaner image datasets that are often used to experiment with new techniques since they have good baselines to be compared to. In this thesis, we use these datasets to explore our methods and their effectiveness. The final dataset, Kvasir, is used to simulate the real-world performance of our methods.

2.6.1 MNIST

The MNIST (Modified National Institute of Standards and Technology) dataset [13] is a collection of black and white images of hand-written digits. The dataset contains 60,000 training images and 10,000 test images. Each image is of dimension 28 by 28. Each image is labeled into one of ten classes. Those classes are the digits zero to nine. Figure 2.9 depicts a few images from the MNIST dataset

Figure 2.9: Example images from the MNIST dataset[6]

2.6.2 Fashion MNIST

The Fashion MNIST dataset [28] is similar to the MNIST dataset wherein there are 60,000 training images and 10,000 test images, each of these images are black and white and have dimensions 28 by 28. However, instead of being images of hand-written digits, they are images of ten different categories of fashion items. The ten categories are as follow:

- Class 0: T-shirt/top
- Class 1: Trouser
- Class 2: Pullover
- Class 3: Dress
- Class 4: Coat
- Class 5: Sandal
- Class 6: Shirt

[6]Credit goes to author Josef Steppan: https://en.wikipedia.org/wiki/MNIST_database

- Class 7: Sneaker
- Class 8: Bag
- Class 9: Ankle boot

Figure 2.10 depicts some images from the Fashion MNIST dataset.

Figure 2.10: Example images from the Fashion MNIST dataset[7]

2.6.3 CIFAR-10

The CIFAR-10 (Canadian Institute For Advanced Research) dataset [12] also contains 60,000 training images and 10,000 test images. However, these images are RGB (in color) and have dimensions 32 by 32. They represent ten different categories of objects and animals. These ten categories are as follow:

Figure 2.11 shows ten random images of each class, and the ten class names.

[7]Credit goes to author Tensorflow.org: https://www.tensorflow.org/tutorials/keras/classification

[8]Credit goes to author Alex Krizhevsky: https://www.cs.toronto.edu/~kriz/cifar.html

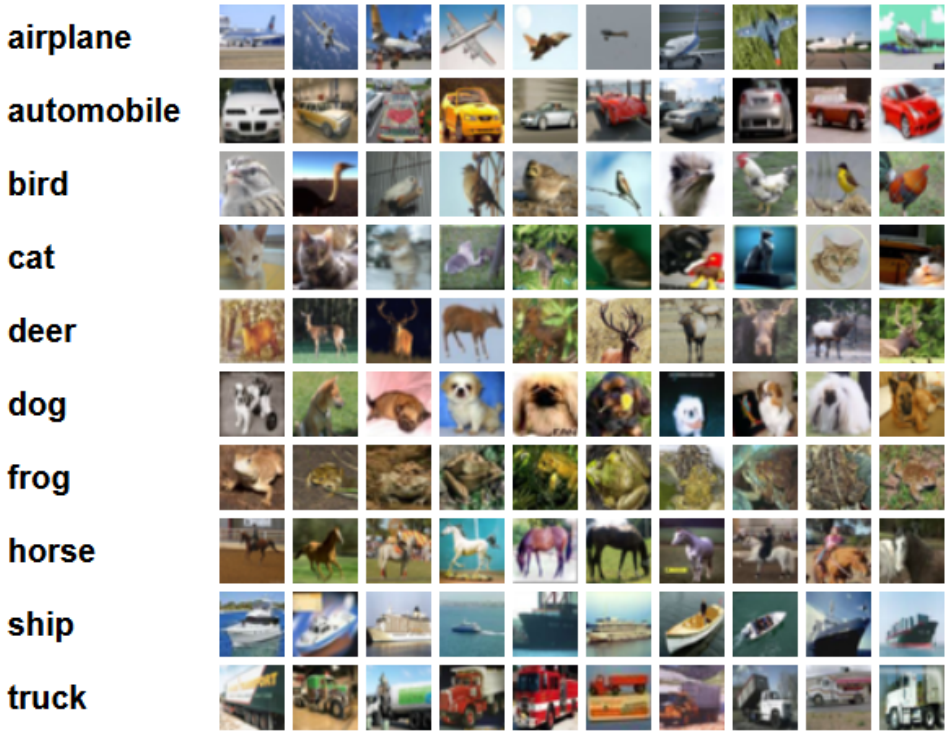

Figure 2.11: Example images from the CIFAR-10 dataset[8]

2.6.4 Kvasir

Unlike the three previous datasets, the Kvasir dataset [19] is used both for research and for real-world applications. The dataset consists of images, annotated, and verified by medical doctors. These images include several classes showing anatomical landmarks, pathological findings, or polyp removals in the GI tract. 8,000 images representing eight classes, each class containing 1,000 images. Each of these images is in RGB and have dimensions going from 720 by 576 to 1920 by 1072. The eight classes represented by these images are the following:

- Class Dyed and Lifted Polyps (dlp): where the polyps have been injected and lifted to make it easier to correctly remove it.
- Class Dyed Recesection Margins (drm): where the image shows the site of a removed polyp for a doctor to determine the completeness of the polyp removal.
- Class Esophagitis (eso): an inflammation of the esophagus visible as a break in the esophageal mucosa in relation to the Z-line.
- Class Cecum (nce): the most proximal part of the large intestine. Reaching it indicates a complete colonoscopy.
- Class Pylorus (npy): the area around the opening from the stomach into the first part of the small intestine.

- Class Z-line (nzl), marks the transition site between the esophagus and the stomach.
- Class Polyps (pol): lesions within the bowel detectable as mucosal outgrows.
- Class Ulcerative Colitis (uco): a chronic inflammatory disease affecting the large bowel.

The Dyed and Lifted Polyps and Dyed Recesection Margins classes are examples of polyp removals. The classes Cecum, Pylorus, and Z-line are examples of anatomical landmarks. The n prefix in the shortened class names indicates that the image represents a normal or healthy anatomical landmark. Finally, the classes Esophagitis, Polyps, and Ulcerative Colitis represent pathological findings. Figure 2.12 shows an example of each class in the Kvasir dataset.

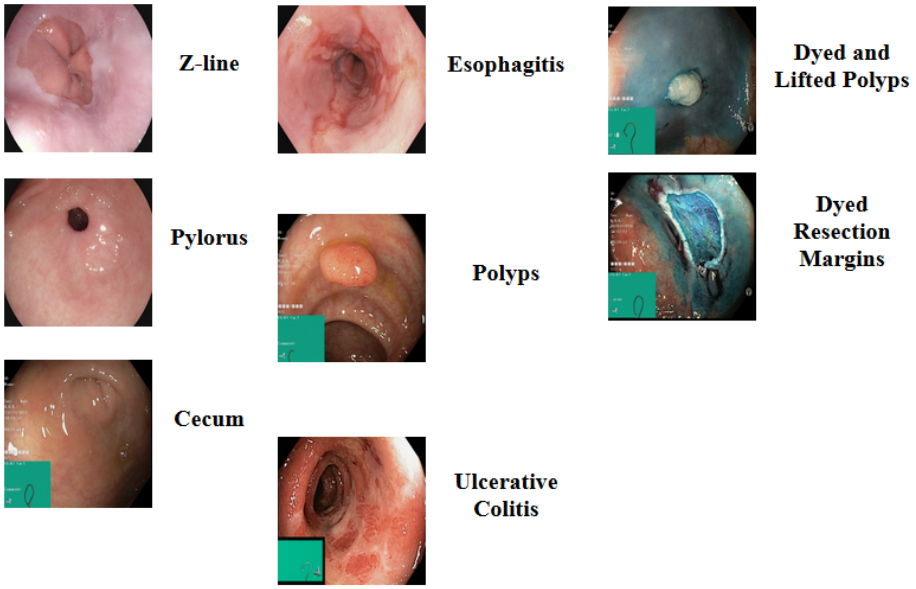

Figure 2.12: An example of each class in the Kvasir dataset[9]

2.7 VGG-16

In this thesis, we will first use simple CNN and MLP architectures with the MNIST, Fashion MNIST, and CIFAR-10 datasets. However, we will also test our methods on a more complex and used model called VGG-16. The VGG-16 model was first proposed by Simonyan and Zisserman [24] for the ImageNet Large Scale Visual Recognition Challenge [23] in 2014. It took the first position to classify images in 200 classes and second position to classify images in 1,000 classes. A visual representation of the VGG-16 architecture can be found in Figure 2.13.

[9]Credit goes to author Simula: https://datasets.simula.no/kvasir/#dataset-details

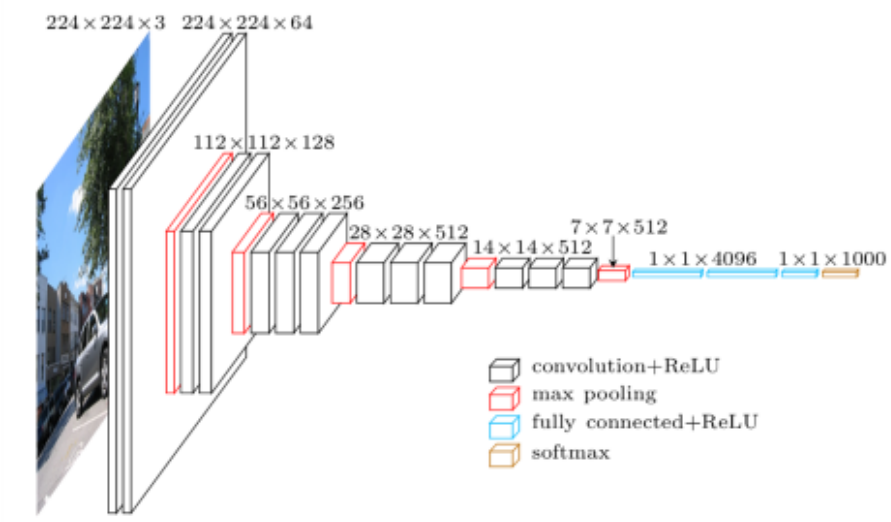

Figure 2.13: The architecture of a VGG-16 model to classify images in 1,000 different classes[10]

While the VGG-16 achieves high performance, it also has two main drawbacks. The first is that is very slow to train. The second is that it is large in terms of memory usage. This is due to the high number of parameters contained in the VGG-16 architecture. Therefore, smaller and similarly performing networks such as GoogLeNet [27] are often preferred.

2.8 Summary

We have presented a lot of different machine learning techniques and algorithms used commonly nowadays. We saw that there has been a great emphasis on using neural networks to do supervised learning. This has led to various new layers and optimization techniques to be developed to make a neural network more accurate, performant, and efficient. Furthermore, there has been much focus on making these networks deeper and larger at each layer (more nodes and filters in each layer). While this has been successful in achieving better results, it has also led to big networks that take up a lot of space and computational power. An investigation on the effects of nodes in each layer and whether all of them are required has been missing. In this thesis, we aim to better understand how nodes contribute to the neural networks' performance. By having a better understanding of which nodes contribute and which do not, we hope to be able to reduce wasted resources and improve performance. In the next chapter, we will describe our methodology. We will mention the techniques we use to classify nodes depending on their contribution to the network. Furthermore, we will describe the algorithms we use to remove the unhelpful nodes of a neural network.

[10]Credit goes to the author Rohit Thakur: https://towardsdatascience.com/step-by-step-vgg16-implementation-in-keras-for-beginners-a833c686ae6c

Chapter 3

Methodology

3.1 Neural Networks

In chapter 4, we consider three different types of neural networks to calculate node importance (see section 3.2) and be pruned. These are a single-layer ANN (which can also be called a single-layer MLP, but since MLP implies multiple layers and to make the distinction between this network and the next, we call it a single-layer ANN), a three-layer MLP, and a five-layer CNN. The reason we chose these three networks are as follows. The first network, a single-layer ANN, was chosen to check whether doing pruning or calculating node importance was viable and to see how removing a hidden node would impact the model, without it being related to other hidden nodes. We then consider a three-layer MLP to see how having multiple layers changes the impact of pruning and the node importance calculations. Finally, we consider a five-layer CNN to look at an even deeper model and to see how a convolutional layer reacts to pruning compared to the dense networks. All three neural networks will be trained on the MNIST and Fashion MNIST datasets. The CNN will also be trained on the CIFAR-10 dataset. The exact composition of the neural networks are as follows, for the single-layer ANN we have:

- *Hidden Layer*: 128 Nodes with a ReLU activation function
- *Activation function for Output Layer*: Softmax
- *Number of epochs*: five epochs for both MNIST and Fashion MNIST.

Then we have the three layer MLP:

- Structure:
    1. Dense hidden layer: 128 nodes with a ReLU activation function.
    2. Dense hidden layer: 64 nodes with a ReLU activation function.
    3. Dense hidden layer: 32 nodes with a ReLU activation function.
    4. Output layer with softmax activation.
- *Number of epochs*: for both MNIST and Fashion MNIST we will run five epochs of training.

Finally, we have the five layer CNN:

- Structure:
    1. Convolutional layer: 32 filters of size 3×3 with a ReLU activation function.
    2. Max Pooling layer with a window size of 2×2 and a stride of 2.
    3. Convolutional layer: 64 filters of size 3×3 with a ReLU activation function.
    4. Max Pooling layer with a window size of 2×2 and a stride of 2.
    5. Convolutional layer: 128 filters of size 3×3 with a ReLU activation function.
    6. Max Pooling layer with a window size of 2×2 and a stride of 2.
    7. Convolutional layer: 256 filters of size 3×3 with a ReLU activation function.
    8. Dense hidden layer of 64 nodes with a ReLU activation function.
    9. Output layer with softmax activation.
- *Number of epochs*: five epochs for all three datasets.

All these networks were coded using the TensorFlow [15] library for python.

Throughout our experiments, we will keep a few hyper-parameters of the networks constant. The batch size for training will be 32 samples, this will be kept constant except for when we explore the effect of batch size on node importance in section 4.3 (see section 3.2 for an explanation on node importance). We will use an ADAM optimizer with a learning rate of 0.001, a β_1 of 0.9, a β_2 of 0.999 and an ϵ of $1e^{-7}$. Further, all the weights are initialized using a Glorot uniform strategy [5]. The reason we chose these parameters was that they are the recommended defaults for these methods [11, 15]. Since we are trying to look at how effective pruning is, using the defaults that generally lead to good results seemed advisable.

From section 4.6, we will start considering a validation set, with an 85:15 split of the original training set, in addition to the train and test set. Before that, we only consider the train and test sets. The reason we consider a validation set is because we want a set that the model is not trained upon but that we can use to prune the model before testing its new performance with the test set. The train set contains 60000 images, and the test set contains 10000 images, this is the case for all three datasets considered. Finally, in section 4.4, we will have slightly modified versions of the ANN and MLP models since we explored the effect of having a dropout layer on the node importance. Namely, the single-layer ANN will be:

- *Hidden Layer*: 128 Nodes with a ReLU activation function and dropout.
- *Activation function for Output Layer*: Softmax

- *Number of epochs*: five epochs for both MNIST and Fashion MNIST.

and the MLP:

- Structure:
    1. Dense hidden layer: 128 nodes with a ReLU activation function and dropout.
    2. Dense hidden layer: 64 nodes with a ReLU activation function and dropout.
    3. Dense hidden layer: 32 nodes with a ReLU activation function and dropout.
    4. Output layer with softmax activation.
- *Number of epochs*: for both MNIST and Fashion MNIST we will run five epochs of training.

For the MLP, all the hidden layers will have the same dropout rate.

In chapter 5, we use a CNN with the VGG-16 architecture (see section 2.7 for details on the architecture) with eight outputs representing the eight classes in the Kvasir dataset. We will have a 70:15:15 split for the train, validation, and test sets, we will use a batch size of 32 and an SGD optimizer with a learning rate of 0.001 with a Nesterov momentum of 0.5 for training. We train the network for a maximum of 200 epochs with early stopping using a patience of seven epochs. Once the training stops, we revert to the weights to the ones that performed best on the validation set.

3.2 Node Importance

To estimate which nodes are important to the neural network when making a prediction, we use a self-defined metric called *node importance*. The node importance of a node is defined as the difference between the loss when the node is included and the loss when the node is removed. The more negative the node importance is, the more important the node is. This is described mathematically in Equation 3.1, where $\psi_{m,n}$ is node importance for node n in layer m and $\mathcal{C}^{m,n}$ is the loss when node n in layer m is removed.

$$\psi_{m,n} = \mathcal{C}(x, y) - \mathcal{C}^{m,n}(x, y) \tag{3.1}$$

Using the node importance metric and two user-defined tolerances called low tolerance and high tolerance, we define three types of nodes:

- Important nodes: when the node has a lower node importance value than the low tolerance.
- Zero nodes: when the node has a node importance value between low tolerance and high tolerance.
- Worse nodes: when the node has a node importance value higher than the high tolerance.

We set the low tolerance to $-1e^{-5}$ and the high tolerance to $1e^{-5}$. These tolerances are not strictly necessary as we could instead label all nodes with a node importance value above zero; worse nodes, all those under zero; important nodes, and all those with a zero node importance zero nodes. However, in this thesis, we want to see the effects of removing worse and zero nodes from the neural networks while keeping the accuracy and loss steady. Therefore, to achieve both goals, we relax the definition of important, zero, and worse nodes with a tolerance for the range of the zero nodes.

3.3 Node Pruning

We use two different strategies to prune nodes in chapter 4. The first, which we explore briefly in section 4.1, is removing nodes at random from the neural networks. We do this in two different ways, the first is to randomly choose n nodes to remove and remove them (see algorithm 1). The other is to choose n nodes to remove, check if removing them improves a combination of the accuracy and loss of the model. If it does improve it, we remove the nodes otherwise we do not remove them (see algorithm 2).

Past section 4.1, we prune nodes depending on the node importance class of the nodes. We do this in three different ways, the first, which we use in section 4.5, is to remove nodes dependent on their pre-calculated node importance's. In other words, we calculate the node importance of each node and then remove all the nodes that are considered worse or zero (see algorithm 3). The second method used (seen in section 4.6) is to find the worse of the worse/zero nodes (the one with the highest node importance value) in the layer, remove it from the neural network and then repeat until all the nodes are considered important nodes. We do this layer by layer, going backward from the hidden layer before the output (see algorithm 8). Finally, we use a variation of the previous version, where instead of finding the node with the highest node importance value, we remove the first worse/zero node we encounter and repeat till all nodes are considered important (see algorithm 9). We also 'ignore' all nodes that are considered too important by the user (have a node importance value under a certain threshold). This method is the one used for chapter 5.

The next section explains the algorithms used in this thesis in more detail and includes a pseudo-code version of the algorithms. With some of the pseudo-code of the algorithms being included in Appendix A.

3.4 Algorithms

In chapter 4 and chapter 5, we use a few self-defined algorithms to either prune nodes, estimate node importance, or edit initial weights of the neural network. In section 4.1, we use two algorithms that remove nodes at random from a single-layer ANN. The first algorithm we use in section 4.1 is algorithm 1.

Algorithm 1: Removing a user defined number of random nodes

```
1 def removeRandomNodes(n, weights, to_consider):
      Input:
      n is the number of nodes removed;
      weights are the weights of the model;
      to_consider is an array containing the nodes to consider in the
      random choosing
      Output: The weights with the nodes removed (set to zero) and
              the positions of the nodes removed
      /* Start of the code                                        */
2     to_drop ← choose n from to_consider without replacement
3     for i in to_drop:
4         weights[0][:,i] = 0 ;      /* weights going to the node */
5         weights[1][i] = 0 ;           /* bias going to the node */
6         weights[2][i,:] = 0 ;      /* weights outgoing the node */
7     return weights, to_drop
```

This algorithm is designed to work on single-layer ANNs only since it only considers the existence of a single hidden layer in the model. The algorithm randomly chooses n nodes from either the whole network or from a list (*to_drop*) passed by the user and then removes them from the network. The second algorithm used is algorithm 2.

Instead of removing nodes randomly, as in algorithm 1, the algorithm selects n nodes to remove from the network and checks whether removing them improves a combination of the accuracy and loss of the model. If it does improve the model, the nodes are removed. Otherwise, the model is kept the same. This is repeated a fixed number of times (the value *to_test* defined by the user), where each time the model improves, nodes are removed. Finally, it outputs the weights of the shrunk model and the number of nodes removed by the algorithm. These two algorithms tend to be unstable since we choose the nodes randomly.

To remedy this, we devise algorithm 4, which estimates the node importance of each node. The algorithm traverses the networks backward, as in backpropagation. This method is used in section 4.2, section 4.3 and section 4.4. We could also have traversed the network forward when estimating importance (as in algorithm 5), however, to simulate removing a node, we need it to output zero. Therefore, it makes sense to go backwards since we set the incoming weights to zero. This algorithm goes through the network layer by layer, removes a node, calculates the difference in loss, classifies the node as important, zero, or worse, and finally reverts back to the full model weights. It does this for every node in the layer before going to the next layer. As mentioned before, we also have algorithm 5, which calculates the node importance by going forward. However, this code is not used but is the basis for calculating node importance in a single layer done by algorithm 6.

algorithm 6 uses the forward logic to calculate node importance but

Algorithm 2: Shrinking the model by removing nodes randomly

```
1  def shrinkModelRandomly(model, acc, loss, weights, n, to_test, x_train,
     y_train, v):
       Input:
       model is the TensorFlow model of the neural network used;
       acc is the accuracy of the original model;
       loss is the loss of the original model;
       weights are the weights of the model;
       n is the number of nodes removed at each step;
       to_test is the number of times we try to remove nodes;
       x_train is the training dataset used;
       y_train is the labels of the training dataset used;
       v defines whether are output should be verbose or not;
       Output: The weights with the nodes removed (set to zero) and
                the number of nodes removed
       /* Start of the code                                          */
2      best_loss ← loss
3      best_acc ← acc
4      best_weights ← copy(weights)
5      num_removed ← 0
6      to_consider ← list from 0 to the number of nodes
7      for _ in range(to_test):
8          test_weights ← copy(best_weights)
9          test_weights, dropped = removeRandomNodes(n, test_weights,
             to_consider)
10         new_loss, new_acc ← evaluate model on x_train and y_train
11         score =
             (1 − (new_loss/best_loss)) + ((new_acc/best_acc) − 1)
12         if score > 0:
13             best_loss ← new_loss
14             best_acc ← new_acc
15             best_weights ← copy(test_weights)
16             increment num_removed by n
17             for node in dropped:
18                 remove node from to_consider
19     return best_weights, num_removed
```

does it only for one layer. As mentioned before, we can simulate removing a node by setting its incoming weights to zero. However, the opposite is also true, since setting all its outgoing nodes to zero means that the node has no effect on the nodes in the next layer as its contribution to the sum is zero. While algorithm 6 is not used on its own in our paper, we use it as part of algorithm 7 to re-randomize weights.

algorithm 7 re-randomizes the initial weights of the network until a defined minimum fraction of the nodes is considered important. We use

this in section 4.9. This algorithm 'optimizes' the weights layer by layer, starting at the first hidden layer and ending at the last one. To do this, it first obtains the node importance of the nodes in the layer it is currently optimizing. After that, it takes all the nodes considered either worse or zero and re-randomizes them using a Glorot uniform strategy. Once the minimum fraction of important nodes is reached, it locks the weights and moves on to the next layer. It repeats this process until all the layers are optimized.

Our final class of algorithms involves node pruning based on the node importance. We have three different algorithms that do node pruning. The first is algorithm 3.

Algorithm 3: This algorithm prunes the nodes classified as either zero or worse after there node importance is estimated.

```
1  def pruneNodesPreCalculatedImportance(model, tester_model,
     layer_sizes, tol_low, tol_high, x, y):
       Input:
       model is the TensorFlow model of the trained neural network
       used;
       tester_model is a untrained copy of model, this will be used as a
       tester model to test different network weights;
       layer_sizes is the number of nodes/filters in each layer of the
       considered network (organized in a backward direction);
       tol_low is the tolerance for which node importance values below
       it, categorize the nodes as important;
       tol_high is the tolerance for which node importance values
       above it, categorize the nodes as worse;
       x is the dataset to consider when estimating the node
       importance;
       y is the labels of x;
       Output: Pruned weights (where incoming weights are set to 0
               for the pruned nodes)
       /* Start of the code                                        */
2      weights ← weights of model
3      weight_len ← len(weights) − 3 ; /* We remove 3 for the bias
        and weight value to the output layer plus 1 because
        python lists start at 0 */
4      _, places ← estimateNodeImportance(model, tester_model,
        layer_sizes, tol_low, tol_high, x, y)
5      for layer, place in enumerate(places):
6          for nodes in place[0 : 2]:
7              weights[weight_len − (2 ∗ layer + 1)][..., nodes] ← 0
8              weights[weight_len − 2 ∗ layer][nodes] ← 0
9      return weights
```

For algorithm 3, the pruning is done by first calculating the node

importance of all nodes and then removing all nodes that are either classified as worse or zero. To do this, we first use algorithm 4 to get the class of each node. We then proceed to remove all the worse and zero nodes classified by algorithm 4. The second method is algorithm 8, this is the most time-consuming algorithm we use in this thesis.

The reason algorithm 8 is time-consuming is that it looks for the node with the highest node importance value in a layer, removes it, and then repeats till no nodes are considered either worse or zero. As for algorithm 4, it traverses the networks backward instead of forward. The reason we chose to traverse the network backward instead of forwards is that a node at a later layer should potentially have a smaller effect on the nodes of the previous layers than on the nodes in the next layers. Therefore, by removing the nodes closest to the outputs first, we try to minimize the effects of removing nodes on the node importance of other nodes in a different layer. As for the reason we remove zero nodes, it is because they could potentially be redundant nodes that do the same or similar work as other nodes in the layer. Furthermore, removing them reduces wasted resources, since it takes less processing power to compute a smaller network.

The final method (algorithm 9) we consider is a variation of algorithm 8, where instead of removing the node with the highest node importance value, we remove the first worse or zero node we encounter. We also 'ignore' nodes where if a node returns a node importance value under a user-defined threshold, we then never recalculate its node importance. The reason we have this more 'greedy' method is to reduce the time it takes to prune a network while having a method that takes into account the metric we use to classify nodes. algorithm 9 ends up being much faster than algorithm 8 while having very similar pruning results, see section 4.7 for the results. This algorithm is the one we use with algorithm 4 to both estimate the number of important, zero, and worse nodes and prune the network in chapter 5.

algorithm 4 to algorithm 9 can be found in Appendix A.

3.5 Summary

All these models, techniques, and algorithms will be used to give us a better grasp on how nodes contribute to a neural network. The node importance metric will help us quantify the contribution of a node, while the pruning algorithms will help us remove the unhelpful nodes that take up resources and processing time. These methods will be used in our two next chapters to experiment evaluating the contributions of nodes as well as the effects of pruning nodes.

Chapter 4

Exploring node pruning and node importance in simple neural networks

This chapter explores how pruning affects three different neural networks, a single-layer ANN, a three-layer MLP, and finally a five-layer CNN. We start by pruning the nodes randomly on the single-layer ANN. We then introduce the node importance metric and look at how having a validation set or not affects it. Next, we look at how changing the batch size affects the node importance of the different neural networks. Finally, we will look at how dropout changes node importance for the single-layer ANN and MLP networks.

We will then move on to pruning the models dependent on the node importance. We will start by pruning the model based on their pre-calculated node importance. We will check if this ends up improving the models. We will move on to pruning the models exhaustively using algorithm 8, seeing how many nodes are removed and how much of an improvement we earn after the pruning. We will then test another method of pruning to prune nodes faster. We will compare the pruning done by the greedy method to the one done by the exhaustive method and check whether they perform similarly. Furthermore, we will look at how pruning the CNN network affects the accuracy of each class. We will end by exploring whether initializing weights using node importance improves the final models.

4.1 Pruning Nodes at Random

In this section, we use two different algorithms to prune nodes at random. Both these algorithms are designed to work on single-layer ANNs since, in this section, we only consider those. algorithm 1 removes n nodes (where n is a value chosen by the user) from either all the nodes in the hidden layer or from a list defined by the user (the value called *to_consider*).

algorithm 2 is one that shrinks an ANN through trying to remove n random nodes at each step. The n random nodes will only be removed

if removing them improves a combination of the loss and accuracy of the model. We try removing nodes *to_test* number of times. Finally, once nodes are removed, they cannot be randomly selected again.

We will start by exploring the effects of removing an increasing number of nodes randomly by using algorithm 1 has on an ANN trained on first the MNIST dataset. Then on another ANN trained on the Fashion MNIST dataset. We will then explore how shrinking our models by using algorithm 2 affects accuracy, loss, and the total number of nodes removed when varying the number of nodes removed at each step.

4.1.1 MNIST

After training our single-layer ANN, we obtain the following statistics on the test set:

- **Accuracy**: 0.9724
- **Loss**: 0.0879

We will first focus on the change in accuracy and loss when removing an increasing number of nodes. To do this, we try removing randomly 1, 2, 4, 8, 16, 32, and 64 nodes from the model. For each number of nodes removed, we repeat the removal algorithm 1000 times on the same initial model and then record the loss and accuracy of the modified model. We then get aggregate the results to get some basic statistics on them.

	1	2	4	8	16	32	64
mean	0.9720	0.9714	0.9704	0.9681	0.9619	0.9427	0.8485
std	0.0009	0.0013	0.0020	0.0032	0.0056	0.0130	0.0417
min	0.9687	0.9624	0.9602	0.9523	0.9270	0.8861	0.6831
25%	0.9715	0.9708	0.9694	0.9664	0.9588	0.9364	0.8215
50%	0.9721	0.9717	0.9708	0.9685	0.9628	0.9448	0.8546
75%	0.9726	0.9724	0.9719	0.9704	0.9660	0.9526	0.8803
max	0.9736	0.9743	0.9744	0.9749	0.9724	0.9662	0.9328

Table 4.1: Statistics on the accuracy of the model after randomly removing a varying number of nodes

As can be seen in Figure 4.1, as the number of nodes removed increases, the accuracy of the model decreases. We also see that, on average, we never get better accuracy than the original model. If we focus on Table 4.1, we see that when removing one, two, or four nodes, the 75-percentile and above have similar or better accuracy than the original model. We can also see that the maximum value when removing 8 or 16 nodes have higher for the former, and the same for the latter, accuracy. Therefore, by removing nodes, we can probably make our model better. However, we might need to remove them in a more controllable way, since on average, we get lower (although not significantly) accuracy (spanning from 0.9720 to 0.9616 when removing 1 to 16 nodes). However, we also see that when we remove too

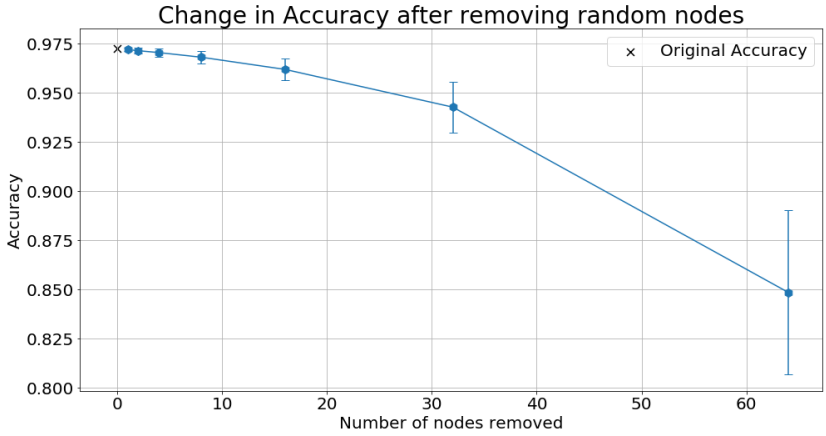

Figure 4.1: The average accuracy of the model after randomly removing nodes

many nodes (32 or 64), we end up with an accuracy that is always lower than the original model (even at its maximum values). A mean accuracy not too significantly lower for 32 nodes (0.9427 instead of 0.9724) but much lower when removing 64 nodes (around a 12% drop in accuracy).

Since the MNIST dataset is relatively unbiased, the accuracy can be considered a good measure to determine the performance of the model. However, it does not tell us everything. Therefore, including the change in loss is important to get a better understanding of the performance implications of the model after removing nodes.

	1	2	4	8	16	32	64
mean	0.0895	0.0914	0.0948	0.1024	0.1216	0.1794	0.4466
std	0.0026	0.0038	0.0058	0.0095	0.0164	0.0361	0.1074
min	0.0848	0.0839	0.0838	0.0844	0.0901	0.1127	0.2306
25%	0.0877	0.0887	0.0907	0.0957	0.1101	0.1516	0.3662
50%	0.0890	0.0907	0.0937	0.1012	0.1186	0.1738	0.4313
75%	0.0908	0.0932	0.0981	0.1069	0.1305	0.1980	0.5112
max	0.0997	0.1245	0.1292	0.1489	0.2233	0.3409	0.9883

Table 4.2: Statistics on the loss of the model after randomly removing a varying number of nodes

Unsurprisingly, in Figure 4.2, we see that the behavior of the loss is very similar to the accuracy. The loss is at its lowest for the original model compared to the average loss of the modified models. However, when we focus on Table 4.2, we see that when removing one node, the 25-percentile and under have a lower loss than the original model. When removing two, four, or eight nodes, we still get at least one model (in this case, the ones that achieve the minimum loss) where the loss is smaller. However, when looking at the average loss, we see another story. For one or two nodes,

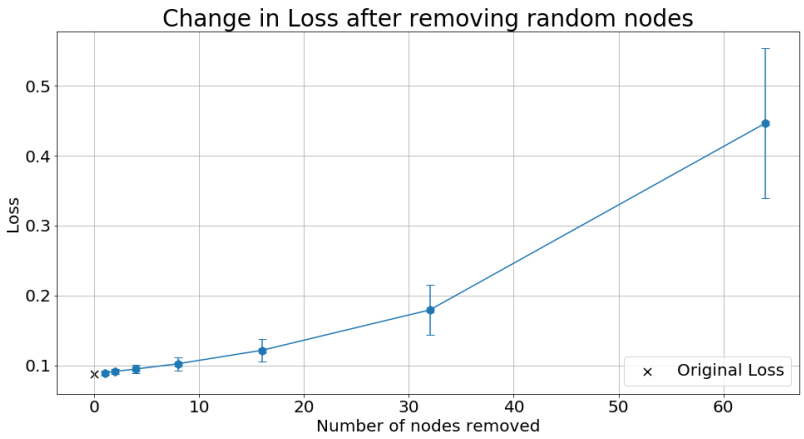

Figure 4.2: The average loss of the model after randomly removing nodes

we have a loss that is close to the original model. Once we start removing four or more nodes, we get a significant increase in the loss, with removing four nodes increasing the loss by 8% to quintupling of the original loss when removing 64 nodes. Therefore, when focusing on the loss, we see that removing nodes randomly deteriorates the performance of the model as the number of nodes removed increases.

To combat this, we will use algorithm 2 to shrink our models. Using this method, we continue picking randomly which nodes to remove while only removing them if they improve the loss and accuracy of the model. Instead of trying to remove up to 64 nodes at a time, we will only consider removing one, two, three, four, or eight nodes at a time. To get our results, we will do 20 tests for each amount to remove. In each of those tests, we will try removing nodes 64 times (the *to_check* value in algorithm 2) from the model.

	1	2	3	4	8
mean	7.4000	8.9000	8.2500	5.8000	2.4000
std	1.6983	2.7891	2.7314	3.3023	3.7613
min	5.0000	4.0000	3.0000	0.0000	0.0000
25%	6.0000	7.5000	6.0000	4.0000	0.0000
50%	7.0000	8.0000	9.0000	4.0000	0.0000
75%	8.2500	10.0000	9.7500	8.0000	8.0000
max	11.0000	16.0000	12.0000	12.0000	8.0000

Table 4.3: Statistics on the number of nodes removed depending on the size of the batch removed at each step

The first thing we see from Figure 4.3 is that we rarely remove more than ten nodes from our model. Another thing we see is that removing a smaller value of nodes at a time tends to remove more nodes in total, with a peak at two nodes in Figure 4.3. However, since the mean value of

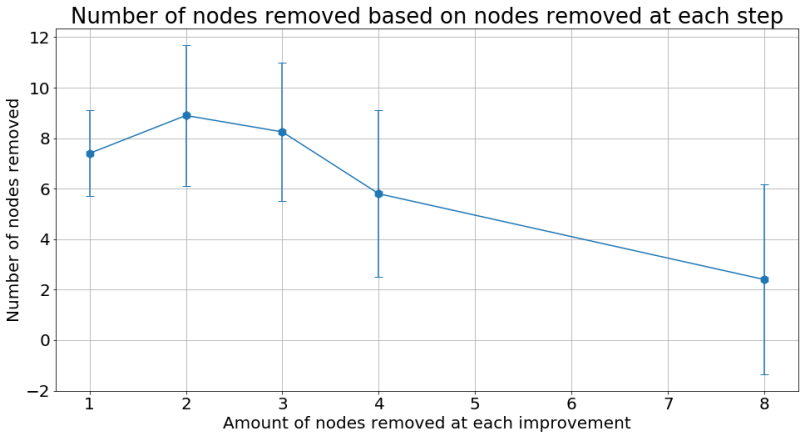

Figure 4.3: Average total number of nodes removed at each number of nodes removed at each step

two nodes is within the error bars of one and three nodes, removing one to three nodes at a time is very similar. This is reinforced by Table 4.3, where the minimum value for removing four or eight nodes at a time is zero nodes removed. By focusing on eight nodes removed at a time, we see that less than half of the models removed any nodes and at best a model removed eight nodes. However, since we only try removing nodes 64 times, as the number of nodes removed at a time approaches 64 we have more combinations that we can choose from, making it less likely to choose a combination that will improve our model compared to when removing a smaller number of nodes at each step.

Now, we will look at how the number of nodes removed affects the loss and accuracy. We will do this by considering just the total of nodes removed and not how big the removal was at each step.

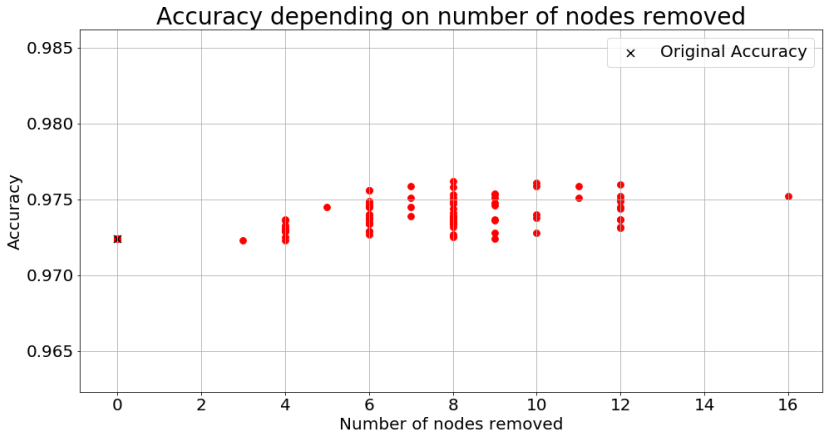

Figure 4.4: Accuracy of shrinked models depending on the number of nodes removed

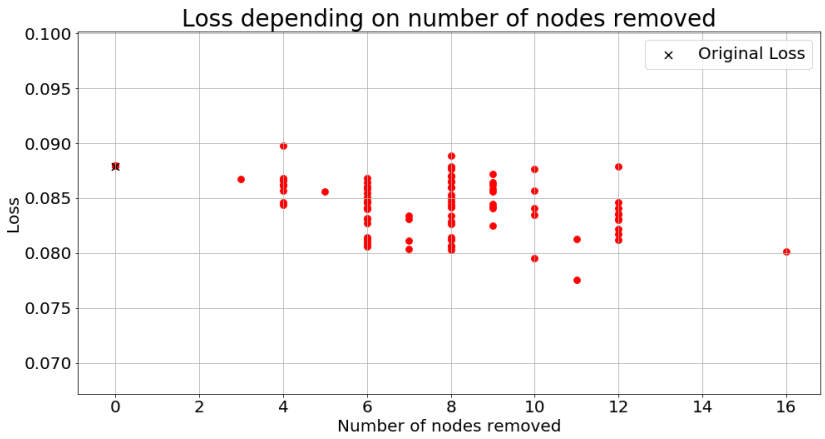

Figure 4.5: Accuracy of shrinked models depending on the number of nodes removed

From Figure 4.4 and Figure 4.5, we see that we get a better loss or accuracy for every modified model, albeit a few models where the final loss was slightly worse. Surprisingly, it looks like the more nodes we remove, the better the model does, although it is not very distinct. Moreover, it looks like the maximum decrease in the loss or increase in accuracy quickly flattens out as more nodes are removed. However, since it does not get worse, it might indicate that by using a more guided method to remove the nodes, we might be able to remove more nodes.

4.1.2 Fashion MNIST

Before we delve into a more guided method, we will re-do the same investigation as in subsection 4.1.1 but on the Fashion MNIST dataset. We expect to get a very similar result, but we might have some noticeable differences since this is a slightly more complicated dataset. After training our single-layer ANN, we obtain the following statistics on the test set:

- **Accuracy**: 0.8685
- **Loss**: 0.3557

As before we, will try removing randomly 1, 2, 4, 8, 16, 32, and 64 nodes. We will do it 1000 times and then get statistics on the accuracy and loss of the updated models.

Looking at Table 4.4, we see that, as for the MNIST dataset, the accuracy stays similar up to 16 nodes removed with all of them having a max accuracy value above the original accuracy. This is slightly different than with the MNIST dataset since, this time, we have a removal of 16 nodes that ends up with higher accuracy than the original accuracy. However, as for the previous dataset, removing 32 or 64 nodes decreases the accuracy noticeably. Now we will focus on the changes in the loss.

When looking at Table 4.5, we again get very similar results for the loss in this new dataset compared to the previous dataset. We get results close

	1	2	4	8	16	32	64
mean	0.8677	0.8667	0.8652	0.8618	0.8543	0.8295	0.7394
std	0.0021	0.0030	0.0042	0.0059	0.0096	0.0200	0.0443
min	0.8578	0.8470	0.8470	0.8405	0.8050	0.7382	0.5493
25%	0.8670	0.8653	0.8632	0.8582	0.8491	0.8190	0.7171
50%	0.8684	0.8674	0.8659	0.8626	0.8558	0.8336	0.7437
75%	0.8686	0.8685	0.8681	0.8663	0.8611	0.8441	0.7698
max	0.8726	0.8735	0.8745	0.8767	0.8747	0.8656	0.8307

Table 4.4: Statistics on the accuracy of the model after randomly removing a varying number of nodes

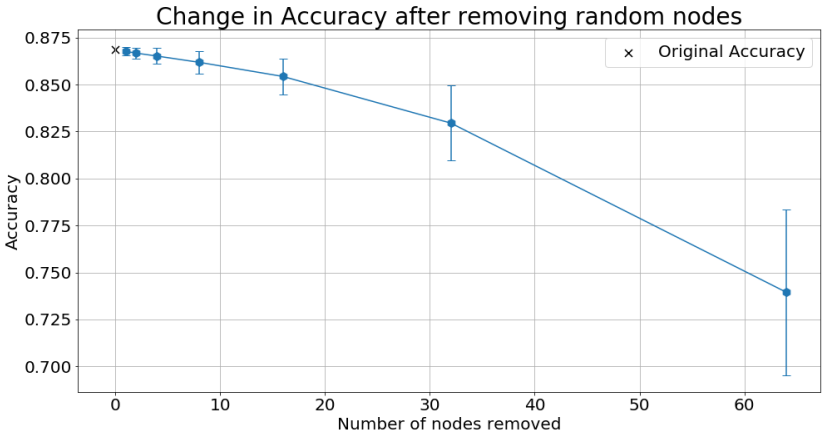

Figure 4.6: The average accuracy of the model after randomly removing nodes

	1	2	4	8	16	32	64
mean	0.3578	0.3601	0.3643	0.3730	0.3939	0.4607	0.7062
std	0.0049	0.0070	0.0098	0.0137	0.0228	0.0495	0.1182
min	0.3459	0.3452	0.3434	0.3448	0.3515	0.3725	0.4730
25%	0.3557	0.3559	0.3576	0.3629	0.3772	0.4245	0.6235
50%	0.3564	0.3584	0.3625	0.3711	0.3900	0.4504	0.6867
75%	0.3591	0.3632	0.3693	0.3811	0.4060	0.4857	0.7619
max	0.3791	0.4055	0.4035	0.4297	0.5106	0.7079	1.3649

Table 4.5: Statistics on the loss of the model after randomly removing a varying number of nodes

to the original loss for removing one to four nodes, before increasing by around 5% on average for eight nodes removed and more as more nodes are removed. However, as for accuracy, we get a shrunk model by 16 nodes, where we get a better loss than the original one. This was not the case for the MNIST dataset. In conclusion, we get similar results for MNIST and

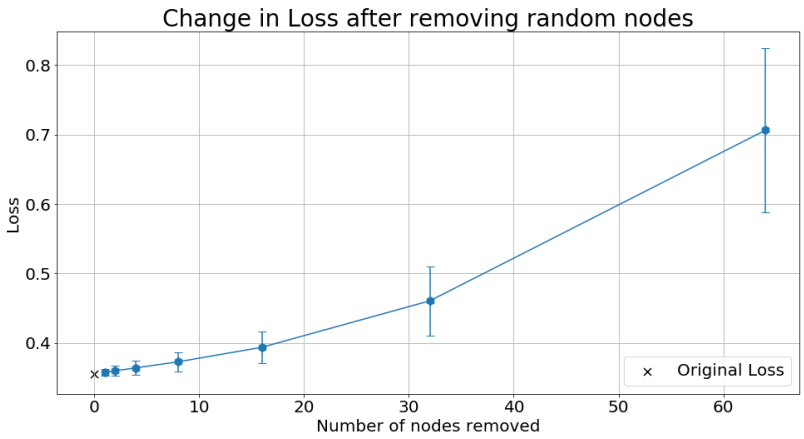

Figure 4.7: The average loss of the model after randomly removing nodes

Fashion MNIST when removing nodes randomly. Still, it seems we will be able to remove a few more nodes when working with the slightly more complicated dataset.

Next, we will use algorithm 2 to shrink the models randomly while still improving them. We will again try removing the nodes 64 times. We will repeat the process 20 times.

	1	2	3	4	8
mean	5.5000	8.6000	10.6500	8.400	5.6000
std	1.5728	1.8468	3.1502	3.409	3.7613
min	2.0000	6.0000	3.0000	4.000	0.0000
25%	5.0000	8.0000	9.0000	7.000	0.0000
50%	5.5000	8.0000	12.0000	8.000	8.0000
75%	6.2500	10.0000	12.0000	12.000	8.0000
max	8.0000	12.0000	18.0000	16.000	8.0000

Table 4.6: Statistics on the number of nodes removed depending on the size of the batch removed at each step

Interestingly by looking at Table 4.6 we see that we remove fewer nodes than for the MNIST dataset when removing one or two nodes at each step, but when removing more nodes at a time, we get noticeably more nodes removed. Especially for eight nodes removed, wherein more than half the tests, we have eight nodes removed. Finally, we will look at the accuracy and loss of each of these models.

For both Figure 4.9 and Figure 4.10 we get very consistent results for any number of nodes removed. This could mean that only a few nodes improve the model, while a few others have almost no effect on the model. In contrast to the MNIST dataset, there is no clear increase in accuracy or decrease in loss when more nodes are removed. The results seem to be more stable.

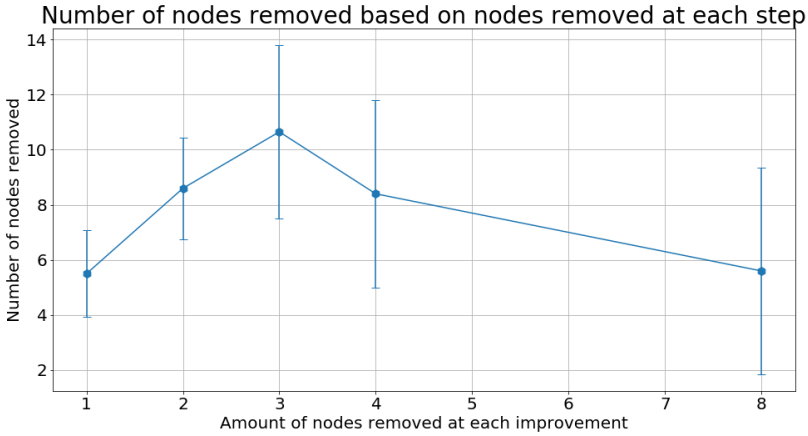

Figure 4.8: Average total number of nodes removed at each number of nodes removed at each step

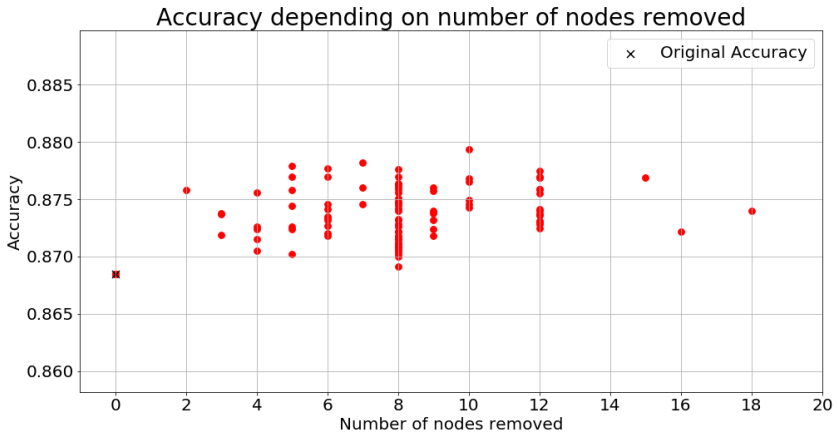

Figure 4.9: Accuracy of shrinked models depending on the number of nodes removed

4.1.3 Summary

In summary, removing nodes randomly is not the best idea if we want to shrink the models. We need to have a more guided way to choose which nodes to remove. We also tried to remove nodes randomly while still checking that the removal makes the model better or stay the same. We see that this does work where we can remove nodes and make the model perform better or at least stay the same. Finally, when comparing the results between models trained on MNIST compared to those trained on Fashion MNIST, we get very similar results, with probably a few more nodes being able to be removed for the Fashion MNIST models.

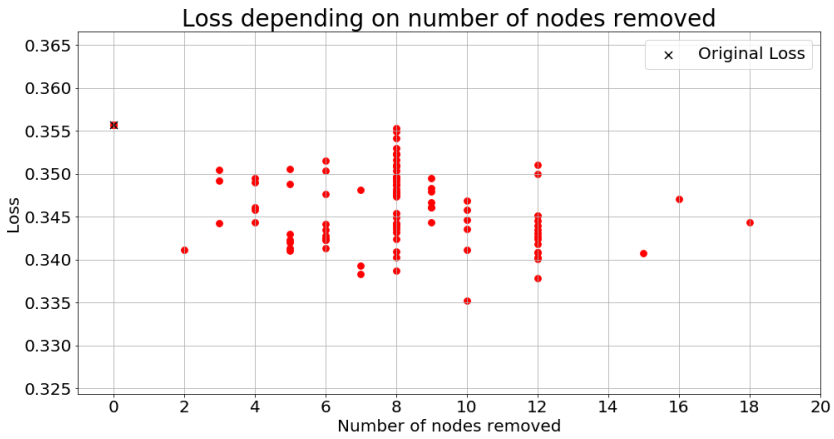

Figure 4.10: Loss of shrinked models depending on the number of nodes removed

4.2 Estimating Node Importance based on Loss

In this section, we use algorithm 4 to estimate the importance of nodes inside different models. We will continue using a batch size of 32 and train each model for five epochs. However, now instead of using one model, we will create multiple models with the same architecture and then estimate the node importance. We will also use two different sets to estimate node importance. First, we will estimate using the training set, where all the dataset is divided between the train and test sets. After we will estimate using the validation set, where the validation set is a part of the previous training set that is not used to train the model. The reason we do not use both sets for all experiments is that if we do not take advantage of the validation set in anyway, then it is better to have a bigger training set than having a smaller training set with extra data we do not use for anything. We consider the validation set in these experiments due to the noise fitting that happens during training. Using the validation set instead of the training set for estimating node importance might turn the nodes considered important by the training to worse nodes when it comes to the validation set. In other words, we think there will be more worse and zero nodes when using the validation set than when using the training set.

4.2.1 Single-layer ANN

We will estimate node importance on two datasets, MNIST and Fashion MNIST. However, we will only discuss our results for the MNIST dataset, for our results on the Fashion MNIST dataset and more detailed results, refer to section B.1. For our experiment, we will estimate the node importance of 50 different single-layer ANNs, for both datasets and sets (validation and training). We will start by discussing our results on the training set. Here is the average number of worse, zero, and important

nodes based on the training set:

- *Worse nodes*: 14
- *Zero nodes*: 2
- *Important nodes*: 112

From our results, it looks like most of our nodes are considered important. However, we still have 16 worse and zero nodes. Therefore, we should be able to reduce our models while also increasing the model performance. Compared to previously, when we removed the nodes randomly, we should be able to prune more nodes by basing ourselves on these values. We will now look at our result when using the validation set to estimate node importance:

- *Worse nodes*: 25
- *Zero nodes*: 2
- *Important nodes*: 101

Compared to when basing ourselves on the training set, we have close to double the number of worse nodes in this case, while the number of zero nodes is approximately the same. Therefore, this could indicate that the validation set picks up on nodes that end up being important for the training set because they fit the noise of the training set. While when using the validation set to classify them, end up being worse instead. This furthers our suspicion that we will able to remove more nodes using node importance than when doing it randomly.

For Fashion MNIST, we have the same difference between the training and validation set node importance estimates. The biggest difference is that there are many more zero nodes and between slightly (for the validation set) to a noticeable (around 50% increase) in worse nodes compared to the number of worse and zero nodes when trained on the MNIST dataset. Therefore, it seems that the more complicated a dataset or the lower the performance of the model is, the less important node there are.

4.2.2 MLP

We will estimate node importance on an MLP trained on both MNIST and Fashion MNIST. This time we will focus on the result obtained for Fashion MNIST. For our result on MNIST and more detailed statistics, refer to section C.1. We will again estimate the node importance for 50 MLP models for both datasets and both the validation set and training set. We will start by looking at our results on the training set. Table 4.7 shows the average number of nodes in each category for each layer.

As we can see, the number of worse nodes decreases as we go deeper into the model. However, if we consider the percentage of worse nodes, then we see that it first increases (from 14% to 19%) before decreasing (from 19% to 13%). Therefore it seems that the second layer has the highest

	Worse Nodes	Zero Nodes	Important Nodes
128-node layer	18	18	92
64-node layer	12	4	48
32-node layer	4	3	26

Table 4.7: Average number of node classified in each category of node importance based on the training set for an MLP trained on the Fashion MNIST

proportion of worse nodes. For the zero nodes, it is quite clear that the first layer has the highest proportion of zero nodes and that the second layer has the lowest. Finally, if we look at the proportion of unimportant nodes (aka zero and worse nodes), we see that the first layer has the highest proportion with 28%, the second layer has 25%, and the last layer has the lowest proportion with 22%. If we now look at Table 4.8 for our results on the validation data.

	Important Nodes	Zero Nodes	Worse Nodes
128-node layer	27	18	83
64-node layer	17	4	43
32-node layer	5	3	24

Table 4.8: Average number of node classified in each category of node importance based on the validation set for an MLP trained on the Fashion MNIST

As for the single-layer ANN, for all the layers, we have more worse nodes when using the validation set to estimate node importance, while the number of zero nodes stays the same. This further reinforces our point that using the validation set helps us capture the nodes that fit the noise rather than the actual data. Otherwise, we have the same layer behavior with the second layer having the most worse nodes and the last layer having the least. Except for the validation data, these proportion differences are slightly higher with 21% for layer one, 27% for layer two, and 16% for layer three. However, we continue having the highest proportion of unimportant nodes in the first layer with 35%, then we have the second layer with 33%, and finally, the last layer with 25%.

Our results on the MNIST dataset are very similar to the results for the Fashion MNIST dataset where all layers have more worse nodes on the validation set, while the number of zero nodes stays constant. For the proportion of worse nodes, the second and first layer have the same proportion (or slightly smaller for the second layer on the validation set) while the last layer again has less, although it is much smaller proportion wise than it is for the Fashion MNIST dataset (about half the proportion of nodes are worse nodes in the last layer compared to the first and second layer).

4.2.3 CNN

Finally, we will also estimate node importance for a CNN. We will do this on the MNIST, Fashion MNIST, and CIFAR-10 datasets. Although we will only discuss the results of the CIFAR-10 dataset, the results for the other two datasets and more detailed statistics can be found in section D.1. Instead of estimating the node importance of 50 models, we will only estimate the node importance of ten CNN models for each dataset and sets (validation and training). We will first look at our results on the training set. Table 4.9 shows the average number of each type of node in each layer of our CNNs.

	Worse Nodes	Zero Nodes	Important Nodes
32-filter layer	2	2	28
64-filter layer	3	8	53
128-filter layer	13	27	88
256-filter layer	25	75	156
64-node layer	1	38	25

Table 4.9: Average number of node classified in each category of node importance based on the training set for a CNN trained on the CIFAR-10

This time we see the opposite behavior compared to the MLP, where the deeper we go into the layer, the higher the proportion of worse nodes and zero nodes. The only exception is when we get to the dense layer, where the number of zero nodes does increase, but the number of worse nodes dramatically reduces. This could mean that most nodes are redundant at the last layer level and end up doing much of the same thing. Therefore removing them does not change anything to the model. While in the convolutional layers, we might still have some of those, but we also have more nodes that negatively impact the model. However, in terms of unimportant nodes, the proportion of them increases as we go deeper into the model, starting at 13% going all the way up to 61%. Table 4.10 shows our results for the validation set.

	Important Nodes	Zero Nodes	Worse Nodes
32-filter layer	5	2	25
64-filter layer	9	10	45
128-filter layer	25	26	77
256-filter layer	72	85	100
64-node layer	2	43	19

Table 4.10: Average number of node classified in each category of node importance based on the validation set for a CNN trained on the CIFAR-10

We see the same general trend as for the other neural network models, where the number of worse nodes increases in each layer. However, for all layers except the first and third, the number of zero nodes also goes

up, while previously, the number of zero nodes stayed similar between the two sets. All of this further supports the theory that using the validation set identifies the nodes that fit the noise of the training set rather than the actual images. Otherwise, the same behavior in terms of the proportion of unimportant, zero, and worse nodes can be observed here compared to when we were estimating using the training set.

We have pretty much the same behavior for the other two datasets. For the Fashion MNIST dataset, we have a higher proportion of worse nodes but a smaller proportion of zero nodes in general. The same goes for the MNIST dataset, except that in general, there are even more worse in each layer (the exception being the last convolutional layer), while the number of zero nodes is in-between the number for Fashion MNIST and the number for CIFAR-10.

4.2.4 Summary

As we saw in all three models, when using the validation set to estimate node importance, we end up having more nodes classified as worse or zero than when using the training set. Therefore our theory that using the validation data to calculate node importance leads to being able to identify nodes that contribute to fitting the noise of the training data instead of the data itself could be valid. Otherwise, we saw that when looking at dense layers, the deeper we go, the less worse and zero nodes there are. On the other hand, with convolutional layers, the opposite is true. However, since the size of layers usually decreases as we go deeper with dense layers, while when using convolutional layers, it usually increases, it could as well be due to the size of layers rather than a difference between convolutional and dense layers.

4.3 Effects of Changing Training Batch Size on Node Importance

Now that we have looked at the node importance on our set batch size of 32, we will look at whether changing the batch size affects the node importance. Furthermore, instead of estimating the node importance on both the training and validation set, we will only estimate node importance on the validation set. This not only speeds up the pruning but also helps us avoid classifying nodes as important when they are fitting the noise of the training set rather than the general pattern of the data. We will test five different batch sizes: 1, 8, 32, 256, and 1024.

4.3.1 Single-layer ANN

For the single-layer ANN, we will estimate node importance on 25 trained models at each batch size. We will record the node importance at each batch size. We will also record the loss and accuracy of each model. Finally, we

will also calculate the average node importance of the important and worse nodes.

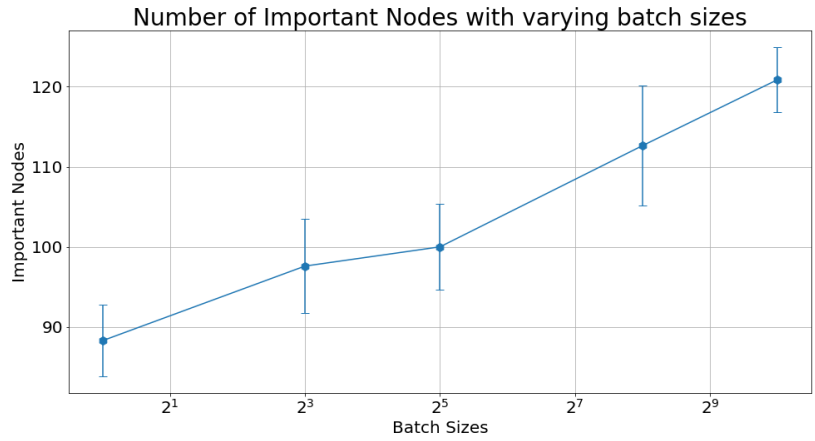

Figure 4.11: Average number of important nodes at each batch size for 25 single-layer ANNs trained on the MNIST dataset

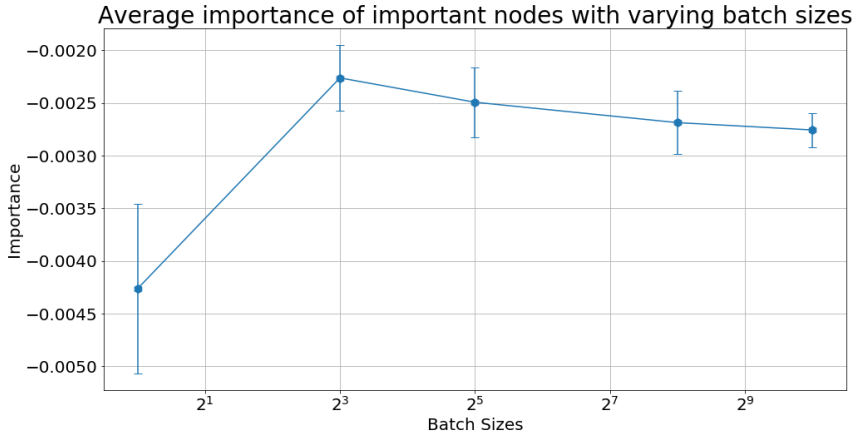

Figure 4.12: Average node importance of important nodes at each batch size for 25 single-layer ANNs trained on the MNIST dataset

If we start by looking at Figure 4.11, we see that the number of important nodes increases as the batch size increases, and with a batch size of 1024, almost all nodes (122 nodes on average) in the network are considered important. If we then look at Figure 4.12, we see that for a batch size of one, we have substantially lower average node importance than for the other batch sizes. The other batch sizes have roughly equal node importance when considering the error. The reason for such a difference is probably due to the differences between each image in the set. Since we do backpropagation for each image, we update the errors based on how one class is being classified. This could result in nodes specializing in detecting

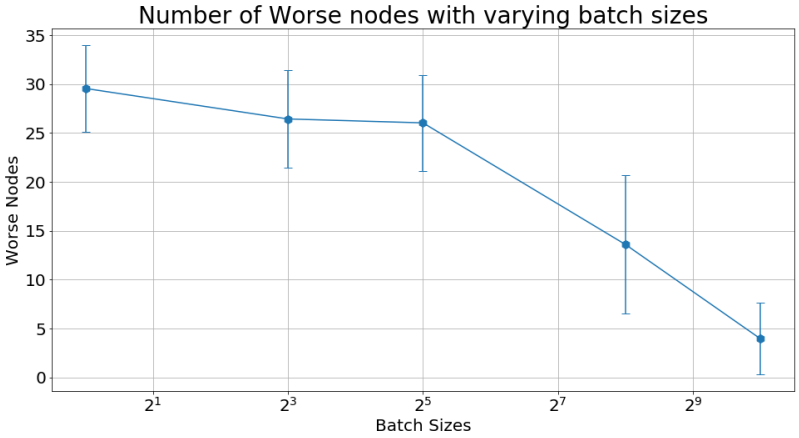

Figure 4.13: Average number of worse nodes at each batch size for 25 single-layer ANNs trained on the MNIST dataset

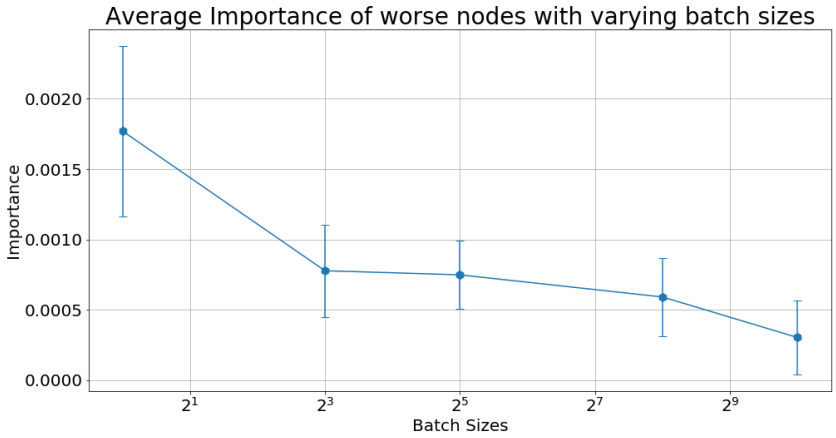

Figure 4.14: Average node importance of worse nodes at each batch size for 25 single-layer ANNs trained on the MNIST dataset

	1	8	32	256	1024
mean	0.9705	0.9759	0.9750	0.9657	0.9453
std	0.0029	0.0019	0.0013	0.0012	0.0017
min	0.9627	0.9715	0.9714	0.9631	0.9408
25%	0.9694	0.9748	0.9745	0.9652	0.9444
50%	0.9704	0.9760	0.9752	0.9658	0.9455
75%	0.9730	0.9776	0.9761	0.9666	0.9465
max	0.9751	0.9794	0.9767	0.9674	0.9481

Table 4.11: Statistics on the accuracy of the 25 single-layer ANNs trained on the MNIST dataset at each batch size

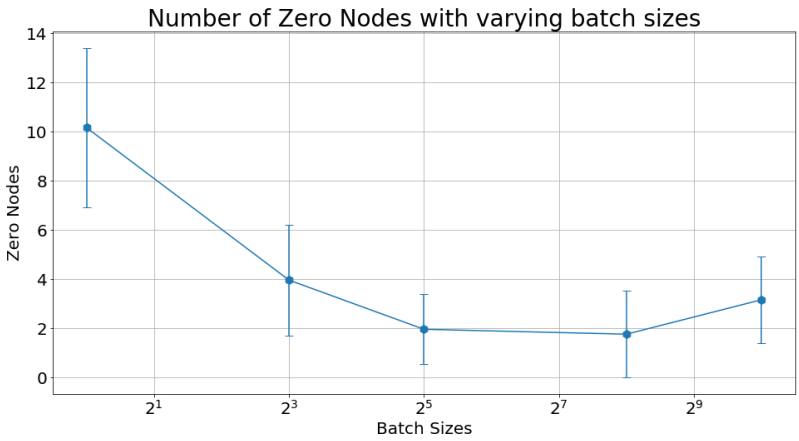

Figure 4.15: Number of zero nodes at each batch size for 25 single-layer ANNs trained on the MNIST dataset

	1	8	32	256	1024
mean	0.1687	0.0873	0.0810	0.1162	0.1924
std	0.0180	0.0062	0.0040	0.0031	0.0050
min	0.1410	0.0761	0.0730	0.1110	0.1855
25%	0.1553	0.0834	0.0786	0.1138	0.1897
50%	0.1682	0.0859	0.0805	0.1159	0.1912
75%	0.1773	0.0923	0.0830	0.1182	0.1950
max	0.2057	0.1031	0.0896	0.1226	0.2085

Table 4.12: Statistics on the loss of the 25 single-layer ANNs trained on the MNIST dataset at each batch size

a specific class, therefore removing such a node would significantly impact the performance of the model.

If we look at Figure 4.13, increasing the batch size decreases the number of worse nodes in the model. This is unsurprising since Figure 4.11 shows an increase in important nodes. However, in this case, the reduction seems to be accelerating as we increase the batch size, with the number of worse nodes for batch sizes 1, 8, and 32 being very similar (within error margins of each other) while a batch size of 256 halves the number of worse nodes in the model. This can be explained by Figure 4.15, where the number of zero nodes quickly decreases between batch sizes 1 and 32 before stabilizing around two zero nodes per model and finally slightly increasing for a batch size of 1024 (although the increase is within error margins). If we take a look at Figure 4.14, we see that the average node importance of the worse nodes decreases as batch sizes increases. This could be due to the same reasoning as for the node importance of important nodes, where, at smaller batch sizes, the model has a node that specializes in fitting the noise of the model and therefore end up being nodes that do not help classify new data.

Finally, if we look at Table 4.11 and Table 4.12, we see that we achieve

the best loss with a batch size of 32, the best accuracy is achieved with a batch size of eight. However, the accuracy for the models trained with a batch size of 32 is very comparable to the ones trained with a batch size of eight, while the loss in the former is noticeably better than the latter. If we now combine what we saw in Figure 4.11 to Figure 4.15 we see that even though we have more worse and zero nodes with a batch size of one compared to a batch size of 32 and the average node importance of worse nodes of the former is significantly higher, the difference in loss (0.17413 for the former and 0.08176 for the latter) is too large to get similar performance after removing them. This is especially true since the number of worse nodes between the two are comparable. Therefore, for the single-layer ANN, we get the best chance in reduction while keeping good performance with a batch size of 32.

We repeat this experiment on the Fashion MNIST dataset with the same parameters.

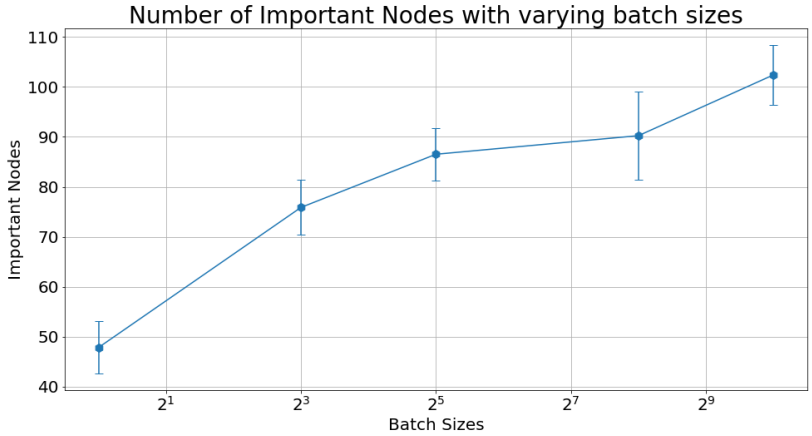

Figure 4.16: Average number of important nodes at each batch size for 25 single-layer ANNs trained on the Fashion MNIST dataset

	1	8	32	256	1024
mean	0.8520	0.8684	0.8703	0.8571	0.8443
std	0.0052	0.0054	0.0040	0.0068	0.0023
min	0.8435	0.8526	0.8619	0.8403	0.8382
25%	0.8494	0.8657	0.8676	0.8541	0.8430
50%	0.8508	0.8698	0.8697	0.8582	0.8446
75%	0.8554	0.8719	0.8741	0.8621	0.8462
max	0.8636	0.8752	0.8793	0.8666	0.8480

Table 4.13: Statistics on the accuracy of the 25 single-layer ANNs trained on the Fashion MNIST dataset at each batch size

If we compare Figure 4.16 to Figure 4.11, we get the same trend but the number of important nodes is smaller for each batch size. Focusing

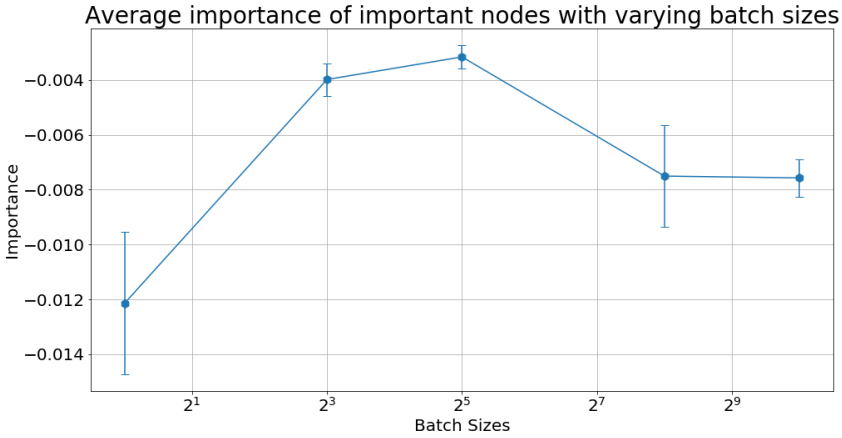

Figure 4.17: Average node importance of important nodes at each batch size for 25 single-layer ANNs trained on the Fashion MNIST dataset

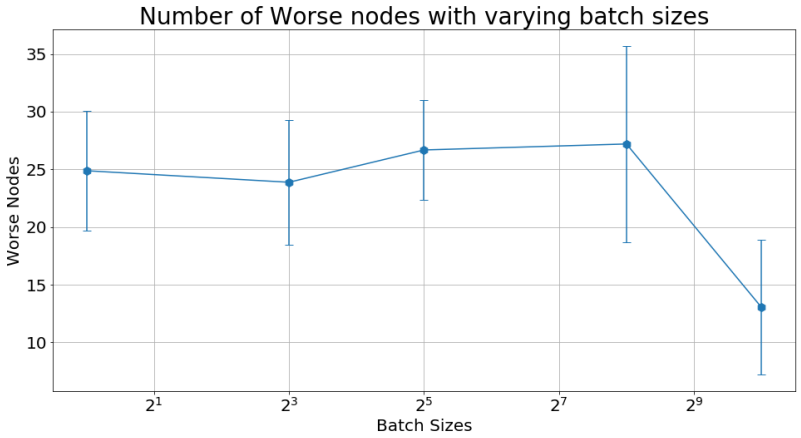

Figure 4.18: Average number of worse nodes at each batch size for 25 single-layer ANNs trained on the Fashion MNIST dataset

	1	8	32	256	1024
mean	0.4720	0.3731	0.3590	0.4012	0.4445
std	0.0231	0.0168	0.0083	0.0161	0.0040
min	0.4433	0.3497	0.3388	0.3807	0.4382
25%	0.4557	0.3604	0.3566	0.3901	0.4407
50%	0.4614	0.3695	0.3599	0.3965	0.4452
75%	0.4863	0.3805	0.3611	0.4076	0.4476
max	0.5336	0.4189	0.3788	0.4406	0.4530

Table 4.14: Statistics on the loss of the 25 single-layer ANNs trained on the Fashion MNIST dataset at each batch size

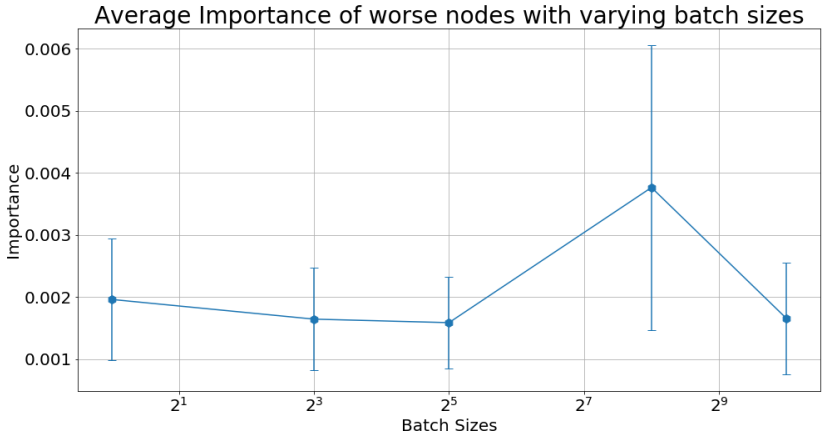

Figure 4.19: Average node importance of worse nodes at each batch size for 25 single-layer ANNs trained on the Fashion MNIST dataset

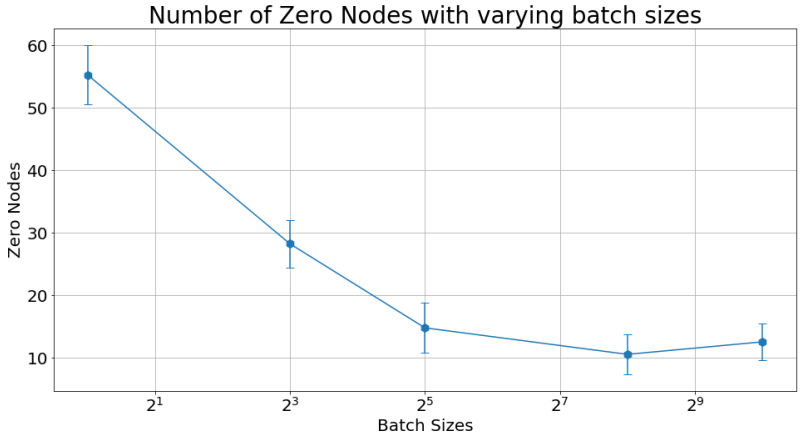

Figure 4.20: Average number of zero nodes at each batch size for 25 single-layer ANNs trained on the Fashion MNIST dataset

on Figure 4.17, we again see that a batch size of one generates important nodes with a smaller average node importance. However, the trend seen in Figure 4.12 is accentuated with the node importance of important nodes being significantly smaller at batch size 256 than at batch sizes 8 and 32. With Figure 4.18, we have a slightly different behavior where the number of worse nodes is approximately the same from batch size 1 to 256 (all within each other's margins of errors) before significantly decreasing at batch size 1024. Furthermore, from Figure 4.19, the average node importance for worse nodes is stable across all batch sizes (except for batch size 256, where it is noticeably higher). This is very different to Figure 4.14 where the node importance decreased as batch size increased. This is quite surprising since it seems to contradict our previous theory that nodes could specialize in the noise of the dataset. However, since this is a slightly more complicated

dataset, this might be due to the model not being to fully capture the differences between the different classes and therefore not being able to start overfitting on the training data. This is somewhat confirmed by Table 4.14, where the difference between the loss for batch size 32 (best loss on average) and the one for batch size one is not as different as previously.

Figure 4.20 behaves in much of a similar fashion as for the MNIST dataset, except that it stabilizes around ten nodes instead of two nodes. Once again, we see that having a batch size of 32 seems to be the sweet spot for node reduction and performance.

More detailed statistics on the number of nodes in each class of nodes and their importance's can be found in section B.2

4.3.2 MLP

As for the single-layer ANN, we estimate the node importance on 25 trained models at each batch size. We do this both for MNIST and Fashion MNIST. However, we will only discuss our results for Fashion MNIST, the results for MNIST as well as more detailed statistics on the results can be found in section C.2.

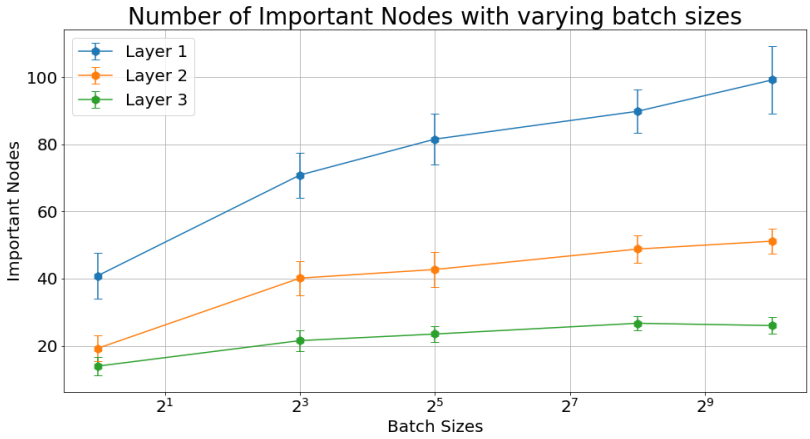

Figure 4.21: Average number of important nodes at each batch size and layer for 25 three-layer MLPs trained on the Fashion MNIST dataset

Before we start looking at our results, we must note that the number of nodes in each layer is different, with the first layer having the most nodes (128), while the last layer has the least nodes (32). This means that the number of nodes in each layer could be smaller as we go deeper into the model. However, we are focusing more on the trends than the actual numbers, so as long we have similar trends per layer, then the layers perform similarly.

From Figure 4.21 we see the same trend as in the ANN, where the bigger the batch size, the more important nodes there are. Looking at Figure 4.22, we have some interesting trends. First, if we look at average node importance for a batch size of one, we see that the deeper in the model

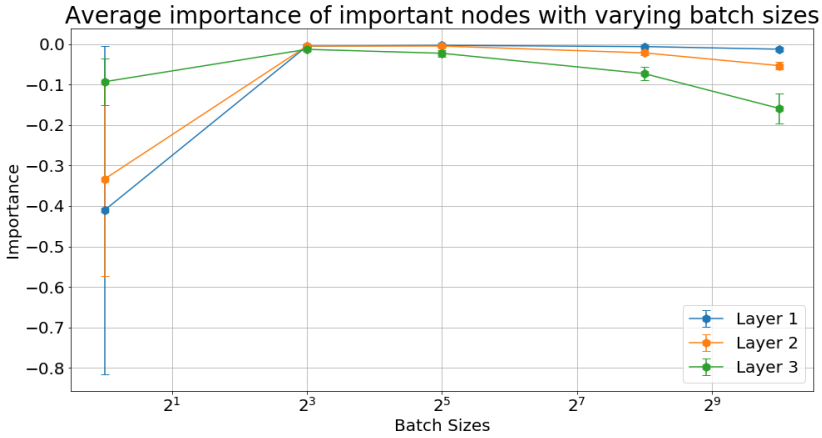

Figure 4.22: Average node importance of important nodes at each batch size and layer for 25 three-layer MLPs trained on the Fashion MNIST dataset

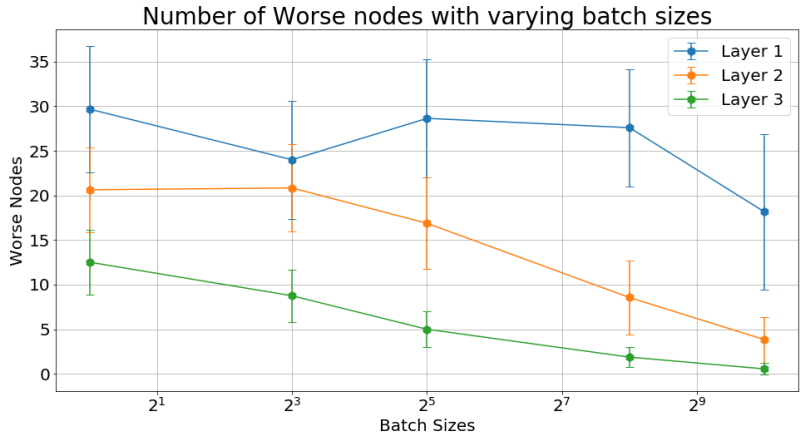

Figure 4.23: Average number of worse nodes at each batch size and layer for 25 three-layer MLPs trained on the Fashion MNIST dataset

	1	8	32	256	1024
mean	0.8448	0.8681	0.8691	0.8620	0.8494
std	0.0085	0.0061	0.0058	0.0048	0.0051
min	0.8258	0.8543	0.8570	0.8510	0.8351
25%	0.8397	0.8651	0.8644	0.8596	0.8467
50%	0.8474	0.8681	0.8700	0.8621	0.8508
75%	0.8504	0.8715	0.8735	0.8659	0.8529
max	0.8572	0.8773	0.8778	0.8689	0.8568

Table 4.15: Statistics on the accuracy of the 25 three-layer MLPs trained on the Fashion MNIST dataset at each batch size

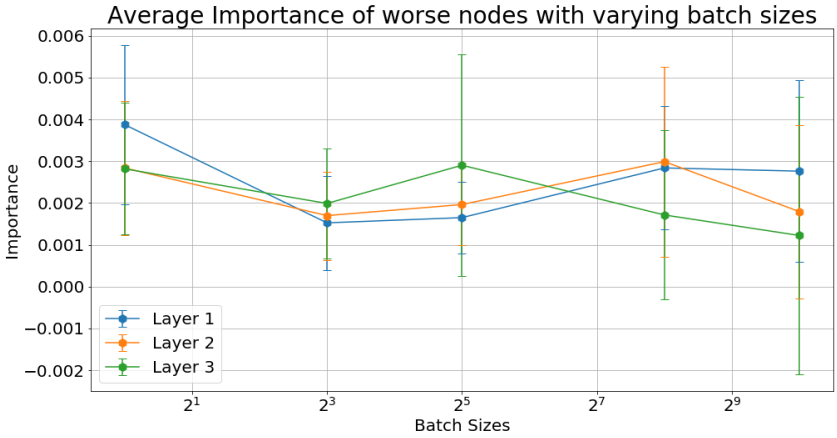

Figure 4.24: Average node importance of worse nodes at each batch size and layer for 25 three-layer MLPs trained on the Fashion MNIST dataset

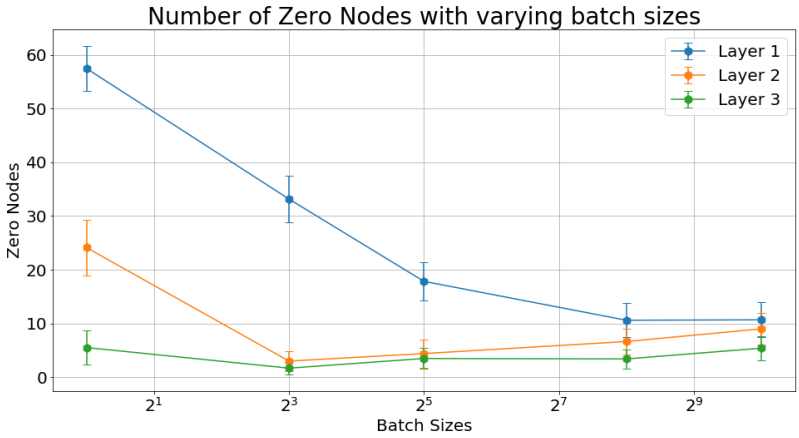

Figure 4.25: Average number of zero nodes at each batch size and layer for 25 three-layer MLPs trained on the Fashion MNIST dataset

	1	8	32	256	1024
mean	0.5081	0.3736	0.3662	0.3821	0.4266
std	0.0493	0.0186	0.0147	0.0119	0.0112
min	0.4367	0.3483	0.3447	0.3650	0.4084
25%	0.4753	0.3579	0.3562	0.3722	0.4190
50%	0.5035	0.3708	0.3625	0.3818	0.4254
75%	0.5289	0.3809	0.3740	0.3911	0.4373
max	0.6304	0.4174	0.3982	0.4074	0.4541

Table 4.16: Statistics on the loss of the 25 three-layer MLPs trained on the Fashion MNIST dataset at each batch size

we go, the closer to zero the node importance gets with a big increase in node importance between layers two and three. Another difference being the reduction in standard deviation as we go deeper into the model. This is interesting since there are more nodes in the first two layers compared to the last layer. This could be due to the compounded effect of removing a node in a shallow layer compared to a deeper layer since a node in a deeper layer affect fewer nodes than in one in a shallower layer. Then, as we increase to a batch size of eight, all the layer has roughly equal node importance which is close to zero. Past that point, as we increase the batch size during training, we see the deeper the layer, the quicker the node importance decreases. Especially for layer three, where the average node importance is noticeably smaller than for the other layers. A final detail is that while the standard deviation of layers one and two starts as much larger than for layer three, once we reach a batch size of 1028, the standard deviation of the node importance in layer three is noticeable while layers one and two have almost no standard deviation.

Figure 4.23 shows that while the number of worse nodes in layer one stays relatively constant (it is smaller at a batch size of 1024, but since the standard deviation is relatively high, we could have a very similar number of worse nodes in layer one at a batch size of 1024), the number of worse nodes decreases as the batch size increases in layers two and three. If we now focus on Figure 4.24, we see that the average node importance of the worse nodes is approximately the same across all batch sizes and layers. Again, the loss and accuracy of the model are not as good as for MNIST, therefore, this could contribute to the theory that nodes can only specialize in the noise if the model can sufficiently learn the differences between the classes. Figure 4.25 has very similar trends to the ones seen in the ANN models, where the number of zero nodes decreases before stabilizing or slightly increasing as batch size increases.

As for the ANN models, we achieve our best loss/accuracy when training with a batch size of 32 (as seen in Table 4.15 and Table 4.16). Since the trends for the worse and zero nodes are very similar to the ones seen in the ANN models and we attain the best loss/accuracy with a batch size of 32, we think using a batch size of 32 is optimal for node pruning.

4.3.3 CNN

We finally repeat the experiment for our CNN model. However, instead of looking at 25 models for each batch size, we only look at 15 models at each batch size. We do it on all three datasets, but we only do a detailed analysis of the CIFAR-10 dataset. For our results on the other datasets and more detailed statistics, refer to section D.2.

The first aspect of note in Figure 4.26 is how close to zero the number of important nodes are when the batch size is one. By looking at Table 4.17, we can understand this better since at least 75% of all models have an accuracy of 0.1. Since there are ten different classes, that means the model is randomly guessing which class each image is from. If we also take a look at Figure 4.28 and Figure 4.30, we see that almost all the nodes are zero

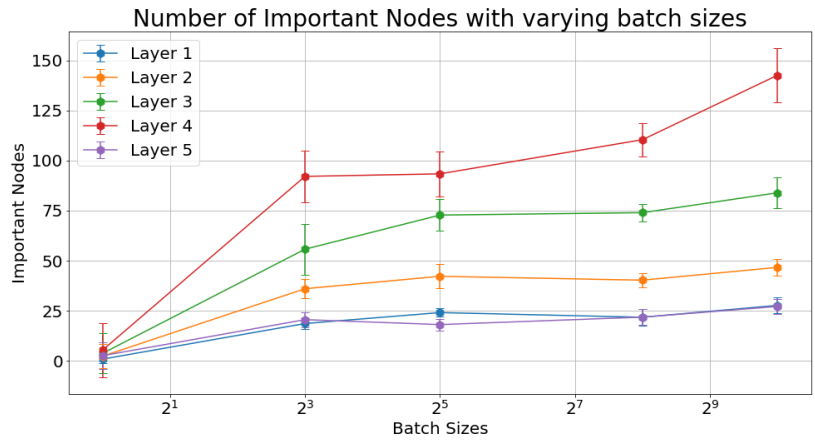

Figure 4.26: Average number of important nodes at each batch size and layer for 15 CNNs trained on the CIFAR-10 dataset

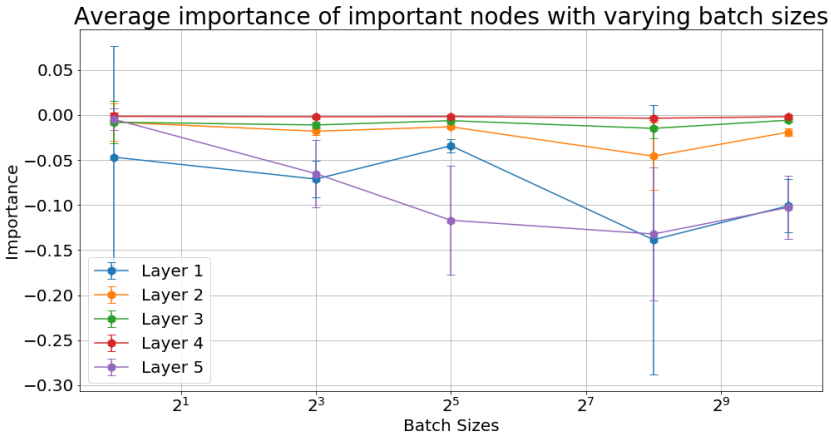

Figure 4.27: Average node importance of important nodes at each batch size and layer for 15 CNNs trained on the CIFAR-10 dataset

	1	8	32	256	1024
mean	0.1592	0.6883	0.7222	0.5757	0.5736
std	0.1564	0.0170	0.0098	0.1130	0.0138
min	0.1000	0.6461	0.7050	0.2925	0.5420
25%	0.1000	0.6827	0.7164	0.5701	0.5669
50%	0.1000	0.6909	0.7239	0.6151	0.5720
75%	0.1000	0.7008	0.7286	0.6381	0.5834
max	0.5503	0.7097	0.7364	0.6595	0.5970

Table 4.17: Statistics on the accuracy of the 15 CNNs trained on the CIFAR-10 dataset at each batch size

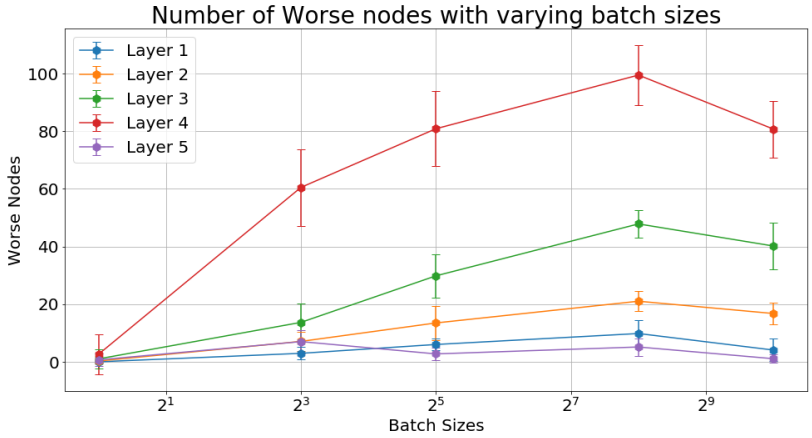

Figure 4.28: Average number of worse nodes at each batch size and layer for 15 CNNs trained on the CIFAR-10 dataset

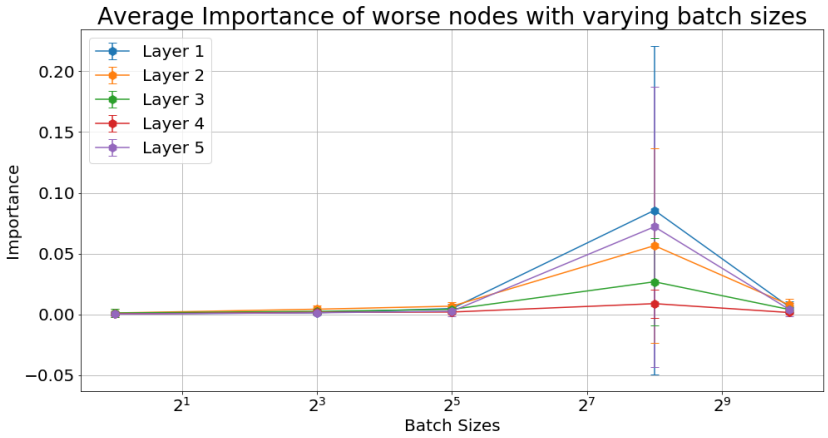

Figure 4.29: Average node importance of worse nodes at each batch size and layer for 15 CNNs trained on the CIFAR-10 dataset

	1	8	32	256	1024
mean	2.1697	0.9285	0.8289	1.4442	1.1973
std	0.3525	0.0557	0.0311	0.8975	0.0347
min	1.2895	0.8709	0.7752	0.9675	1.1469
25%	2.3030	0.8878	0.8094	1.0275	1.1760
50%	2.3031	0.9233	0.8350	1.0945	1.1993
75%	2.3033	0.9338	0.8460	1.2947	1.2112
max	2.3038	1.0835	0.8787	4.0439	1.2748

Table 4.18: Statistics on the loss of the 15 CNNs trained on the CIFAR-10 dataset at each batch size

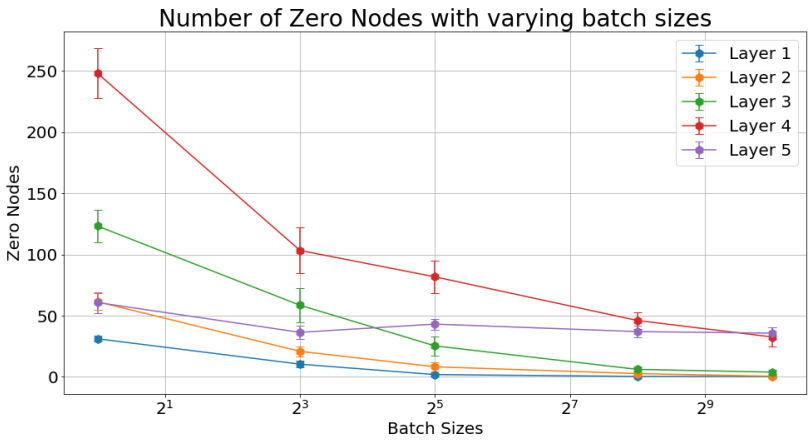

Figure 4.30: Average number of zero nodes at each batch size and layer for 15 CNNs trained on the CIFAR-10 dataset

nodes, meaning that the model does not do much classification, but rather randomly assigns values to each data point. Therefore it would make sense that there are either none or almost no important or worse nodes in the model (this is confirmed in the tables found in section D.2).

Otherwise, the number of important nodes follows the same trend as for previous neural network models. However, the number of worse nodes in the model behaves very differently from the previous models. They first increase (except for layer five where they decrease if we ignore the first result) till reaching a maximum at a batch size of 256, at which point it starts decreasing again. Layer five is the only layer that follows the trend of the layers in the previous models, it is also the only dense layer, which is the only type of layers considered in the previous models. From that, we can theorize that batch size affects dense and convolutional layers in different ways when it comes to worse nodes, but relatively similarly when it comes to important nodes.

When looking at Figure 4.27, we see that the first and last layers have the lowest node importance on average, with the last layer having the lowest node importance on average. The fact that node importance is at its lowest in the first and last layer, could mean that most of the classification is done there. However, one thing to take into account is that these are also the layers with the least nodes, this could lead to inflated results for those layers. The results shown in Figure 4.29 are very surprising since the average node importance of worse nodes is relatively stable across batch sizes except for when the batch size is 256. This also happened when training the single-layer ANN on the Fashion MNIST set. We had considered it not to be too significant then. However, it has happened again for the CNN, and what is more, it happens for all five layers. The reason for this is hard to tell and would need to be investigated more. A final note on this is that this discrepancy for a batch size of 256 does not happen when the CNN is trained on MNIST or Fashion MNIST.

As for Figure 4.30, we have very similar trends to the previous neural network models. Again, we get our best accuracy/loss for a batch size of 32 (see Table 4.17 and Table 4.18). The results for the other datasets are very similar except that the number of worse nodes maximizes at a batch size of 32 and, as mentioned before, there is no discrepancy when the batch size is 256. Moreover, the average node importance of important nodes follows the trend seen in the previous neural network models, with very low node importance when the batch size is one, before being relatively stable or slowly decreasing as the batch size increases.

4.3.4 Summary

From our results, we can see that the batch size affects node importance. Whether it be in a single-layer ANN, an MLP, or a CNN. In general, the larger the batch size, the more important nodes there are in the model. When it comes to worse nodes, if we consider dense layers, then the number of worse nodes decreases as we increase batch size. However, for convolutional layers, the number of worse nodes starts by increasing before reaching a maximum and then decreasing. For the average node importance of important nodes, in most cases, we have the lowest node importance when the batch size is one (the notable exception being for the CNN trained on CIFAR-10, where it was unable to classify when the batch size is one). Before stabilizing or slightly decreasing as we increase batch size. As for the average node importance of worse nodes, it stays relatively similar for all batch sizes except for the CNN trained on CIFAR-10 and the single-layer ANN trained on Fashion MNIST where we have a sudden increase in node importance for batch size 256. This increase at batch size 256 is hard to explain with our current results and would require more investigation. Otherwise, the zero nodes decrease and then stabilize as we increase batch size, and overall we achieve the best accuracy/loss with a batch size of 32.

In conclusion, the best batch size to use for all datasets and neural network models seems to be 32 because it both yields a good number of worse nodes compared to the other batch sizes and has a good initial loss/accuracy. Even though looking at batch size 256 for a CNN trained on CIFAR-10 to see how the discrepancy affects pruning would be interesting. To keep results consistent between neural network models and datasets, we will use a batch size of 32 going forward.

4.4 Effects of Using Dropout

We will now look at whether adding dropout to each layer affects node importance. We will only focus on the single-layer ANN and MLP neural networks. Again we will calculate node importance based on the validation set. We will test five different dropout rates: 0.1, 0.3, 0.5, 0.7, and 0.9. All the models will be trained for five epochs.

4.4.1 Single-layer ANN

As for the batch size experiment, we will train 25 models for each dropout rate. For the results without dropout, we will refer to section 4.2 and section 4.3 (the results when batch size is equal to 32). We will do this for both the MNIST and Fashion MNIST datasets, but we will only discuss MNIST for the single-layer ANN. For the results on Fashion MNIST or more detailed statistics, refer to section B.3.

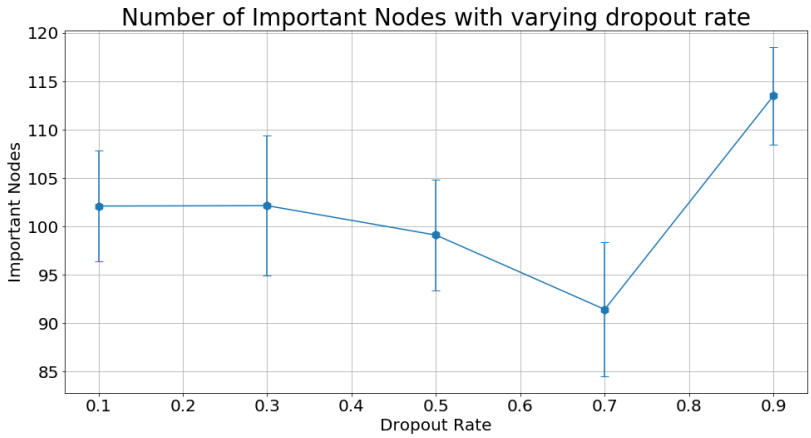

Figure 4.31: Average number of important nodes at each dropout rate for 25 single-layer ANNs trained on the MNIST dataset

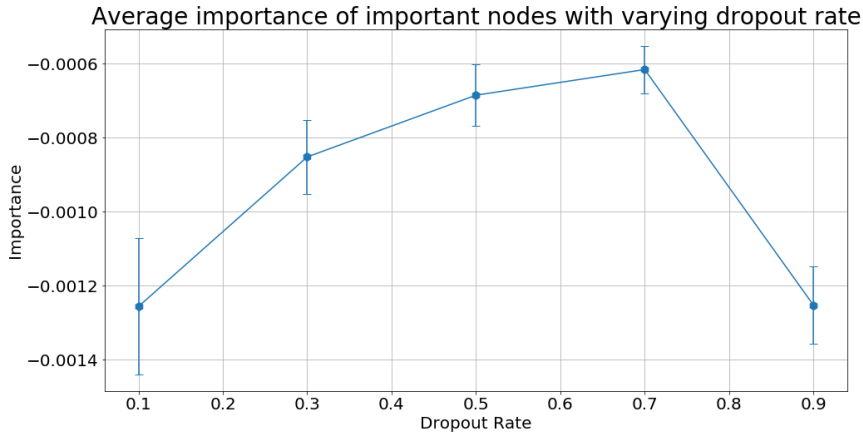

Figure 4.32: Average node importance of important nodes at each dropout rate for 25 single-layer ANNs trained on the MNIST dataset

Looking at Figure 4.31, we see that the number of important nodes starts by decreasing till a dropout rate of 0.7 before increasing significantly when the dropout rate is 0.9 (goes from 92 to 113). A possible reason for this might be a specialization of nodes to certain classes once the number of nodes trained on at each step becomes much smaller. This

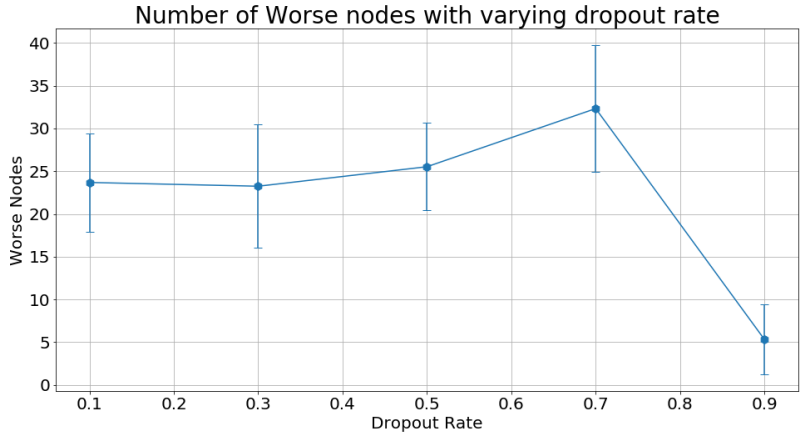

Figure 4.33: Average number of worse nodes at each dropout rate for 25 single-layer ANNs trained on the MNIST dataset

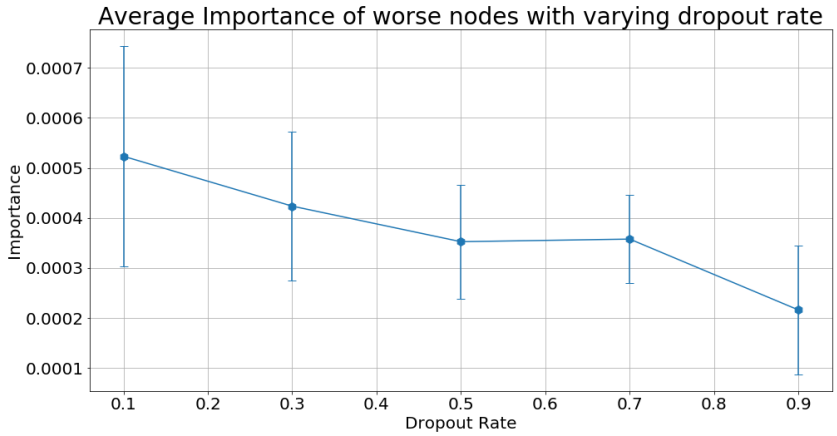

Figure 4.34: Average node importance of worse nodes at each dropout rate for 25 single-layer ANNs trained on the MNIST dataset

	0.1	0.3	0.5	0.7	0.9
mean	0.9761	0.9749	0.9710	0.9606	0.9293
std	0.0015	0.0012	0.0010	0.0013	0.0022
min	0.9710	0.9723	0.9689	0.9579	0.9242
25%	0.9755	0.9742	0.9705	0.9597	0.9284
50%	0.9764	0.9748	0.9710	0.9609	0.9299
75%	0.9771	0.9759	0.9717	0.9617	0.9308
max	0.9775	0.9767	0.9725	0.9629	0.9334

Table 4.19: Statistics on the accuracy of the 25 single-layer ANNs trained on the MNIST dataset at each dropout rate

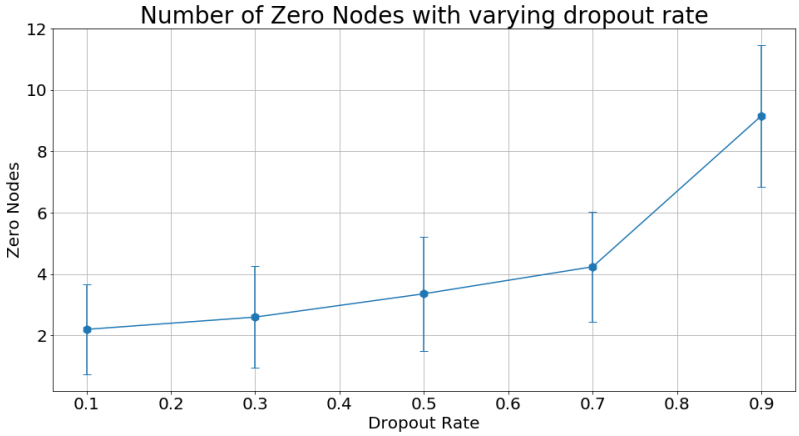

Figure 4.35: Average number of zero nodes at each dropout rate for 25 single-layer ANNs trained on the MNIST dataset

	0.1	0.3	0.5	0.7	0.9
mean	0.0775	0.0822	0.0962	0.1333	0.2714
std	0.0044	0.0033	0.0027	0.0032	0.0076
min	0.0713	0.0753	0.0913	0.1261	0.2612
25%	0.0750	0.0802	0.0944	0.1313	0.2641
50%	0.0765	0.0819	0.0957	0.1331	0.2710
75%	0.0797	0.0842	0.0982	0.1363	0.2749
max	0.0925	0.0893	0.1034	0.1393	0.2926

Table 4.20: Statistics on the loss of the 25 single-layer ANNs trained on the MNIST dataset at each dropout rate

seems to be confirmed by Figure 4.32 where the node importance starts by increasing before dropping back down to around the node importance observed when the dropout rate is 0.1. The theory of node specializing is furthered by Figure 4.33, where the number of worse sharply decreases when the dropout rate is 0.9. Otherwise, it slowly increases as the dropout rate increases.

As for Figure 4.34 and Figure 4.35, the trends are pretty stable where the average node importance of worse nodes decreases while the number of zero nodes increases as the dropout rate increases. Finally, looking at Table 4.19 and Table 4.20, the best results are obtained when the dropout rate is 0.1. The models tend to get worse as we increase the dropout rate.

If we compare our results to the results without dropout, we see that apart from having a dropout layer with a dropout rate of 0.1, where we get a similar result as when there is no dropout, having a dropout layer seems to make results worse. Therefore, having no dropout seems to be a better ides for node pruning.

The results for the single-layer ANNs trained on Fashion MNIST are very similar, except that there are less important nodes and more zero and

worse nodes at each dropout rate.

4.4.2 MLP

We will again train 25 models for each dropout rate. In this experiment, we chose to apply the same dropout rate to every layer. Finally, we train both on the MNIST and Fashion MNIST dataset. However, we only discuss our result on the Fashion MNIST dataset. For our other result and more detailed statistics, refer to section C.3

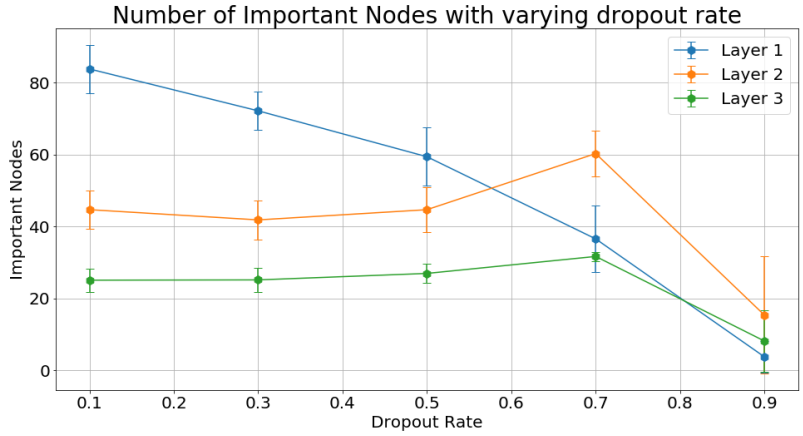

Figure 4.36: Average number of important nodes at each dropout rate and layer for 25 three-layer MLPs trained on the Fashion MNIST dataset

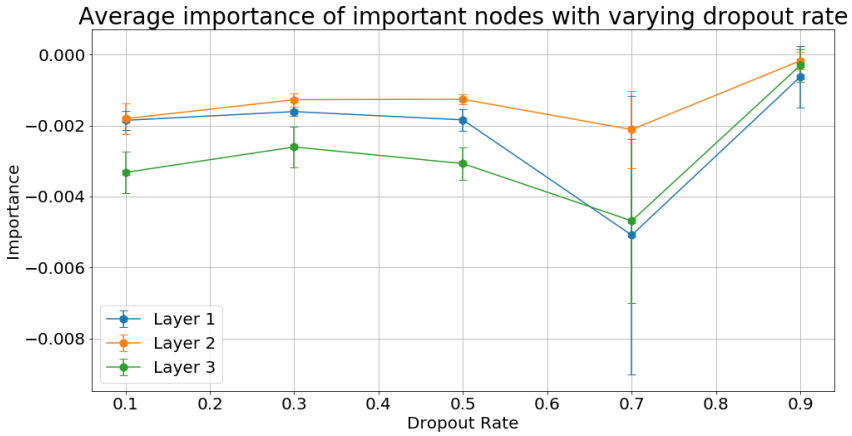

Figure 4.37: Average node importance of important nodes at each dropout rate and layer for 25 three-layer MLPs trained on the Fashion MNIST dataset

If we start by focusing on layers two and three in Figure 4.36, we see that instead of having a high number of important nodes with a dropout rate of 0.9, we maximize the number of important nodes at a dropout rate

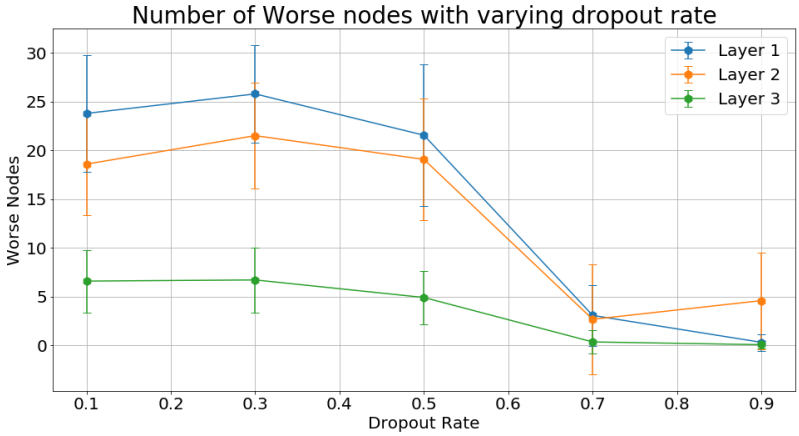

Figure 4.38: Average number of worse nodes at each dropout rate and layer for 25 three-layer MLPs trained on the Fashion MNIST dataset

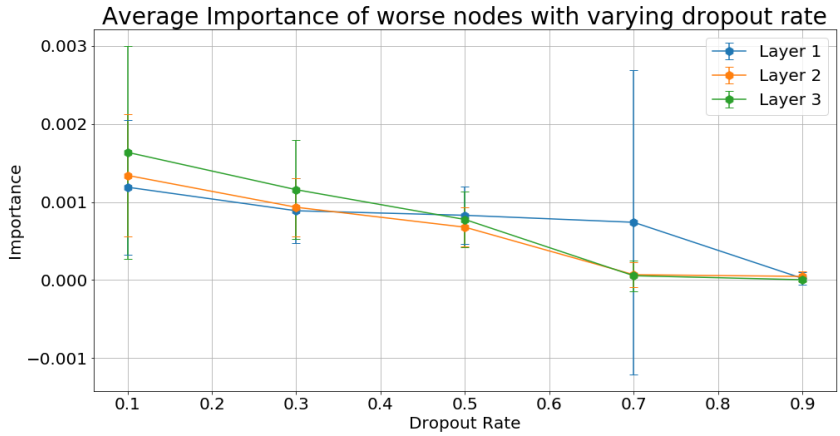

Figure 4.39: Average node importance of worse nodes at each dropout rate and layer for 25 three-layer MLPs trained on the Fashion MNIST dataset

	0.1	0.3	0.5	0.7	0.9
mean	0.8672	0.8552	0.8082	0.5525	0.1135
std	0.0043	0.0040	0.0198	0.0742	0.0187
min	0.8575	0.8433	0.7440	0.4459	0.1000
25%	0.8645	0.8532	0.8007	0.4869	0.1000
50%	0.8675	0.8568	0.8071	0.5501	0.1020
75%	0.8703	0.8581	0.8209	0.6084	0.1283
max	0.8742	0.8592	0.8308	0.6871	0.1518

Table 4.21: Statistics on the accuracy of the 25 three-layer MLPs trained on the Fashion MNIST dataset at each dropout rate

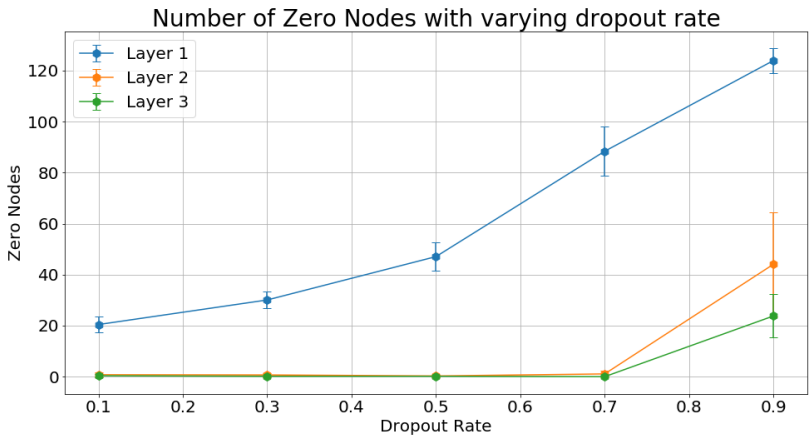

Figure 4.40: Average number of zero nodes at each dropout rate and layer for 25 three-layer MLPs trained on the Fashion MNIST dataset

	0.1	0.3	0.5	0.7	0.9
mean	0.3685	0.4043	0.5125	1.1333	2.2959
std	0.0113	0.0082	0.0247	0.1173	0.0123
min	0.3545	0.3902	0.4648	0.8891	2.2610
25%	0.3605	0.3984	0.4967	1.0668	2.2945
50%	0.3661	0.4032	0.5072	1.1045	2.3025
75%	0.3739	0.4086	0.5314	1.2491	2.3026
max	0.4037	0.4213	0.5569	1.3161	2.3027

Table 4.22: Statistics on the loss of the 25 three-layer MLPs trained on the Fashion MNIST dataset at each dropout rate

of 0.7. A good explanation for this can come from Table 4.21, where we see that at least 25% of the models trained with a dropout rate of 0.9 are guessing which class each image is in (and thus reaching an accuracy of 0.1). Therefore, having a dropout rate of 0.9 is too high and will often not let the model learn the patterns. Therefore, if we ignore that result, we see that layers two and three have similar behavior to the single-layer ANN. If we now focus on layer one, we see that the behavior is different where the number of important nodes decreases as the dropout rate increases. However, if we now look at Figure 4.37, we see that all the layers follow the same trend as for the single-layer ANN. Therefore our theory that nodes specialize seems to still be applied here. This is further supported by Figure 4.38, where the number of worse nodes increases until the dropout rate is 0.5, before dropping when the dropout rate is 0.7. This is the same behavior seen when using a single-layer ANN.

As for the node importance of worse nodes, looking at Figure 4.39, it seems to decrease as we increase the dropout rate. For layer one, it remains relatively constant as the dropout rate increases, while for layers two and three it decreases slowly at first before dropping significantly for a dropout

rate of 0.7. Finally, the number of zero nodes behaves in a very similar fashion as for a single-layer ANN when looking at layer one in Figure 4.25. However, for layers two and three, there seems to never be any zero nodes between a dropout rate of 0.1 and 0.7. Table 4.21 and Table 4.16 again shows that the model gets worse as we increase the dropout rate, and that overall, we have a slightly worse model to prune with a dropout rate of 0.1 compared to having no dropout layer.

The results for the MNIST dataset are similar to the previous methods as having a dropout rate of 0.9 being too high to train a decent model.

4.4.3 Summary

As for batch size, having dropout and different dropout rates affect the node importance of the nodes. Interestingly, in this case, it seems that at a certain dropout rate, we have node specializing in classifying certain classes and therefore increasing the number of important nodes and decreasing their node importance. We see that the behavior of a single-layer ANN can be seen in the two last layers of an MLP, and while the behavior of the first layer of the MLP is slightly different (the number of important nodes continuously decrease instead of significantly increasing at the highest usable dropout rate), overall the behavior is still very similar to a single-layer ANN.

Even though adding a dropout rate induces interesting patterns, the models almost always perform worse in both loss and accuracy. The only case where we get very similar results is when the dropout rate is 0.1, but since the result is just similar and not better, we will not have a dropout layer moving forward.

4.5 Pruning network with pre-calculated importance

This the first section in which we will prune our models based on their node importance. In this case, we will start by calculating their node importance and then removing all the nodes that are either classified as worse or zero. We will also do this for both node importance based on the training set and the validation set. As mentioned in section 4.3 and section 4.4, we will not have dropout and will train with a batch size of 32.

4.5.1 Single-layer ANN

For the single-layer ANN, we will train 50 models for each dataset (MNIST and Fashion MNIST) and each set (training and validation). We will only discuss the MNIST dataset results, but our results for the Fashion MNIST dataset and more statistics on our results can be found in section B.4. We will start by analyzing our results when estimating node importance on the training set.

As Figure 4.41 and Figure 4.42 mirror each other and that node importance is based on loss, we will focus Figure 4.42. The first thing we

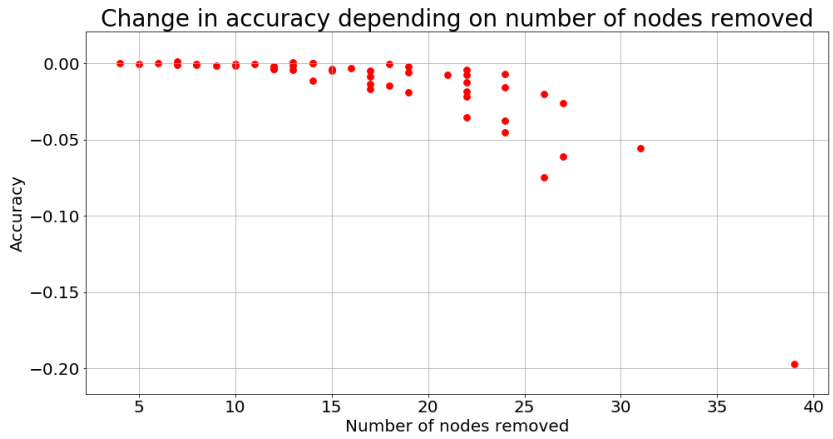

Figure 4.41: Impact of number of nodes removed based on their pre-calculated node importances, calculated on the training set, from single-layer ANNs trained on the MNIST dataset on the accuracy of the model

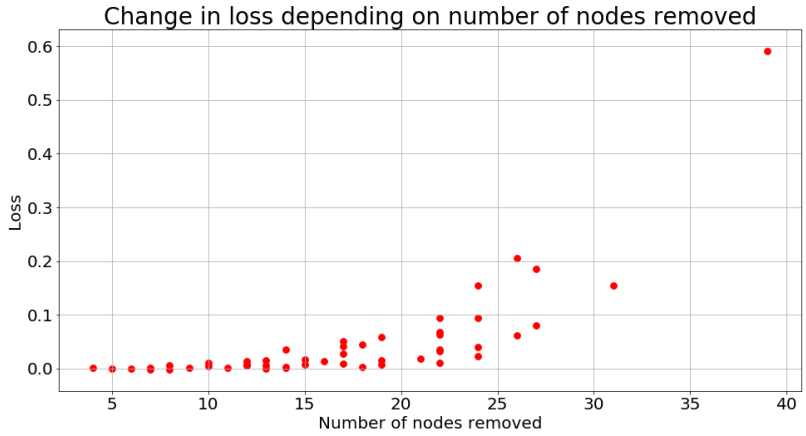

Figure 4.42: Impact of number of nodes removed based on their pre-calculated node importances, calculated on the training set, from single-layer ANNs trained on the MNIST dataset on the loss of the model

notice is that almost no model has improved after being pruned. The loss stays relatively the same up to 15 nodes pruned before it starts increasing exponentially. However, if we look at Table 4.23, we see that on average we are pruning 17 nodes, which does not seem to change the loss of the model by a significant margin. However, on average, the change loss from pruning based on pre-calculated node importance is 0.03542 (when ignoring the result when almost 40 nodes are removed, otherwise it is 0.04654), which, since the average loss of the model is around 0.081, is almost a 50% increase in loss. Therefore it seems that pruning based on pre-calculated node importance might not be a great way to prune our models. Next, we will look at how estimating the node importance on the validation

	Number of Nodes
mean	16.92
std	7.28
min	4.00
25%	12.00
50%	17.00
75%	22.00
max	39.00

Table 4.23: Statistics on the number of nodes pruned from single-layer ANNs trained on the MNIST dataset when pruning based on pre-calculated node importance calculated on the training set

set changes our results.

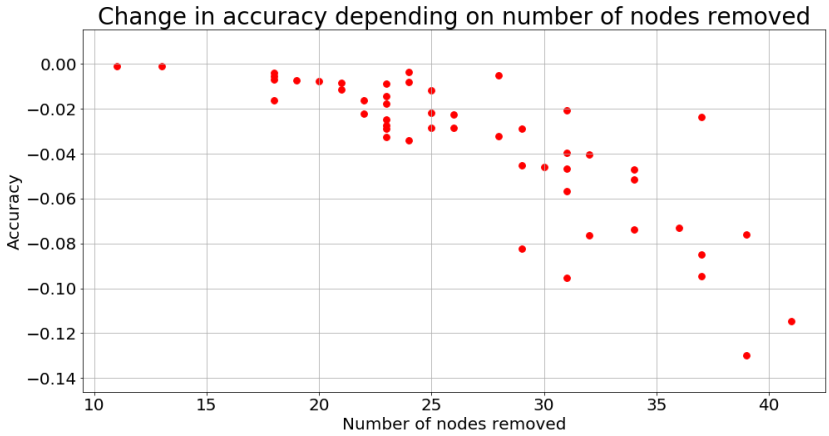

Figure 4.43: Impact of number of nodes removed based on their pre-calculated node importances, calculated on the validation set, from single-layer ANNs trained on the MNIST dataset on the accuracy of the model

	Number of Nodes
mean	26.82
std	6.89
min	11.00
25%	23.00
50%	25.50
75%	31.00
max	41.00

Table 4.24: Statistics on the number of nodes pruned from single-layer ANNs trained on the MNIST dataset when pruning based on pre-calculated node importance calculated on the validation set

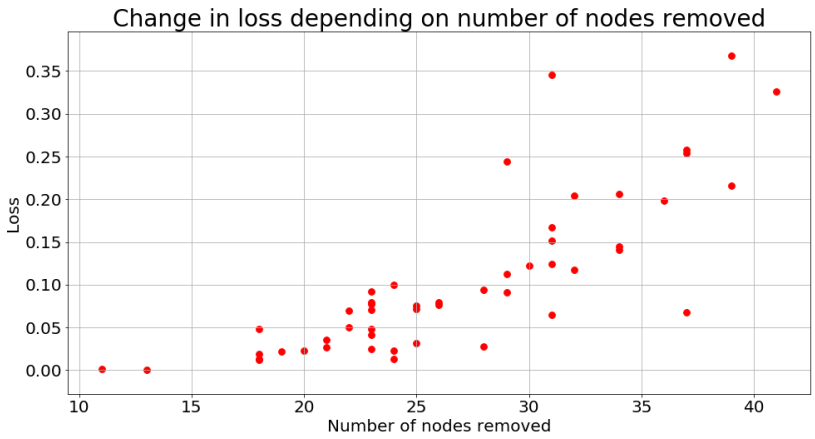

Figure 4.44: Impact of number of nodes removed based on their pre-calculated node importances, calculated on the validation set, from single-layer ANNs trained on the MNIST dataset on the loss of the model

As before, Figure 4.44 and Figure 4.43 mirror each and we will therefore focus on Figure 4.44. At first glance, our results look worse, with almost all pruned models having worse loss than the original model. However, if we focus on the x-axis and Figure 4.43, we see that we prune many more nodes compared to before. From Table 4.24, we see that we prune on average 27 nodes, which is ten more than when pruning based on node importance calculated on the training set. If we compare our result at each number of nodes removed, we see that we now have a smaller or similar change in loss, which could indicate that pruning on the validation set can lead to more removal, with better or similar results. This is especially true for the higher number of nodes removed, where we never increase our loss by more than 0.4, while before, at around 40 nodes, we increase our loss by almost 0.6 (although there is only a single data point there and could be an outlier). Finally, if we compare the increase in loss when removing between 25 and 30 nodes, we see that when using the validation set, we never increase the loss by much more than 0.1, while when we use the training set, we can increase the loss by 0.2. Therefore it seems that using the validation set to estimate node importance might be a better idea.

A reason that pruning based on pre-calculated node importance gives us models with worse or significantly worse loss could be that some nodes are redundant and therefore removing one of the redundant nodes does not affect our model much or even could make it better. However, if we remove all the nodes, then we also remove all the redundant nodes. In other words, we remove all the nodes doing the specific classification. Therefore it might be better to re-evaluate each time we remove a node.

For the Fashion MNIST dataset, we get very similar results except that more nodes can be removed. Also since the initial loss is higher, the percentage increase in the loss for the Fashion MNIST set is smaller than for the MNIST set. However, only two or three models have a

negligible increase in the loss while all the others have either a noticeable or significant change in the loss. Therefore, we come to the same conclusion that using pre-calculated node importance is not the best idea for node pruning.

4.5.2 Other models

We get similar results for MLP and CNN, where the pruning based on pre-calculated node importance generally leads to significantly worse models. The biggest differences are that more nodes are removed for those models since they have more nodes, to begin with. We also observe the same trend that when calculating the node importance on the validation set, we get more node removed (although that is expected since in section 4.2 we saw that there were more worse and zero nodes when estimating on the validation set), but we get a lower or similar increase when the same number of nodes are removed. The results for these models can be found in section C.4 and section D.3.

4.5.3 Summary

Using pre-calculated node importance to prune our models seemed to be a good idea but after experimenting with it, we saw that, in general, the pruned models end up having significantly worse loss and accuracy. A reason that might be the case is that a few nodes might be doing the same type of classification meaning that they are redundant in respect to one another. Therefore the node importance of each one of them classifies them as either worse or zero nodes. With our current way of pruning, we end up removing all of them instead of keeping one of them (which is needed since it does some type of classification). We also saw that pruning based on node importance calculated via the validation set gives us a similar or better result when the number of nodes removed is the same. Besides, it removes more nodes on average, although this is expected since we saw in section 4.2 that there are more worse and zero nodes when we use the validation set to estimate node importance.

4.6 Pruning Nodes based on the Loss

We saw that pruning nodes based on pre-calculated node importance does not yield very good results. To remedy this, we will try a new approach to pruning where we start by finding the nodes with the highest node importance in the layer, remove it, and then re-calculate the node importance of the other nodes. We do this till all the nodes in the layer are considered important and then move on to the previous layer. We do this till all hidden layers are pruned. We will again consider both node importance calculated on the training set and the validation set for the single-layer ANN and the MLP. For the CNN, we will only consider node importance calculated on the validation set.

4.6.1 Single-layer ANN

We will prune 20 different single-layer ANN models for both datasets (MNIST and Fashion MNIST) and both sets (training and validation). We will again only discuss our results for MNIST. More detailed statistics and the results for the Fashion MNIST dataset can be found in section B.5. We will start by analyzing the pruned models where pruning is based on the training set loss.

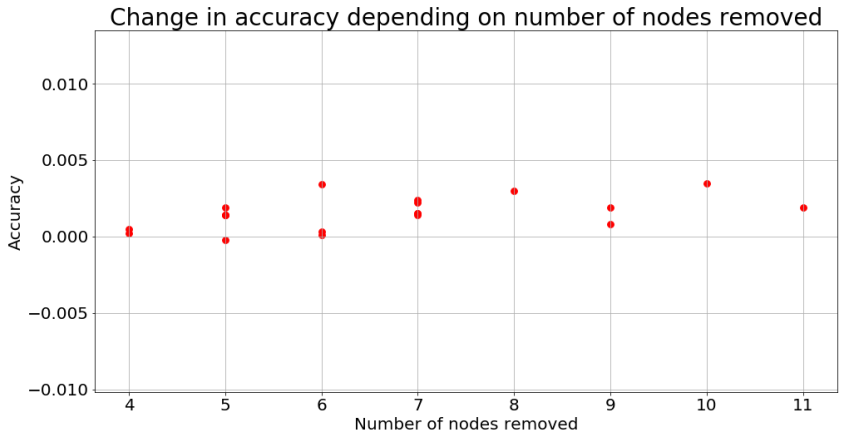

Figure 4.45: Impact of number of nodes removed based on their loss and pruned in an exhaustive fashion, calculated on the training set, from single-layer ANNs, trained on the MNIST dataset, on the accuracy of the model

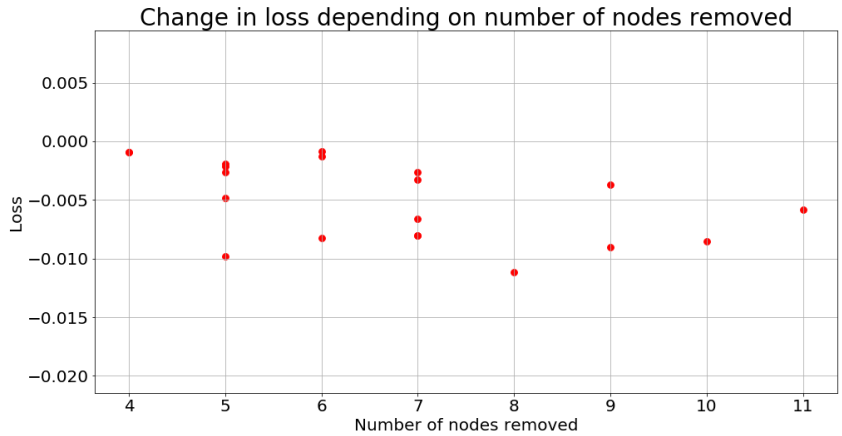

Figure 4.46: Impact of number of nodes removed based on their loss and pruned in an exhaustive fashion, calculated on the training set, from single-layer ANNs, trained on the MNIST dataset, on the loss of the model

The first thing we notice when looking at Figure 4.46 is that the number of nodes removed is much smaller than the estimated number of zero and worse nodes. Here, we have an average of seven nodes removed, whereas

section 4.2 estimates there to be 16 unimportant nodes. This seems to support our argument that some nodes are considered redundant and that removing them does not change or makes the model better. We can also see that the model always improves (albeit slightly for some cases). On average, we reduce the test loss by 0.005, which is around a 7% decrease in loss. A final observation to make is that the decrease in loss seems to slightly increase as we remove more nodes, although it is not a big increase, and we could just as well have it be stable across the number of nodes removed. Further, since we only look at the difference in loss and not the final and initial loss, it could be that those with more nodes removed start at a higher initial loss. For the accuracy, we see that Figure 4.45 also shows an average increase in accuracy, but it is usually less than half a percent and therefore not very consequential. If we compare these results to the ones we obtain using random removal (section 4.1), we see that we obtain similar results in terms of the number of nodes removed (seven versus nine), but we get a higher reduction in the loss on average. Furthermore, the big difference is that here we use different models, while we only have one model for when randomly removing. Therefore, to get similar results, we would have to hope that we choose the right variation, or we need to try many variations to find a sensible one. Therefore, using this method leads to more stable results across models than randomly removing. We will now look at how the pruning differs when we consider node importance calculated on the validation set.

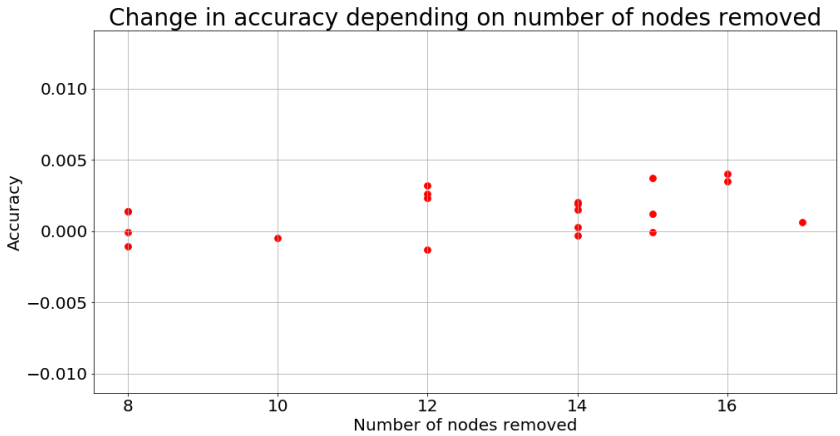

Figure 4.47: Impact of number of nodes removed based on their loss and pruned in an exhaustive fashion, calculated on the validation set, from single-layer ANNs, trained on the MNIST dataset, on the accuracy of the model

Looking at the x-axis in Figure 4.48, we see that we now remove more nodes than when considering node importance calculated on the training set. We have, on average, 13 nodes removed instead of the seven previously. However this still less than the estimated number of zero and worse nodes estimated (in this case 27). Moreover, it looks like we

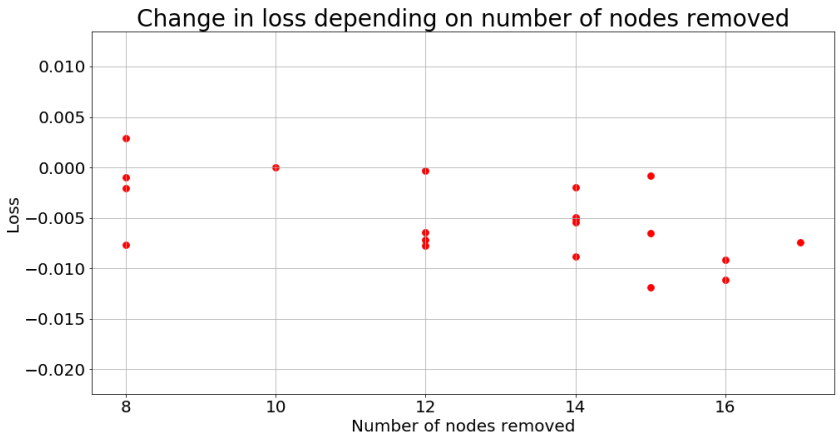

Figure 4.48: Impact of number of nodes removed based on their loss and pruned in an exhaustive fashion, calculated on the validation set, from single-layer ANNs, trained on the MNIST dataset, on the loss of the model

have a few models that end up having worse performance than before being pruned. However, we still, on average, decrease the loss by 0.005. However, the average loss of the models trained with a validation set is 0.004 higher. Therefore, while using the validation set prunes more of the model, the end model tends to have a higher loss than if we only had a training set. However, for more complex models, we always tend to have a validation model so that we can make sure we do not overfit a model. While the end loss is higher, the amount of loss decreased is roughly the same (6% decrease instead of 7%). A final advantage is that since the validation set is much smaller than the training set, it takes less time to prune when calculating the node importance on the validation set.

One reason why we can remove more nodes without sacrificing performance in terms of loss reduction might be that when using the validation set, we can remove nodes that classify the noise of the training set. With the validation set, we have them classified as worse or zero nodes, while on the training set, they would be considered important nodes.

To end this section, we will look at the evolution of loss and accuracy for two single-layer ANNs, one where node importance is calculated on the training set, whereas the other is calculated on the validation set.

For the model pruned on the training set, we have:

- Original Loss: 0.0695
- Original Accuracy: 0.9783
- Number of nodes removed: 2
- New Loss: 0.06689
- New Accuracy: 0.9797

For the model pruned on the validation set, we have:

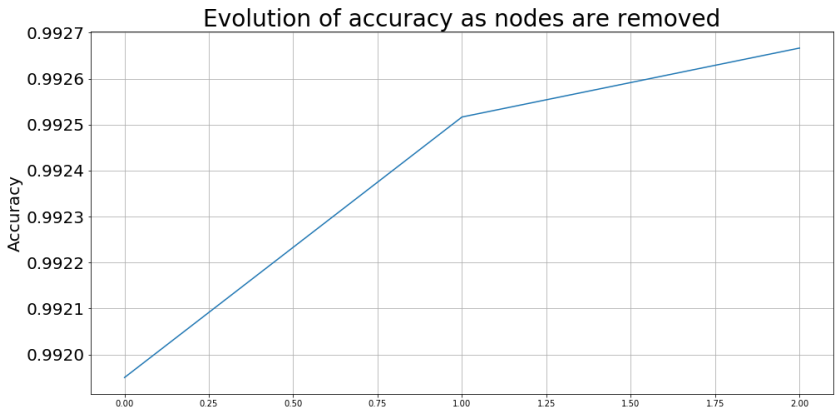

Figure 4.49: Evolution of the training accuracy during the exhaustive pruning of a single-layer ANN trained on the MNIST dataset

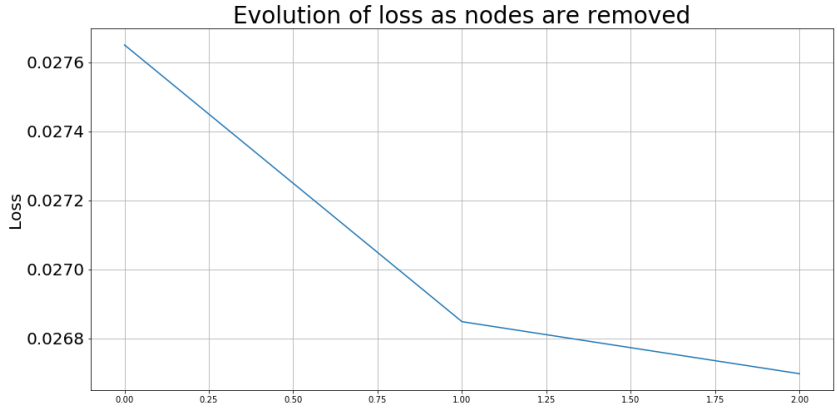

Figure 4.50: Evolution of the training loss during the exhaustive pruning of a single-layer ANN trained on the MNIST dataset

- Original Loss: 0.0833
- Original Accuracy: 0.9759
- Number of nodes removed: 15
- New Loss: 0.0773
- New Accuracy: 0.9768

We can see from both evolution curves that we start by decreasing loss faster before slowing down, which is to be expected given our method. As expected from our previous results, we get more reduction with the validation set than with the training set, but since the initial loss is quite different, the difference in the number of nodes removed is enhanced.

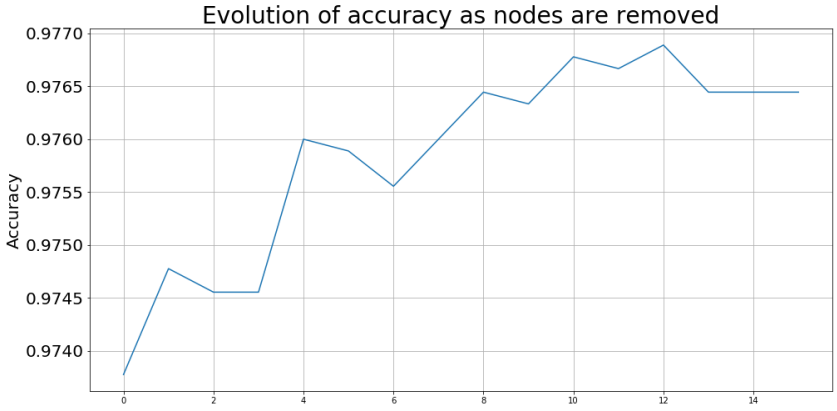

Figure 4.51: Evolution of the validation accuracy during the exhaustive pruning of a single-layer ANN trained on the MNIST dataset

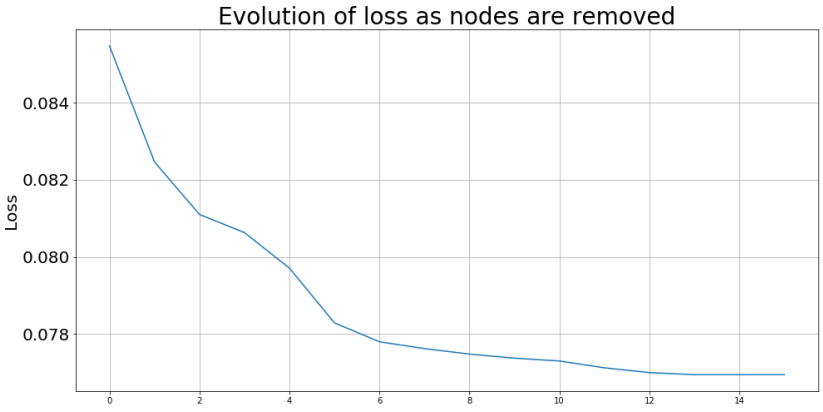

Figure 4.52: Evolution of the validation loss during the exhaustive pruning of a single-layer ANN trained on the MNIST dataset

Otherwise, both models increase the accuracy, but, for the models pruned on the validation data, we have a much less smooth increase in the accuracy.

We get very similar results for the Fashion MNIST data, except that the number of nodes removed and the loss decreased (in value at least) is larger. The biggest difference is that the loss reduction for nodes pruned on the validation set is noticeably larger than the nodes pruned on the training set (0.021 vs. 0.014 reduction in loss).

4.6.2 MLP

Now we go on to prune MLP models, instead of testing 20 models, we will now test ten different models. We will again test both on MNIST and

Fashion MNIST and both on the training set and the validation set. As for previous sections, we will only discuss our results for the Fashion MNIST dataset, but our result for the MNIST dataset and more statistics on our results can be found in section C.5. We will begin with pruning based on the training set.

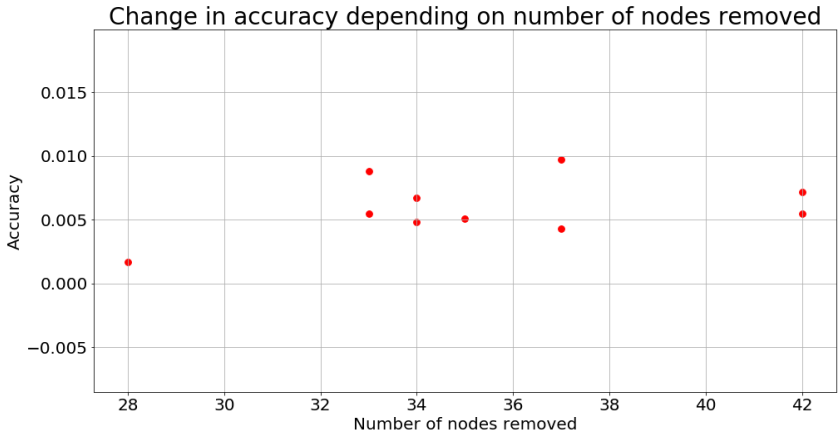

Figure 4.53: Impact of number of nodes removed based on their loss and pruned in an exhaustive fashion, calculated on the training set, from three-layer MLPs, trained on the Fashion MNIST dataset, on the accuracy of the model

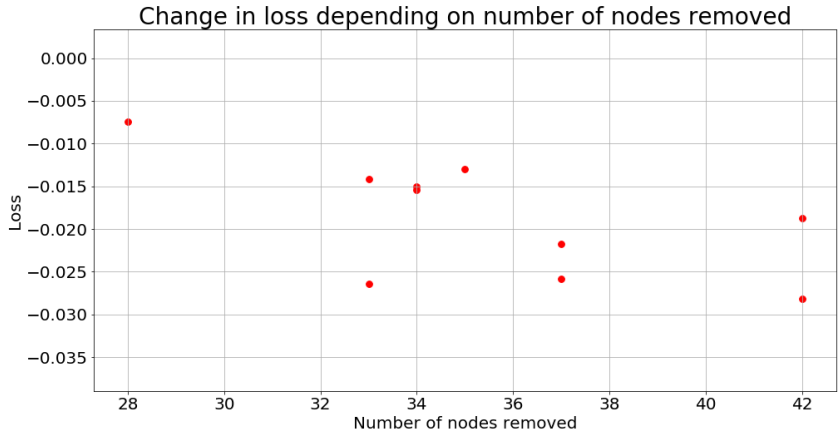

Figure 4.54: Impact of number of nodes removed based on their loss and pruned in an exhaustive fashion, calculated on the training set, from three-layer MLPs, trained on the Fashion MNIST dataset, on the loss of the model

From Table 4.25, we can see that we remove fewer nodes than the estimated number of worse and zero nodes, both in terms of the total number of nodes removed (36 pruned versus 59 estimated) and per layer (23 instead of 36 for layer one, 8 instead of 16 for layer two, 5 instead of 6 for layer three). While the proportion of unimportant nodes is lower, the

	Layer 3	Layer 2	Layer 1
mean	4.60	7.90	23.00
std	1.84	2.28	3.53
min	2.00	6.00	19.00
25%	3.25	6.00	20.00
50%	4.50	7.50	22.50
75%	5.00	8.75	25.50
max	8.00	13.00	30.00

Table 4.25: Statistics on the number of nodes pruned from three-layer MLPs trained on the Fashion MNIST dataset when pruning exhaustively based on node importance calculated on the training set

first layer still has the highest proportion of unimportant nodes. However, the last and second layers flip, with the last layer now having a higher proportion than the second layer. Nevertheless, compared to single-layer ANN, the number of nodes we remove is closer to the estimate, around 60% here instead of 50% for the single-layer ANNs. Moreover, the number of nodes removed for layer three is very close to the number of worse and zero nodes estimated. As for the single-layer ANN, Figure 4.54 shows that the decrease of loss is relatively similar across the number of nodes removed, except when 28 nodes are removed where the decrease in loss is around 0.0075 instead of the average of 0.0186. However, this might be due to a lower initial loss for this model compared to the other models tested. As for accuracy, Figure 4.53 shows that we at most increase the accuracy by 1%, meaning that, while the accuracy does increase, it is not significant.

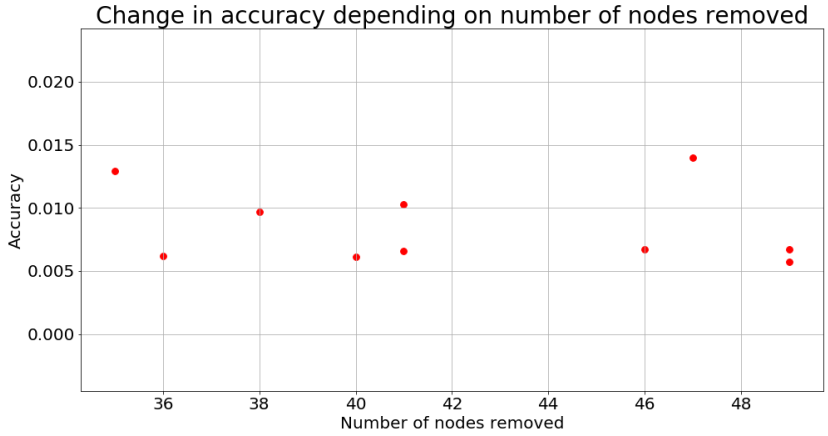

Figure 4.55: Impact of number of nodes removed based on their loss and pruned in an exhaustive fashion, calculated on the validation set, from three-layer MLPs, trained on the Fashion MNIST dataset, on the accuracy of the model

For the validation data, we have very similar results in terms of the

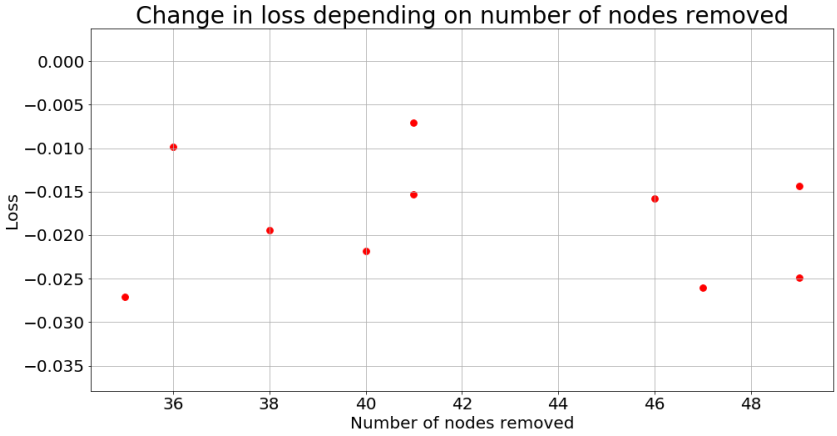

Figure 4.56: Impact of number of nodes removed based on their loss and pruned in an exhaustive fashion, calculated on the validation set, from three-layer MLPs, trained on the Fashion MNIST dataset, on the accuracy of the model

	Layer 3	Layer 2	Layer 1
mean	6.40	9.90	25.90
std	1.71	3.45	3.78
min	3.00	6.00	20.00
25%	6.00	8.00	22.75
50%	6.00	9.50	26.00
75%	7.75	10.00	28.00
max	9.00	18.00	31.00

Table 4.26: Statistics on the number of nodes pruned from three-layer MLPs trained on the Fashion MNIST dataset when pruning exhaustively based on node importance calculated on the validation set

number of nodes removed compared to the number of estimated worse and important nodes. Instead of around 60% of the estimated nodes are pruned, we have around 55% of estimated nodes pruned with again the 3^{rd} layer highest percentage of estimated nodes pruned (75% on average, all of these are based on the number of estimated nodes and not their exact positions). We again have a very similar experience to the single-layer ANN with nodes pruned based on the validation set compared to nodes pruned based on the training set. Where we have more nodes pruned when using the validation set against using the training set, but we have a similar decrease in loss, 0.0182 for the former versus 0.0186 for the latter (or around 5% for both). However, we again have a lower initial loss for the model trained on a full training set compared to a reduced one. Since we do get similar performance, and the difference in loss is not extreme and the gain in time to prune and the number of nodes removed is substantial, using the validation set seems to be a good idea.

We will now look at the evolution of loss during pruning of two MLP models, one pruning based on the training set while the other prunes are based on the validation set.

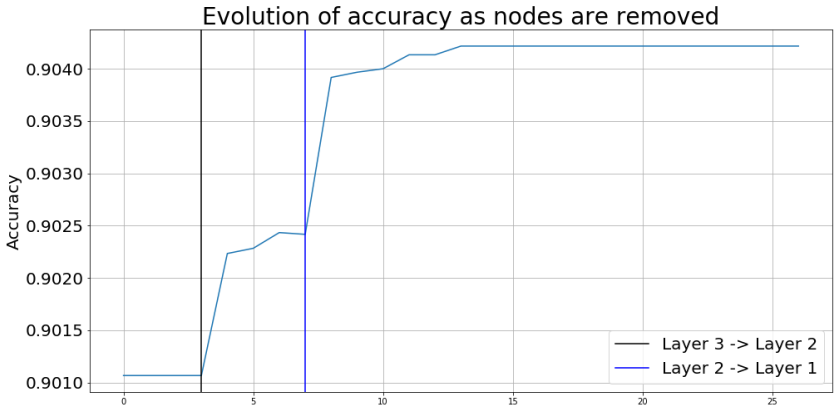

Figure 4.57: Evolution of the training accuracy during the exhaustive pruning of a three-layer MLP trained on the Fashion MNIST dataset

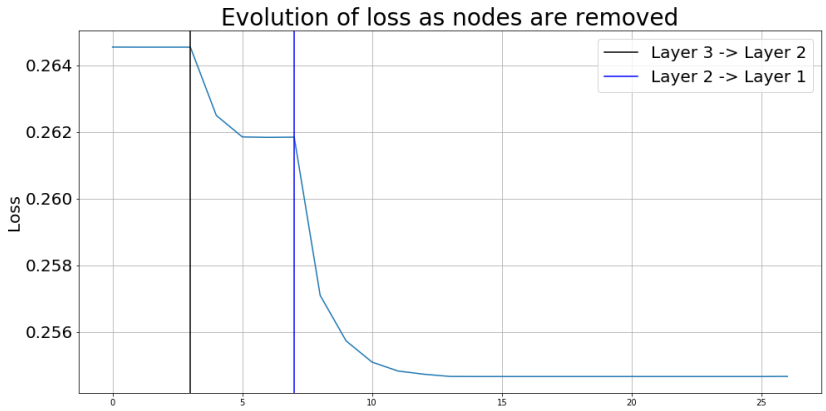

Figure 4.58: Evolution of the training loss during the exhaustive pruning of a three-layer MLP trained on the Fashion MNIST dataset

For the model pruned on the training set, we have:

- Original Loss: 0.3453
- Original Accuracy: 0.8785
- Number of nodes removed (layer 3): 3
- Number of nodes removed (layer 2): 4
- Number of nodes removed (layer 1): 19

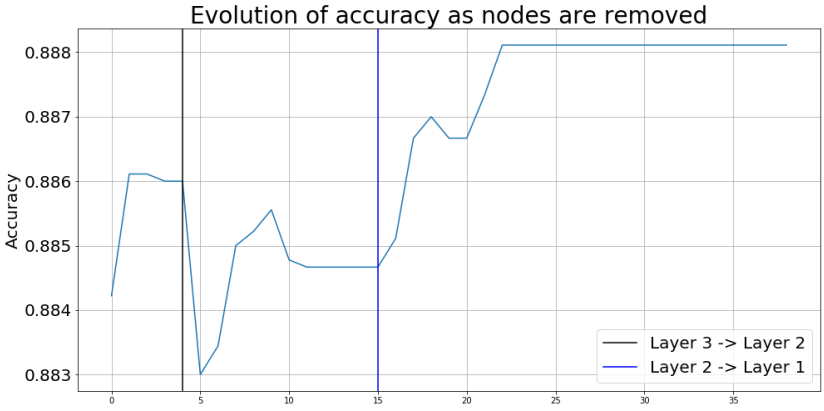

Figure 4.59: Evolution of the validation accuracy during the exhaustive pruning of a three-layer MLP trained on the Fashion MNIST dataset

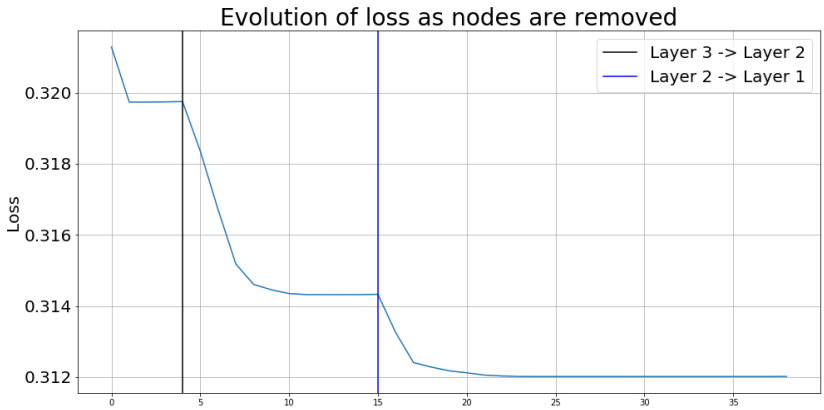

Figure 4.60: Evolution of the validation loss during the exhaustive pruning of a three-layer MLP trained on the Fashion MNIST dataset

- New Loss: 0.3282
- New Accuracy: 0.883

For the model pruned on the validation set, we have:

- Original Loss: 0.3531
- Original Accuracy: 0.8707
- Number of nodes removed (layer 3): 4
- Number of nodes removed (layer 2): 11
- Number of nodes removed (layer 1): 23
- New Loss: 0.3456

- New Accuracy: 0.8744

As previously, we see that while the evolution of the loss and accuracy is relatively smooth on the training set, only the evolution of loss is smooth on the validation set. The evolution of accuracy jumps around but does settle at a higher point. A notable difference in the evolution of loss between the training set and validation set is that most of the reduction in loss happens on the first layer for the training set, while it happens on the second layer for the validation set. This is probably due to the model differences and not a systematic difference between the two types of pruning. Although, more research is needed to understand this in more detail. Finally, it seems that the majority of nodes removed in the first layer are zero nodes rather than worse nodes.

4.6.3 CNN

We experimented with the CNN model too. However, due to the amount of time it takes to prune one model based on the training set, we decided to only do the pruning based on the validation. Using the two previous models, we see that we still get good results pruning on the validation set, even if we have a higher initial loss, we tend to reduce more nodes without sacrificing too much performance. Moreover, it is much faster to prune based on the validation data compared to the training data. Furthermore, we only test five models for each dataset, MNIST, Fashion MNIST, and CIFAR-10.

For our result, we have very similar results to previous architectures. We have fewer nodes pruned than the number of estimated worse and zero nodes, with the last layer having the closest number of nodes pruned to the estimated number of worse and zero nodes. Further, suppose we order the layers based on the proportion of nodes removed. In that case, we see that we get the same order as for the estimation of node importance, where the deeper in the model we get, the larger the proportion of removed nodes (or estimated zero and worse nodes). Moreover, if we ignore the last layer, the differences in the proportion of nodes removed at each layer is similar to the difference in the number of estimated zero and worse nodes between layers. Our full results can be found in section D.4.

4.6.4 Summary

From our results, we see that the most significant difference between pruning based on the training or validation set comes from the number of nodes pruned. There is also the difference in start loss, which makes pruning on the training set slightly more performant than pruning on the validation set. However, considering the difference in speed in pruning on the validation set against the training set and that we do not only care about performance but also reducing the models the most possible, using the validation set to prune the models seems to an adequate solution to pruning.

Even though pruning with the validation set is faster than pruning with the training set, it is still relatively slow once we start adding layers. Therefore, we will consider a new approach to pruning that should speed up our pruning while keeping a similar performance in terms of decreased loss and number of nodes removed.

4.7 Greedy approach to pruning instead of Exhaustive approach

To try to reduce the amount of time it takes to prune nodes, we propose a new approach where we search for the first zero or worse node we run across, we remove it, and restart searching the layer from the start. As before, we do this layer by layer and in a backward fashion, starting with the last layer and moving back to the first layer. Another time-saving feature of this algorithm is that we ignore in future passes any node that has a node importance under a user-defined threshold. In this experiment, we try with two different thresholds, $-1e^{-2}$ and $-1e^{-3}$. Moreover, since pruning on the single-layer ANN is relatively quick, we only consider the MLP and CNN models. Further, based on our results from section 4.6 and that our goal is to reduce the time it takes to prune, we will only consider node importance based on the validation set. From now on, we will refer to this method as greedy, while referring to the one used in section 4.6 as exhaustive.

4.7.1 MLP

We will start by experimenting with our MLP model. We will prune ten different MLP models with the same architecture for each dataset (MNIST and Fashion MNIST) and each threshold ($-1e^{-2}$ and $-1e^{-3}$). We will only discuss the Fashion MNIST dataset in detail. For more results and detailed statistics, check section C.6. We will start looking at our results using a threshold of $-1e^{-2}$.

From Figure 4.61, we see that we remove slightly more nodes using the greedy method (two more nodes on average). However, we have a slightly lower decrease in loss (0.0022 or an 18% less decrease compared to the exhaustive method. If we now look at our result when using a threshold of $-1e^{-3}$ (Figure 4.63), we see that we get a comparable increase in the number of nodes removed (again two on average) and also a similar lower decrease in loss (0.0023 or 19% less decrease). Therefore, using either threshold delivers similar performances.

Now that we have discussed the performance of the greedy method, we should focus on Table 4.27 to see if the method is faster and by how much. The greedy method is much faster than the exhaustive method, with the lower threshold pruning the model 2.5 times faster (103 seconds instead of 256 seconds), while the higher threshold prunes six times faster (40 seconds instead of 256 seconds). Since the performance for each threshold is similar, using the higher threshold is a better idea. Therefore, to finish, we will look

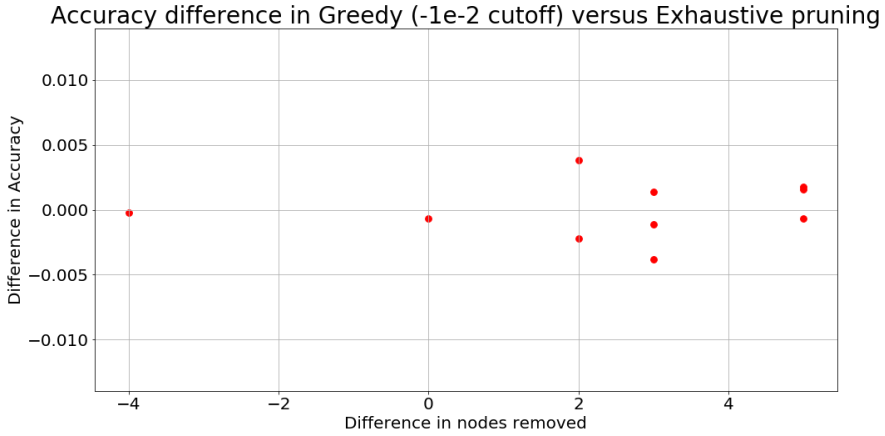

Figure 4.61: Impact of number of nodes removed based on their loss and pruned in an greedy fashion with an ignore cutoff of $-1e^{-2}$, calculated on the validation set, from three-layer MLPs, trained on the Fashion MNIST dataset, on the accuracy of the model

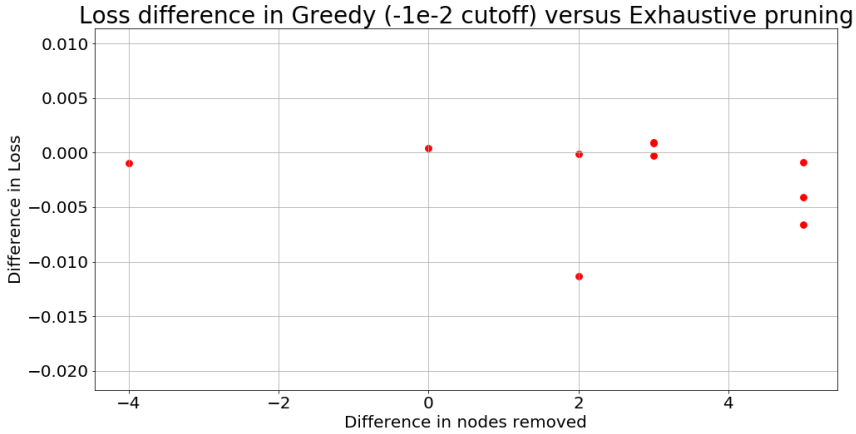

Figure 4.62: Impact of number of nodes removed based on their loss and pruned in an greedy fashion with an ignore cutoff of $-1e^{-2}$, calculated on the validation set, from three-layer MLPs, trained on the Fashion MNIST dataset, on the loss of the model

at how the loss and accuracy evolve as we remove nodes using the greedy method with a threshold of $-1e^{-3}$. Our results are as follow:

- Original Loss: 0.3681
- Original Accuracy: 0.8661
- Number of nodes removed (Layer 1): 6
- Number of nodes removed (Layer 2): 16
- Number of nodes removed (Layer 3): 24

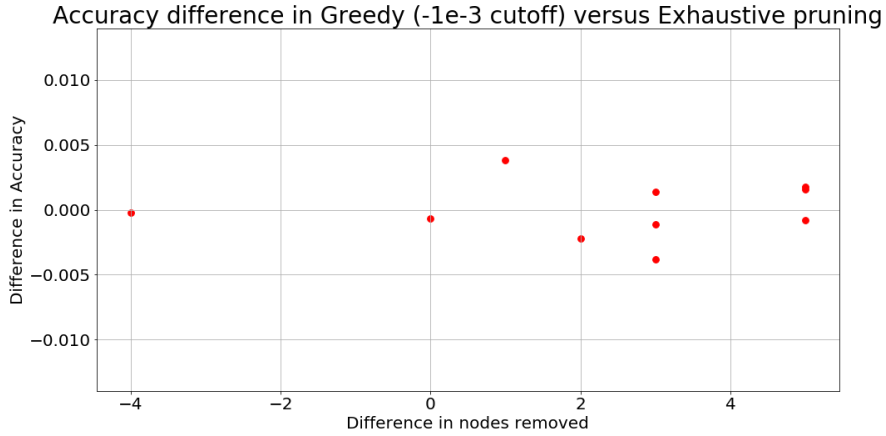

Figure 4.63: Impact of number of nodes removed based on their loss and pruned in an greedy fashion with an ignore cutoff of $-1e^{-3}$, calculated on the validation set, from three-layer MLPs, trained on the Fashion MNIST dataset, on the accuracy of the model

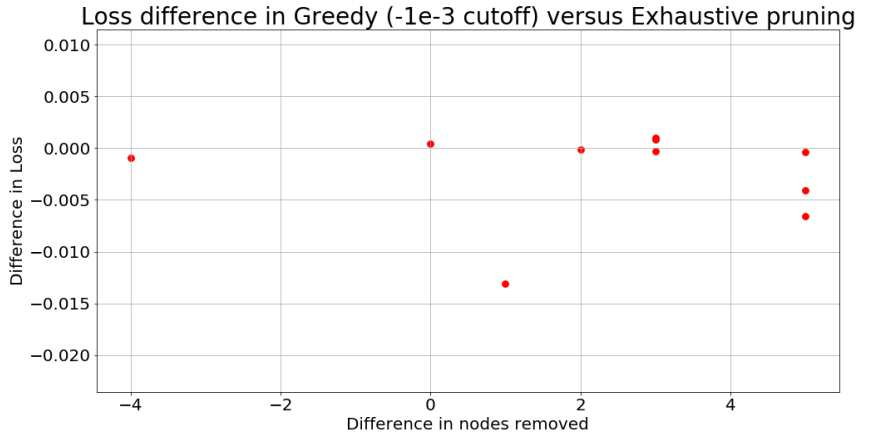

Figure 4.64: Impact of number of nodes removed based on their loss and pruned in an greedy fashion with an ignore cutoff of $-1e^{-3}$, calculated on the validation set, from three-layer MLPs, trained on the Fashion MNIST dataset, on the loss of the model

- New Loss: 0.3456
- New Accuracy: 0.8771

Contrarily to pruning with the exhaustive method, the evolution of loss using the greedy method is very erratic, with the rate of loss decrease randomly changing. However, this is to be expected since we remove the first worse or zero nodes we encounter.

For the MNIST dataset, the results are a bit more mitigated. While the number of nodes removed is slightly higher (five on average), the decrease

	Exhaustive	Greedy (-1e-2 cutoff)	Greedy (-1e-3 cutoff)
mean	256.68	103.87	40.89
std	38.38	11.50	5.75
min	186.69	85.84	33.24
25%	239.47	94.70	35.96
50%	255.32	104.55	41.23
75%	277.03	114.04	44.40
max	321.65	119.49	49.47

Table 4.27: Time taken in seconds to prune MLPs trained on the Fashion MNIST dataset using two different ignore cutoff points (if a node has an importance inferior to the cutoff, it will not be re-evaluated in future passes of the algorithm)

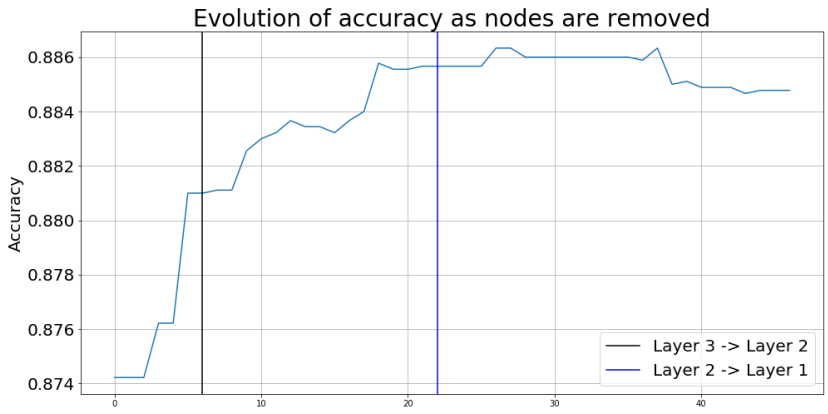

Figure 4.65: Evolution of the validation accuracy during the greedy pruning with a ignore cutoff of $-1e^{-3}$ of a three-layer MLP trained on the Fashion MNIST dataset

in loss is substantially smaller (0.0099 versus 0.0066 or a 33% less decrease in loss). However, the increase in speed is only slightly smaller, with the greedy method with the smaller threshold going five times faster, while the other goes two times faster. Therefore, it is a toss-up on whether to use the greedy method to prune for the MNIST dataset or the exhaustive method. It will depend on whether it is reasonable to sacrifice a noticeable amount of performance to go faster, but since the MNIST is a relatively simple dataset, taking only three minutes to prune exhaustively, we are not sure that the sacrifice in performance is worth it. However, using this method in combination with pruning on the training set might be interesting since the speed up in that case (assuming that it is similar or to the one when using the validation set) is more noticeable.

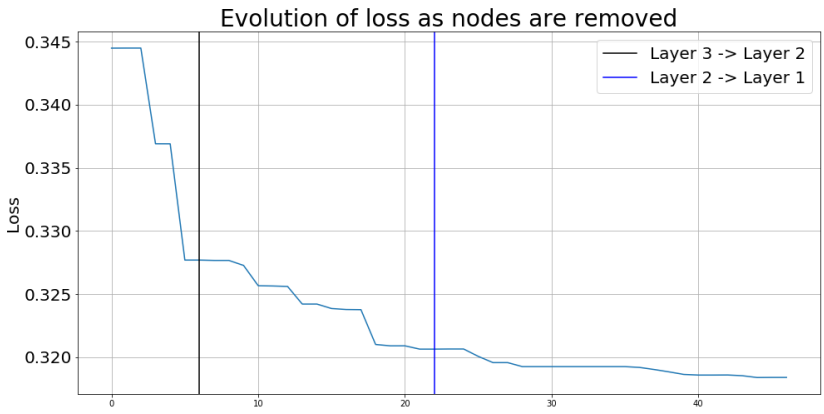

Figure 4.66: Evolution of the validation loss during the greedy pruning with a ignore cutoff of $-1e^{-3}$ of a three-layer MLP trained on the Fashion MNIST dataset

4.7.2 CNN

We repeated these experiments on CNN models, but instead of using ten models for each dataset, we only use five models for each dataset. On the CIFAR-10, we get slightly worse results for CNN, where the decrease in loss is smaller when using the greedy method compared to the exhaustive method (around 75% of the loss decrease). However, we do remove more nodes (14 more nodes removed on average), especially in the last convolutional layer (11 more nodes removed). In this case, we get better results using a threshold of $-1e^{-3}$ to ignore future nodes. For the Fashion MNIST, we get very similar results to the MLP, where the decrease in loss is very similar between the two methods. However, we do remove a couple more nodes (around ten more nodes removed). Finally, for MNIST, we again get pretty poor results using the greedy method. The decrease in loss is noticeably smaller, while the increase in the number of nodes removed is the smallest of the three datasets (around 6). However, the time saved using this method is more accentuated for the CNN, where the pruning was anywhere between 2.5 to 3.75 times faster when the threshold is $-1e^{-3}$ and 2.8 to 11 times faster when the threshold is $-1e^{-2}$, with the largest speed-up being observed for the CIFAR-10 dataset.

As for the evolution of loss, we again do not have a smooth decrease but random decrease increments, but this is to be expected with our pruning method. For all the results and more detailed statistics, refer to section D.5.

4.7.3 Summary

From our result, we saw that using the greedy method can give us anywhere from similar results to significantly worse results. However, the worse results seem to be on simpler datasets which usually take less time to run through. Further, the simpler dataset benefits the least from

the speed-up of using the greedy method. For a more complex dataset on a larger and deeper neural network, we saw a similar or slightly worse performance with a consequential speed-up (up to 11 times faster than the exhaustive method). Therefore, using this method for deeper neural networks with a more complex/larger dataset is useful since the small sacrifice in performance leads to a noticeable increase in speed. However, for the MNIST dataset and simpler datasets where the loss is smaller, it is possible that using a smaller threshold will lead to better results. Also as mentioned before, using this method in conjunction with pruning on the training set might lead to interesting results.

4.8 Looking at effects of per class accuracy after pruning

Up to this point, we have only looked at general accuracy, but since we are doing multi-class classification, it might be interesting to look at how node pruning affects class wise accuracy. We will also take this occasion to look at the accuracy and loss of the pruned model compared to the original model. For this experiment, we will only use CNN models to be pruned on. We will use the greedy method with a threshold of $-1e^{-3}$. For each dataset (MNIST, Fashion MNIST, and CIFAR-10), we will look at the ten different CNN models. While we have results for all datasets, we will only go into detail for the CIFAR-10 dataset. For the other results and more detailed statistics refer to section D.6.

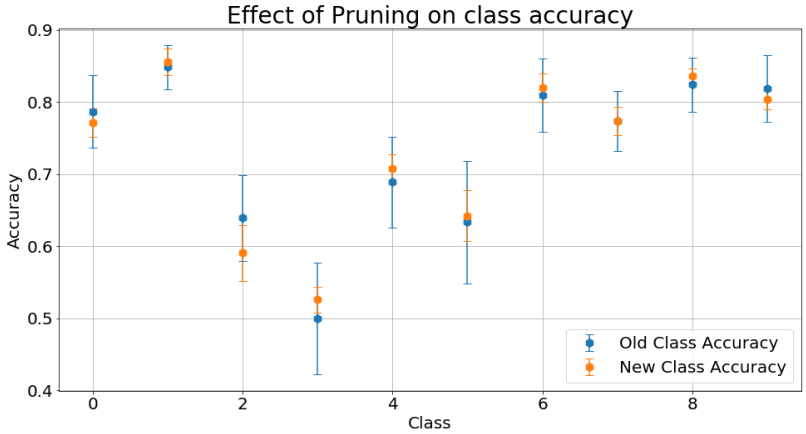

Figure 4.67: Average class accuracy before and after pruning ten CNNs trained on the CIFAR-10 dataset. These models were pruned based on the validation set and with the greedy method.

When looking at Figure 4.67 we see that while the class accuracy of each class is relatively similar (except for classes two and three or bird and cat), the error bars or standard deviation is much smaller for all classes. On average the standard deviation for the class accuracy before pruning

	Before Pruning	After Pruning
mean	0.7323	0.7328
std	0.0064	0.0058
min	0.7250	0.7256
25%	0.7274	0.7279
50%	0.7314	0.7326
75%	0.7344	0.7363
max	0.7455	0.7430

Table 4.28: Statistics on the accuracy before and after pruning of ten CNNs trained on the CIFAR-10 dataset. These models were pruned based on the validation set and with the greedy method.

	Before Pruning	After Pruning
mean	0.8011	0.7746
std	0.0238	0.0192
min	0.7598	0.7379
25%	0.7953	0.7645
50%	0.8076	0.7835
75%	0.8159	0.7855
max	0.8300	0.7946

Table 4.29: Statistics on the loss before and after pruning of ten CNNs trained on the CIFAR-10 dataset. These models were pruned based on the validation set and with the greedy method.

is 0.054, while after pruning it drops to 0.019, which is almost three times smaller. This reduction in standard deviation is not limited to the class accuracy, it also occurs for the loss, where it goes from 0.0238 to 0.0192 or a 19% drop, and for the accuracy, where it goes from 0.0064 to 0.0058 or a 9% drop. While neither is as impressive as for the class accuracy, they are still significant, at least for the loss. For the accuracy it is debatable. Therefore, even if the number of nodes removed is not stable, it seems that by pruning the models, we get more stable models that produce much similar results.

The other datasets show the same result but amplified, especially in the loss and accuracy department. For the MNIST dataset, we have a 77% drop in loss variation and a 71% drop in accuracy variation, while for Fashion MNIST we have a 62% drop in loss variation and a 44% drop in accuracy variation. It does seem that having a simpler model or one easier to train on leads to larger drops in variation. However, this might also due to how the threshold for the greedy algorithm and the zero node definition fits with the models. Although based on the previous section, we should see the opposite if we are looking at how well the greedy method works to prune these models.

To wrap-up, it looks like pruning mainly decreases variation in the model, with a more stable loss, accuracy, and class accuracy. Furthermore,

the simpler dataset seems to profit the most from this with a drop of at least 70% in the variation of loss and accuracy between models trained on the MNIST dataset. Digging more into this stability could be interesting. Moreover, inspecting how the exhaustive method or using the training set instead of the validation set affects the stability, could lead to even stabler models.

4.9 Iterative weight initialization using Node importance

Up to now, we have removed nodes after training, but what if we try to optimize the nodes before training such that the nodes are for the majority considered important. In this section, we will look into it and see if optimizing the weights before training leads to an improvement in performance.

4.9.1 Single-Layer ANN

We will try to optimize the weights of 25 different single-layer ANN models for both the MNIST dataset and Fashion MNIST dataset, such that at least 90% of nodes are considered important when calculating the node importance on the validation set. We will only go into details for our results on the MNIST dataset, for our other results go to section B.6.

	Unoptimized Weights	Optimized Weights
mean	0.8689	0.8691
std	0.0060	0.0061
min	0.8571	0.8562
25%	0.8643	0.8669
50%	0.8691	0.8698
75%	0.8745	0.8731
max	0.8789	0.8782

Table 4.30: Comparison of statistics of the accuracy of 25 single-layer ANNs where the weight are un-optimized using pruning versus the same layers where weight has been optimized with pruning. The ANNs are trained on the MNIST dataset.

Whether we look at Table 4.30 or Table 4.31, the result is the same. Optimizing the weights such that at least 90% are considered important before training, leads to almost the same final loss. The only difference is that it takes much longer to do it (80 seconds instead of six). Therefore, optimizing weights does not seem to be a viable solution. The results for the Fashion MNIST are very much the same with training taking longer for no gain.

	Unoptimized Weights	Optimized Weights
mean	0.0791	0.0814
std	0.0041	0.0051
min	0.0712	0.0731
25%	0.0764	0.0763
50%	0.0783	0.0819
75%	0.0813	0.0840
max	0.0866	0.0929

Table 4.31: Comparison of statistics of the loss of 25 single-layer ANNs where the weight are un-optimized using pruning versus the same layers where weight has been optimized with pruning. The ANNs are trained on the MNIST dataset.

4.9.2 Other Models

The results do not get better for the CNN model or the MLP model. Whether we optimize the weights or not we end up with either similar or slightly worse loss and accuracy. Further, the method now starts to become very long with on average 270 seconds to train an MLP on the MNIST dataset and 510 seconds to train on the Fashion MNIST. It is even worse for the CNN, where we are forced to reduce the percentage of important nodes to 60% to make it work consistently for the MNIST and Fashion MNIST (taking 600 and 760 seconds respectively to train), while we could not get the optimization to work on the CIFAR-10 model and therefore could not get results for it. For comparison, it normally takes seven seconds to train an MLP on Fashion MNIST and 12 seconds to train a CNN on CIFAR-10. The full results can be found in section C.7 and section D.7.

4.9.3 Discussion

In theory, this seemed like an interesting idea, setting all the nodes to be important might have led the model in the correct direction when training. However, this was not the case, and using this ends up being much too long and either not improving results or making it slightly worse. We could try to optimize on the training set instead of the validation set to see if that ameliorates the situation, but, when considering these results and the fact that it would probably take longer to optimize on the training set, we would probably still obtain very similar unsatisfying results. A more interesting idea would be to increase the number of zero or worse nodes at the start instead.

4.10 Summary

We started by trying to prune our models randomly. First by removing n nodes at random and seeing how that affected the model. Those results usually gave equal or slightly worse models up to four nodes removed.

Once we passed four nodes removed, the models were significantly worse. Therefore, we decided to still remove randomly but in a more controlled manner where if the model on the train set did not improve from the removal, then we did not remove them. On the other hand, if it did, then we did we remove them. This was better, with the models often being better while being able to remove eight nodes on average. However, it was not very consistent, where we could remove anywhere between zero and 16 nodes in a lot of cases depending on the dataset trained on and the random combination of nodes that happened to occur.

Since these methods did not seem to be sufficiently reliable, we looked into using node importance to prune. However, before starting to prune using node importance, we needed to investigate how different datasets and models affected the node importance and how node importance worked with multiple layers. We saw that for a single-layer ANN, more complex models tended to generate more worse and zero nodes, while that was not so evident when we had multiple layers, wherein a lot of cases the opposite tended to happen. We also saw that for dense layers, the deeper we went in the model, the less unimportant nodes there were, while for the convolutional layers it was the opposite. However, this could also be due to the sizes of layers, since they evolve in opposite directions when looking at dense and convolutional layers. Finally, we also looked at estimating node importance with the validation set versus using the training set. We saw that we always had more unimportant nodes when using the validation set compared to using the training set. This leads us to believe if we use the validation set, then we can capture the nodes that optimize the noise rather than the data itself.

Then we started looking at how two different parameters could affect node importance. The first was batch size during training. We saw that in general, as batch size increased, so did the number of important nodes. For worse nodes, it was a bit more complicated, where for dense layers, the number would decrease, while for convolutional layers it would start by increasing before reaching a maximum and then decreasing rapidly. As for zero nodes, they would usually decrease, before stabilizing at a minimum, or slightly increase. When looking at average node importance for important nodes, we saw that it would be at its lowest (which means that they were more important) when the batch size was one and then increasing to a stable point and only slightly decreasing again for a few layers. The notable exception was for the CNNs, where the models were unable to learn anything when the batch size was one. Where it would instead minimize at the same point as when the number of worse nodes is maximized. Finally, the average node importance for worse nodes is relatively stable across all batch sizes, except for batch size 256, where for a single-layer ANN trained on Fashion MNIST or a CNN trained on CIFAR-10, it is maximized. Overall, we decided to keep a batch size of 32 as our defacto batch size, since it offered the best starting performance to the number of worse/zero/important nodes.

Then we focus on seeing how adding dropout to networks only containing dense layers affected their node importance. In general, we

have the number of important nodes decrease along with its average node importance increasing, before suddenly jumping up for the number of important nodes and jumping down for its average node importance. This leads us to believe that at a certain dropout rate we have the node specializing in classifying specific classes rather than being general. This is emphasized by the opposite behavior for the worse nodes, where they increase before suddenly jumping down. The same cannot be said for their average node importance, where it constantly decreases instead of having any erratic behavior. Moreover, while this is the case for the single-layer ANN, it is only the case for the two last layers of the MLP, the first layer instead has a constant decrease of important nodes as the dropout rate increases, while the number of worse nodes still follows the same pattern as all the others. As for the number of zero nodes, it just constantly goes up as the dropout rate increases. Overall, the results obtained using dropout are not that impressive since they are either equal or worse than when there is no dropout. Therefore we decided to have no dropout going forward.

Now that we have explored how different parameters, models and datasets can affect node importance, we move on to prune the models using node importance. We first start by pruning based off of pre-calculated node importance, where all the nodes classified as zero or worse would be removed. This leads to lackluster results, with models almost always being considerably worse than before they are pruned. This led us to believe that removing all the nodes classified as worse and zero could be an error since some of these nodes might be redundant and that we would need to keep at least one of these nodes to avoid losing model performance.

Thus, we decided to prune the models in an iterative fashion and per layer, where we would find the node with the highest node importance in a layer, remove it, and then re-calculate the node importance of the other nodes. We would do this until there were only important nodes left in the layer, then we would move on to the previous layer since we did this in a backward fashion. Overall, we ended up removing fewer nodes than the number of zero and worse nodes we estimated there to be. However, this made sense since removing them all made the model worse. Furthermore, we tended to remove as many nodes as when removing randomly. However, it was generally faster and more consistent when looking at a per model average. We also, once again, compared using the training set and validation set to calculate node importance. Unsurprisingly, we removed more nodes using the validation set compared to using the training set. However, we also saw that the decrease in loss between using either set being very similar. Moreover, using the validation set was faster than using the training set. Therefore we decided to use only the validation set going forward, even though the initial model performance was lower than when having no validation set.

As mentioned before, using the validation set is faster than using the training set. However, it is still relatively slow. Therefore, we decided to test a new method to remove nodes, where instead of removing the node with the highest node importance, we remove the first worse or zero node we encounter in the layer before restarting the node importance

calculations. Further, we set an upper threshold under which nodes with a node importance value under it would be ignored in further node importance calculations (and therefore not be able to be removed any more). We will refer to this method as greedy, while the previous method is called exhaustive. The results we obtained were pretty positive, with slightly lower performing models compared to the models pruned with the exhaustive model, but still better than the unpruned models. However, the number of nodes removed was pretty similar. Finally, and most importantly, the speed-up of using this technique was very significant, with up to 11 times faster pruning with this method compared to the exhaustive method. There were a few models (especially with simple datasets on simple neural networks such as the single-layer ANN) where the greedy method performed significantly worse. However, we only tested two different thresholds, and even with only two, we saw that for some models the higher one was better, while for others, it was the opposite. Therefore, this gives us hope that if we used a different threshold, we could end up with similarly performing models using this method, so, we decided to stick to this method for future experiments.

Now that we have found a relatively stable and fast method to prune nodes, we decided to look at how pruning affected the class accuracy instead of the whole accuracy. This was very informative. While the class accuracy stayed either relatively similar or slightly higher, the standard deviation of the class accuracy, general accuracy, and loss reduces significantly after pruning. This is especially true for simpler datasets, where standard deviation for loss dropped by up to 77%. This tells us that there might be a common model that all other models could be based off and if we tried to tweak the threshold for ignoring nodes, the limits of zero nodes, or switching to exhaustive pruning instead of greedy pruning, we might be able to find it.

Our final experiment was to try optimizing the weights using the node importance before training. The idea behind it was that if we started with a higher percentage of important nodes, then this might guide the model in the right direction to learn the model. However, this was not the case, with the models trained with and without optimization had the same final performance. Not only that, optimizing the model significantly increased the time taken to train the model, and for the CNN on the CIFAR-10 data, we were unable to optimize it, even when dropping the starting percentage of important nodes.

Chapter 5

Case study: Reducing a VGG-16 model trained on the Kvasir dataset

Now that we have explored how to prune networks and how different methods, datasets, and parameters affect pruning, we will try to prune a bigger network trained on the Kvasir dataset. For our neural network architecture, we decided to use the VGG-16 architecture because it is commonly used in image classification problems and has achieved high results in the ImageNet Challenge [23]. Moreover, it is a relatively large network, therefore it might show the effects of pruning and node importance more clearly. Since we are using the original VGG-16 structure, we first resize our images to be of size $224 \times 224 \times 3$. We then split our data into three sets, a test set, a training set, and a validation set. We split the data in a 70:15:15 fashion where the largest set is the training set. We also make sure that the class bias (number of samples per class in the set) in each set is as close to zero as possible. We initialize our VGG-16 weights using the Glorot Uniform [5] initialization and then train our model. For training we use an early stopping strategy where, if the model does not improve the validation data within seven epochs, it will stop training and revert to the best weights obtained (which were the weights seven epochs ago). For example, if we train our model for 60 epochs before it stops, our weights would be the ones obtained after 53 epochs of training since there has not been any improvement after epoch 53. The performance of the model on each set after training can be found in the first column of Table 5.1.

5.1 Node importance estimation

After the first basic training and testing iteration, we start the procedures to prune the model. Before pruning our model, we first need to estimate our node importance to have an idea about how many nodes will be removed during the process and what the distribution of worse, zero, and important nodes is in each layer. For reasons explained in chapter 4, we will use the validation set in both the estimating node importance process and during

		Before Pruning	After Pruning	Change
Training set	**Loss**	0.1371	0.2818	+0.1447
	Accuracy	0.9554	0.9034	-0.052
Validation set	**Loss**	0.357	0.192	-0.165
	Accuracy	0.8817	0.9625	+0.0808
Test set	**Loss**	0.4099	0.4103	+0.0004
	Accuracy	0.8725	0.8467	-0.0258

Table 5.1: The loss and accuracy of each set before and after pruning the trained VGG-16 model. It also shows the change in those metrics after pruning.

pruning.

	Important Nodes	Zero Nodes	Worse Nodes	Zero Worse Nodes	Pruned Nodes
1^{st} **64-filters layer**	36	14	16	30	21
2^{nd} **64-filters layer**	25	17	22	39	25
1^{st} **128-filters layer**	56	30	42	72	46
2^{nd} **128-filters layer**	47	34	47	81	47
1^{st} **256-filters layer**	75	88	93	181	96
2^{nd} **256-filters layer**	85	73	98	171	94
3^{rd} **256-filters layer**	85	79	92	171	84
1^{st} **512-filters layer**	198	139	175	314	170
2^{nd} **512-filters layer**	182	137	193	330	186
3^{rd} **512-filters layer**	222	84	206	290	143
4^{th} **512-filters layer**	271	52	189	241	109
5^{th} **512-filters layer**	282	28	202	230	88
6^{th} **512-filters layer**	299	11	202	213	33
1^{st} **4096-nodes layer**	1976	287	1833	2120	1337
2^{nd} **4096-nodes layer**	2002	207	1877	2094	2334

Table 5.2: Estimated number of important, zero and worse nodes for a VGG-16 trained on the Kvasir dataset. After the separation looks at the number of unimportant nodes (zero and worse nodes) compared to the number of nodes removed by pruning of the network.

If we look at the first three columns of Table 5.2, we can see our results after estimating the node importance. At a first glance, we see that for a number of layers, less than half of its nodes/filters are considered important. The four exceptions being the first layer and the three last 512-filter layers. Another noteworthy aspect is that there are more worse nodes than zero nodes in every layer, and the ratio of worse nodes to zero nodes increases as we get deeper into the model. When we reach the dense layers it increases at the switch between convolutional and dense layers before decreasing again. With the high number of worse nodes, we expect our model to reduce the number of nodes significantly and hopefully have a lower loss after pruning.

5.2 Model pruning

To prune our model we used the greedy method (described in section 3.4). We utilized an ignore threshold of $-5e^{-5}$. The reason for using such a high ignore threshold is mainly due to the time it takes to prune this network. We started by trying a lower threshold, but after a week of computing time (over 150 hours of computation), we were still had not gotten through half of the last layer (the first layer to be pruned).

The pruning process had to be adjusted for this network. Due to the consequential amount of time to run this pruning process, we decided to save the nodes removed, the evolution of the loss, and accuracy as nodes were removed. The reason for this two-fold. Firstly, the GPU (an Nvidia RTX 2070 Super) sometimes hit a limit where the memory that stored the data corrupted and therefore became inaccessible, stopping the progress of the pruning. By saving which nodes were removed, we made sure that the progress up to that point was not lost.

The second reason is that the computer used to run this test was a home computer that needed to be shut down infrequently (due to its placement). Therefore, having our progress saved, helped us to make sure that the progress was not lost. Since we saved the nodes removed, we did not need to test them again when restarting the algorithm. However, we did not save the previously ignored nodes. Therefore the program reconsidered these nodes as potential nodes to be pruned. Which in turn led to some of them being removed even though had the program run in a single execution, they would have been kept. However, due to how the files with the nodes removed were written and read, we did not have the problem of removing nodes from a layer after the previous layer had already started removing nodes.

5.3 Pruning Results

Looking at the two last columns of Table 5.2, we can observe that for almost all layers we have the same behavior as previously shown where the number of pruned nodes (last column in the table) is under the number of unimportant nodes (before-last column).

However, this is not the case for the last layer, where there are 240 more nodes removed than the estimated number of zero and worse nodes. From our previous experiments, we know that removing all estimated unimportant nodes is not a good solution. This leads us to believe that a substantial number of estimated important nodes are being removed. Therefore, we hypothesize that some of the important nodes are only important because they fix errors/miss-classification of other nodes. Therefore once those nodes are removed, the nodes correcting them become either unnecessary and contribute nothing to the loss (becoming zero nodes), or over-correct and increase the loss (becoming worse nodes). Therefore using an ignore cutoff could be risky if it is too low since some of these correcting nodes could end up being ignored.

The proportion of pruned nodes to estimated unimportant nodes strongly decreases between the 1^{st} 4096-nodes layer and the 6^{th} 512-filter layer. After that, it increases as we go back through the layers to reach about 0.5 at the 3^{rd} 512-filter layer. It then stays relatively high, with the proportion varying between 0.5 and 0.7. Therefore the last three convolutional layers (4^{th} to 6^{th} 512-filter layers) are the only layers with significantly less than half the estimated number of unimportant nodes pruned. Seeing that these are the layers right before the dense layers and that we get the lowest proportion of pruned nodes to estimated nodes at the last convolutional layer, we suspect that the change from convolutional to dense layers might affect the number of truly worse and zero nodes in the layer. However, we would need further investigation to verify this.

The results obtained after pruning can be seen in the second column of Table 5.1. The third column of Table 5.1 represents the change in loss and accuracy after pruning the model.

As one can see in Table 5.1 the obtained results are rather surprising. Firstly, the test set loss almost did not change (only 0.0004 higher after pruning), but the accuracy drops by 2.5%, which, while not excessive, is still noticeable. Another surprising result is that we now have a validation set with a lower loss and higher accuracy than the training set. While it is normal that the loss of the training set increases through pruning based on the validation set, we did not expect it to increase this significantly. This seems to indicate that through pruning we achieved overfitting for the validation set even though we never trained our weights on it.

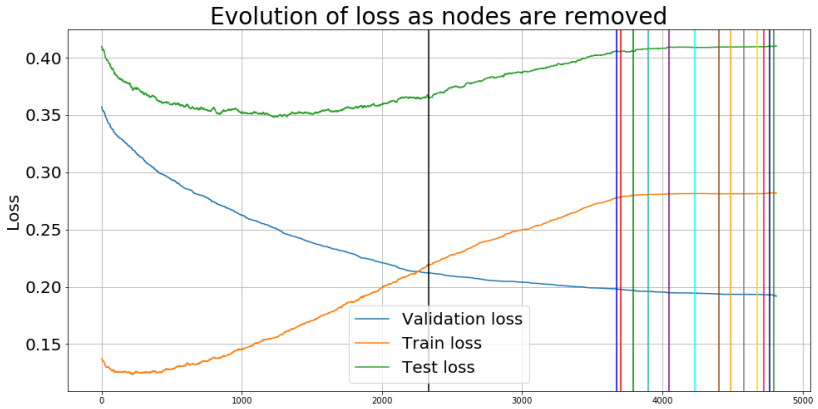

Figure 5.1: Evolution of the loss for the training, validation and test set as the model is pruned.

This is confirmed by looking at the evolution of the loss of the training and validation set in Figure 5.1, where we see the validation loss passing under the training set and then the gap in the two losses widening as we prune nodes before slightly stabilizing. This is an unexpected phenomenon, and an initial theory that we can posit is that there is noise present in a small subset of the training set which is much more common

in the validation set. Therefore, pruning on the validation set leads to focusing the classification on this noise. Which in turn increases the overall loss of the training set since it is only present in a minor part of the data while decreasing the loss of the validation set where this noise has a bigger influence on the data. However, since we do not have class bias in our data for each set (all sets have the same amount of each class samples or each class represents 12.5% of the data of each set), this is hard to prove. Focusing on the evolution of the test loss (seen in both Figure 5.1 and Figure 5.2), we again see signs of overfitting, with the loss reducing down to around 0.35 before increasing back to around 0.41.

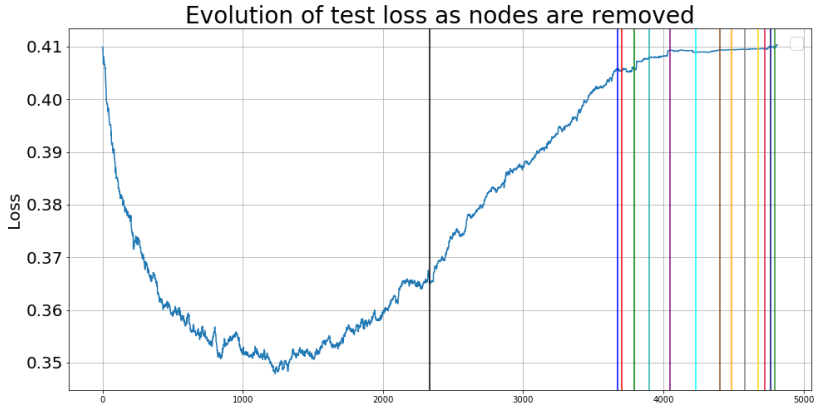

Figure 5.2: Evolution of the loss for the test set as the model is pruned.

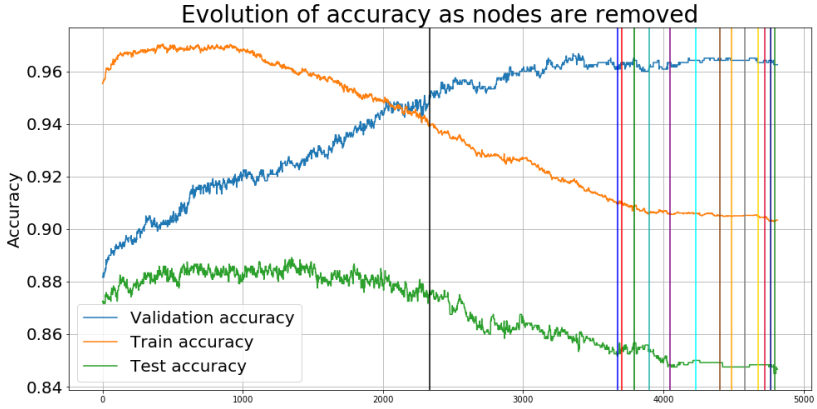

Figure 5.3: Evolution of the accuracy for the training, validation and test set as the model is pruned.

This trend is less pronounced for the accuracy of the test set. While the accuracy of the test set did decrease, from 87.3% to 84.7%, the accuracy during pruning never increase by much, peaking right under 89%. However, it only started significantly reducing around the same point as

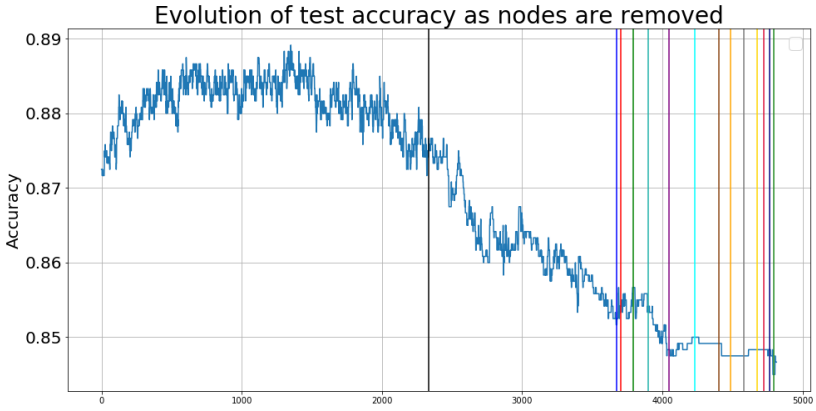

Figure 5.4: Evolution of the accuracy for the test set as the model is pruned.

when the loss started increasing. Therefore, we can say that the trend is maintained for the accuracy, where we see overfitting on the validation set. The reason, the initial increase is not as significant as the reduction in the loss, is probably due to us pruning solely based on the loss and not based on the accuracy.

A final metric we decided to look at was the change in the class accuracy before and after pruning. Table 5.3, Table 5.4 and Table 5.5 show the percentage change between the confusion matrix before pruning to the one after pruning. These values are calculated by taking the difference between the two confusion matrices and dividing each value by the number of images in each class (150 for the validation and test sets, 700 for the train set), and finally multiplying the values by 100. In the tables, the positive values represent an increase of images from class x being classified as class y, while the negative values represent a decrease. The reason we look at the differences between the confusion matrices instead of looking at the two confusion matrices is to emphasize the effects of pruning the networks.

Looking at Table 5.4, we can see that the diagonal is almost all positive values, meaning that more images are classified correctly. The only class where this is not the case is the Esophagitis class with a 0.7% decrease in correctly classified images. However, this only represents one image, and therefore we assume that overall, it did not affect the classification for the Esophagitis class. The same can be said for classes Pylorus and Ulcerative Colitis, where the increase in correctly classified images is also negligible. If we now look at the changes in more depth, we can observe that the vast majority of newly correctly classified images in the classes Dyed and Lifted Polyps and Dyed Recesection Margins come from each other. In other words, a significant number of Dyed and Lifted Polyps images previously incorrectly classified as Dyed Recesection Margins became classified as Dyed and Lifted Polyps, and vice-versa (although it is less important in the other direction). This makes sense since they are both dyed images and therefore have more in common with each other than with other

		Predicted Labels							
		dlp	**drm**	**eso**	**nce**	**npy**	**nzl**	**pol**	**uco**
Actual Labels	**dlp**	-4.7	+4	0	0	0	0	0	+0.7
	drm	+4.7	-5.3	0	+0.7	0	0	+1.3	-1.3
	eso	0	0	-12.7	0	-0.7	+13.3	0	0
	nce	0	0	0	+2.7	0	0	0	-2.7
	npy	0	0	-0.7	0	-1.3	+2	-0.7	+0.7
	nzl	0	0	-8.7	0	+1.3	+7.3	0	0
	pol	+0.7	0	0	0	-0.7	0	+6	-6
	uco	+2	-0.7	0	+5.3	+1.3	0	+4.7	-12.7

Table 5.3: Confusion matrix on the **percentage change of classification per class** on the test set after pruning. The percentage is calculated based on the number of images in each class (150 in this case). In other words, the number of images that changed classification over the number of images in each class (150).

		Predicted Labels							
		dlp	**drm**	**eso**	**nce**	**npy**	**nzl**	**pol**	**uco**
Actual Labels	**dlp**	+22	-18.7	0	0	0	0	-2	-1.3
	drm	-10	+11.3	0	+0.7	0	0	-0.7	-0.7
	eso	0	0	-0.7	0	-0.7	+1.3	0	0
	nce	0	0	0	+4.7	0	0	-2	-2.7
	npy	0	0	-1.3	0	+0.7	+0.7	0	0
	nzl	0	0	-14.7	0	-3.3	+18	0	0
	pol	0	0	0	0	-1.3	0	+8	-6.7
	uco	0	0	0	+2	0	0	-2.7	+0.7

Table 5.4: Confusion matrix on the **percentage change of classification per class** on the validation set after pruning. The percentage is calculated based on the number of images in each class (150 in this case). In other words, the number of images that changed classification over the number of images in each class (150).

classes. Otherwise, there was a significant increase in the number of images classified as Z-line, where the largest increase was for the correctly classified images, while the number of images classified as Esophagitis noticeably reduced. Unsurprisingly, the most movement was between these two classes since they both look at the z-line. The last classes did not get as significantly impacted by the network pruning.

Comparing Table 5.3 to Table 5.4, we see that there are both, some similarities and some major differences. The most surprising difference is the reduction in classification accuracy between the Dyed Recesection Margins and Dyed and Lifted Polyps classes. While we still see a switch of images between the two classes, we can observe it in the opposite direction, where previously correctly classified images become incorrectly classified. This is quite confusing since we expected such a significant increase in

		Predicted Labels							
		dlp	**drm**	**eso**	**nce**	**npy**	**nzl**	**pol**	**uco**
Actual Labels	**dlp**	-7.7	+7.9	0	+0.1	0	0	-0.1	-0.1
	drm	+6.1	-6.3	0	0	0	0	+0.1	0
	eso	0	0	-13.3	0	-0.6	+12.7	0	0
	nce	0	0	0	-2.3	0	0	+2	+0.3
	npy	0	0	+0.4	0	-1.9	+1.3	0	+0.1
	nzl	0	0	-0.9	0	-0.6	+1.4	0	0
	pol	+0.4	0	0	+2.6	0	+0.4	-2.3	-1.1
	uco	+0.3	0	0	+4.3	+0.7	+0.1	+3.3	-8.7

Table 5.5: Confusion matrix on the **percentage change of classification per class** on the training set after pruning. The percentage is calculated based on the number of images in each class (700 in this case). In other words, the number of images that changed classification over the number of images in each class (700).

classification on the validation set to be due to a better understanding of the general patterns of the classes rather than the noise. Otherwise we, still have three classes that have an increase in accuracy, the Cecum, Z-line, and Polyps classes. This is not very surprising since they were classes with significant increases in accuracy for the validation set. We again see the trend of images previously classified as Esophagitis, now being classified as Z-line. However, it now also occurs for the previously correctly classified Esophagitis images, which is disappointing. A final major difference is a heavy decrease in the number of correctly classified Ulcerative Colitis images (12.7% equivalent to 19 images). A majority of those were reclassified as either Cecum or Polyps. In the validation set, it seemed that there was no real change in how the Ulcerative Colitis images were classified.

Table 5.5 repeats much of the same trends as the test set, except that they are more pronounced. Here only the Z-line class becomes more accurate. However, it is a much smaller increase (1.4% versus 18% and 7.3%). Except for that difference we have the same trends as for the test set.

5.4 Summary

To summarize, through pruning a more complex network trained on a real-world dataset, we learned a new behavior that nodes have in neural networks and an important effect pruning has on the model. Through the fact the number of pruned nodes on the last layer was larger than the number of estimated unimportant nodes in that layer, we saw that some nodes initially important end up becoming unimportant. This leads us to believe that some nodes in the network are there to correct errors in the model. This goes in line with the idea of co-adaption which dropout tries to remove. The other main finding from this case study was the possibility of

overfitting on the validation set when pruning based on it. We previously hypothesized that pruning on the validation set rather than the training set would lead to a reduction of noise fitting by the model. This was disproved for this case study, where we saw the model becoming overfitted on the validation set. Our current hypothesis is that the model is overfitting on the noise shared by both the validation and training set. However, this can be very data set specific (the Kvasir dataset has some green screen in screen on some frames, which could lead to this behavior) and we need to do more research to either reject our hypothesis or refine it.

Chapter 6

Conclusion

While we have extensive knowledge of neural network layers and increasing their depth, specific nodes in those layers still hold many mysteries to their contributions to a neural network. Throughout this thesis, we endeavoured to expand our knowledge of their inner workings, but we have only scratch the surface. We developed a metric to understand their impact on the neural network (which is similar to the metric developed by Nvidia [18]). However, our metric does not tell us how a node affects or works with another node of the same layer or a different layer. We also look at how removing an individual node impacts the loss of the model. This being a key metric for understanding model performance tells us a lot about how a node impacts a neural network. However, to gain an even deeper understanding, looking at a variety of model metrics might help. Finally, we saw that changing hyper-parameters such as batch size during training can have a significant impact on the number of nodes in each class of node importance (important, zero, and worse).

During our thesis, we improved our pruning techniques and derived hypotheses based on our results as to how pruning was affecting neural networks. We started by randomly removing nodes, this led to a mediocre result that was not reproducible since the nodes removed were chosen at random. Therefore, we started using our node importance metric to remove nodes. Our initial idea was to pre-calculate the node importance of all nodes and remove all the nodes that either negatively impacted the model performance, or had no effect on it (worse and zero nodes). This led to models that constantly and noticeably performed worst after pruning which led us to hypothesis that some nodes are redundant. Therefore, the number of unimportant nodes is inflated due to some unimportant nodes being important when considered in isolation to their redundant nodes in the network. For example, if two nodes in a model both extract the same feature, then removing either of them does not change the performance of the model, however, by removing both, we lose the ability to extract that feature. Based on our results and hypothesis we changed our strategy to removing nodes one by one and re-calculating node importance after each iteration. This supported our hypothesis to always removing fewer nodes than the number of pre-calculated unimportant nodes.

We then decided to try pruning based on the validation set rather than the training set. The reason for this was two-fold, first using the smaller validation set would decrease the time to prune networks, second, we hypothesized that using the training set to prune the networks will not catch the nodes that are trained on the noise of the training set. We saw that pruning based on the validation set led to a very similar result in terms of an increase in performance while increasing the number of nodes pruned. Therefore, we decided to stick with pruning on the validation set as a more optimal form of pruning.

While pruning on the validation set reduced the time to prune our networks, the pruning was still relatively slow. Therefore, we devised a new pruning strategy. This new strategy ended up being a ten-fold acceleration compared to the previous. While the model performance increase (i.e., the decrease in loss) was slightly reduced, the number of nodes pruned was slightly increased. Therefore, we decided to continue using this pruning strategy going forward.

Now that we have improved our pruning techniques, we decided to look at how pruning affected the class accuracy, general loss, and general accuracy. We saw that overall the class accuracy went up. For those classes where it did not, it either stayed the same or reduced slightly. However, a more interesting result was that the variation between the class accuracies of different models with the same initial architecture was at least halved if not more. This was also observable for the general loss and accuracy of the models, where the reduction is usually smaller, reaching from a 10% reduction up to a 70% reduction.

Finally, we considered all our previous results and applied them to a more complex network (VGG-16) on a new dataset (Kvasir). While we expected to get similar results, we ended up getting surprising results. Firstly, after pruning we end up with a very similar performing model, instead of a better model. Upon further investigation, we saw that the pruning overfitted the validation set and that the test loss started by reducing before increasing again. Secondly, for the last layer, we pruned more nodes than the estimated number of unimportant nodes. This might indicate that some initially important nodes were correcting unimportant nodes, which once removed, made those important nodes unimportant. Therefore, these new results gave us a new dimension to how nodes affect neural networks. Whereas previously we only saw nodes either being redundant or unhelpful, we now have nodes that are only helpful for correcting unhelpful ones. Therefore once the node a node corrects for is removed, the previously helpful node starts over-correcting for an error that it no longer there, thus introducing a new error.

6.1 Main Contributions

In this thesis, we explored how nodes and their removal affect neural networks. To guide our exploration, we split our research questions into three different objectives. These are as following:

Objective 1: *Exploring the effects of pruning neural networks and developing different pruning techniques for both reproducibility and time-consumption.*

Through the thesis, we explore different facets of this objective. We go from a pruning technique that is hardly reproducible due to its inherent randomness to one that works based on unchanging node parameters and metrics but is rather time-consuming. Before finally settling for one that keeps the reproducibility of the previous but reduces its time-consumption by at least a factor of two. We also look at various metrics affected by the pruning of a network, and we can conclude that pruning a network has a noticeable impact on neural networks by generally reducing the loss, or in some cases keeping it constant or slightly increasing it. This might also be interesting for generalization of the networks which needs to be further studied.

Objective 2: *Classifying the nodes based on how they affect the neural network if removed.*

The thesis supports this objective since we develop a metric, called node importance, to classify these nodes based on their effect on the loss of the model. The node importance is calculated by removing a node from the neural network and then calculating the change in the loss of the model generated by this removal. Then if the loss of the model increases, the node is classified as an important node. If the loss does not change or the change is insignificant then the node is considered to be a zero node. Finally, if the removal of the node decreases the loss, then it falls in the worse node category. We also use node importance to help us prune networks. To not prune nodes randomly, we iteratively remove worse and zero nodes from the network, till all the nodes left are important. We do this both in an exhaustive fashion (finding the node with the highest node importance value and removing it) and greedily (removing the first zero or worse node we come across).

Objective 3: *Seeing how different parameters and types of neural networks affect the nodes impact on neural networks.*

During the thesis, we use three different types of neural networks to explore their reaction to pruning. We also modify one hyper-parameter (batch size during training) and see how the inclusion of dropout to our purely fully connected networks affects performance.

We were therefore able to achieve our three objectives and create hypotheses on how pruning affects neural networks. By focusing our node importance metric solely on the loss of the model, arguably the most important metric for model performance, we see that individual nodes impact the model in both positive (important nodes) and negative ways (worse nodes). This led us to hypothesize that some nodes are redundant or even unhelpful for the model overall. To be able to strengthen this hypothesis further, we should continue exploring these effects by broadening our metric to how nodes impact other model metrics such as accuracy and precision, etc. Moreover, looking at the individual effects of

nodes is a good starting point in giving us intuition on the impact of nodes on the model. However, looking at how one node impacts another node, would help us to better understand how nodes interact with each other and ultimately how these interactions impact the neural network as a whole.

For the pruning, we have shown throughout the thesis that it frequently has a positive effect. In most cases, the model loss went down and the accuracy either stayed similar or increased slightly. Even though the model loss did not change much for the larger network, we were able to significantly reduce the size of our network (over 35% reduction in the number of filters and nodes). This in itself is a positive effect since using a much smaller network will require less memory and computation. This also led us to a new hypothesis where a network, through pruning, could be overfitted to a different dataset than the one it was trained on which would lead to a kind of transfer learning effect. Further researching this new path could lead us to a better understanding of neural networks and transfer learning. Moreover, by logging the evolution of the model loss through the pruning, we saw that the loss of our model did start by decreasing before increasing due to overfitting. This helps ascertain our views that not all nodes in a network directly benefit the network.

Our thesis does start to answer our research question on how nodes contribute to the overall performance of neural networks. As mentioned in the two previous paragraphs, we have seen and hypothesize some of the effects of nodes on neural networks. We have also shown that removing nodes frequently improve our models. These experiments have opened new paths to better understand how one of the base elements of neural networks, the nodes, work. Continuing the exploration of these previously unknown new paths could lead us to better answer our research question and truly understand the impact of nodes on neural networks.

The code for this thesis can be found at https://github.com/lgcharpe/Masters (MIT License).

6.2 Future Works

As mentioned before, in this thesis we have only scratched the surface of how nodes affect neural networks. We have created ways to prune the neural networks that do, in general, improve the performance while reducing the size of those networks. We were able to classify the nodes into three different categories, depending on their contribution to the loss of the model. Finally, we have started looking at how some parameters affect the node importance of the model's nodes. From these results, we have seen various new avenues to explore to have a better grasp of how nodes affect neural networks. Some of them are discussed in the following.

- **Explore how different hyper-parameters affect node importance**

  While we have only looked at how batch size during training affects node importance in this thesis, we saw that it did have a significant

effect on the node importance. Therefore, we should further investigate how other hyper-parameters affect node importance. We believe that looking at how changing the number of nodes per layer or changing the learning rate can affect node importance could help us better understand the role of nodes in neural networks.

- **Develop a more refined node importance metric that considers various performance metrics**

  As mentioned previously, we measure node importance solely based on the loss of the model. While this a key metric to judge model performance since it looks at how close to perfectly classifying each image to its class, it is not the only one. Therefore, redesigning our node importance metric by calculating node importance based on a linear combination of model metrics such accuracy, AUC, F1-score, etc. in addition to the loss could lead to a better understanding of nodes and a more efficient and reliable way to prune models.

- **Develop a metric that measures node interactions**

  This builds up on the above discussed idea. Our current metric and the redesigned one proposed by the point above only consider the individual effect of the nodes on the model performance. It does not tell us much about how nodes interact with each other. Therefore, trying to develop a metric that measures node interactions and how removing one, affects the others would help us strengthen the hypothesis observed during this thesis, such as redundant nodes and error-correcting nodes. Developing this metric will deepen our understanding of nodes.

- **Improve the pruning algorithm to make it less time-consuming**

  While our greedy method does improve the pruning time, it is still very long on larger models when compared to the training time. A possible solution to remedy this problem is using different metrics that do a coarser estimation of the node importance (as seen in the paper by Nvidia [18]). However, a more interesting solution would be to develop a pruning technique that continues to use node importance (or a redesigned version of it) to prune networks. To do this, we would look at ways to accelerate our current technique using parallel programming would be interesting. This would need to be validated as parallelizing pruning might lead to some inaccuracies in terms of node importance.

- **Delve further into the reduction in variance between models after pruning**

  A very interesting result of our exploration was the reduction in variation caused by pruning models. We believe that further exploring this reduction might lead to finding a base model performance that is shared by all similar models. In other words, we believe that it might

be possible to reduce the variance to zero. If this is the case, then we would try to develop a mathematical explanation for this phenomenon. This would greatly help not only our understanding of neural networks but also give us a better idea of performance before even training a model.

- **Explore and reduce the validation overfitting caused by pruning**

  As seen in our case study, pruning on the validation set caused the model to overfit on the validation set. This was unexpected since, while being a form of training, we do not change any of the weights of non-pruned nodes. Therefore delving deeper into how pruning leads to overfitting is needed. Our current hypothesis is that the noise present in both the train and validation set is amplified when pruning on the validation set. However, we need to look further into it to confirm this theory. Once we get a better grasp of how the overfitting occurs, we will try to develop a pruning technique that reduces it. By doing this, we will develop a more reliable pruning technique.

Appendix A

Algorithms

Algorithm 4: Algorithm to estimate the node importance of each node and then classifying them

```
1  def estimateNodeImportance(model, tester_model, layer_sizes,
     tol_low, tol_high, x, y):
       Input:
       model is the TensorFlow model of the trained neural network
       used;
       tester_model is a untrained copy of model, this will be used as a
       tester model to test different network weights;
       layer_sizes is the number of nodes/filters in each layer of the
       network (ordered backwards from the layer closest to the
       output to the one closest to the input);
       tol_low is the tolerance for which node importance values below
       it, categorize the nodes as important;
       tol_high is the tolerance for which node importance values
       above it, categorize the nodes as worse;
       x is the dataset to consider when estimating the node
       importance;
       y is the labels of x;
       Output: The number of nodes in each category for each layer
                and their position in the layer
       /* Start of the code                                        */
2      loss ← the loss of model when evaluated on x and y
3      original_weights ← weights of model
4      weight_len ← len(original_weights) − 3 ;  /* We remove 3 for
        the bias and weight value to the output layer plus 1
        because python lists start at 0 */
5      amounts_removed ← empty list
6      places ← empty list
7      for layer, size in enumerate(layer_sizes):
8          num_zeros, num_worse, num_important ← (0, 0, 0)
9          zeros_place ← empty list
10         worse_place ← empty list
11         important_place ← empty list
12         for i in range(size):
13             w ← copy(original_weights)
               /* Setting the values of the bias and weights
                  incoming to the node to zero, has the same
                  effect as removing the node.                  */
14             w[weight_len − (2 ∗ layer + 1)][..., i] ← 0 ; /* We set the
                weights incoming to node i in layer to 0 */
15             w[weight_len − (2 ∗ layer)][i] ← 0 ;  /* We set the bias
                incoming to node i in layer to 0 */
16             weights of tester_model ← w
17             new_loss ← the loss of tester_model when evaluated on x
                and y
18             change ← loss − new_loss
```

```
16
17
18
19                  if change ≤ tol_high and change ≥ tol_low:
20                      num_zeros ← num_zeros + 1
21                      zeros_place append i
22                  elif change > tol_high:
23                      num_worse ← num_worse + 1
24                      worse_place append i
25                  else:
26                      num_important ← num_important + 1
27                      important_place append i
28              amounts append (num_zeros, num_worse, num_important)
29              places append (zeros_place, worse_place, important_place)
30      return amount, places
```

Algorithm 5: Algorithm to estimate the node importance of each node and then classifying them. This traverses the network in a forward manner, from the hidden layer after the input layer to the hidden layer before the output layer.

```
1  def estimateNodeImportanceForward(model, tester_model,
     layer_sizes, tol_low, tol_high, x, y, back, forward):
       Input:
       model is the TensorFlow model of the trained neural network
       used;
       tester_model is a untrained copy of model, this will be used as a
       tester model to test different network weights;
       layer_sizes is the number of nodes/filters in each layer of the
       network;
       tol_low is the tolerance for which node importance values below
       it, categorize the nodes as important;
       tol_high is the tolerance for which node importance values
       above it, categorize the nodes as worse;
       x is the dataset to consider when estimating the node
       importance;
       y is the labels of x;
       back is a boolean that indicates if we remove the incoming
       connections;
       forward is a boolean that indicates if we remove the outgoing
       connections;
       Output: The number of nodes in each category for each layer
                and their position in the layer
       /* Start of the code                                          */
2      loss ← the loss of model when evaluated on x and y
3      original_weights ← weights of model
4      amounts_removed ← empty list
5      places ← empty list
6      for layer, size in enumerate(layer_sizes):
7          num_zeros, num_worse, num_important ← (0, 0, 0)
8          zeros_place ← empty list
9          worse_place ← empty list
10         important_place ← empty list
11         for i in range(size):
12             w ← copy(original_weights)
13             if back:
14                 w[2 * layer][..., i] ← 0 ;       /* We set the weights
                    incoming to node i in layer to 0 */
15             if forward:
16                 w[2 * layer + 2][..., i, :] ← 0 ; /* We set the weights
                    outgoing the node i in layer to 0 */
17             w[2 * layer + 1][i] ← 0 ;  /* We set the bias incoming
                to node i in layer to 0 */
18             weights of tester_model ← w
19             new_loss ← the loss of tester_model when evaluated on x
                and y
```

```
1
17
18
19          change ← loss − new_loss
20          if change ≤ tol_high and change ≥ tol_low:
21              num_zeros ← num_zeros + 1
22              zeros_place append i
23          elif change > tol_high:
24              num_worse ← num_worse + 1
25              worse_place append i
26          else:
27              num_important ← num_important + 1
28              important_place append i
29      amounts append (num_zeros, num_worse, num_important)
30      places append (zeros_place, worse_place, important_place)
31  return amount, places
```

Algorithm 6: Algorithm to estimate the node importance of each node in a single layer and then classifying them

```
def estimateNodeImportanceSingleLayer(model, tester_model,
  layer, layer_size, tol_low, tol_high, x, y, back, forward):
    Input:
    model is the TensorFlow model of the trained neural network
    used;
    tester_model is a untrained copy of model, this will be used as a
    tester model to test different network weights;
    layer is the layer to consider;
    layer_size is the number of nodes/filters in the layer considered;
    tol_low is the tolerance for which node importance values below
    it, categorize the nodes as important;
    tol_high is the tolerance for which node importance values
    above it, categorize the nodes as worse;
    x is the dataset to consider when estimating the node
    importance;
    y is the labels of x;
    back is a boolean that indicates if we remove the incoming
    connections;
    forward is a boolean that indicates if we remove the outgoing
    connections;
    Output: The number of nodes in each category and their
              position in the layer
    /* Start of the code                                          */
    loss ← the loss of model when evaluated on x and y
    original_weights ← weights of model
    num_zeros, num_worse, num_important ← (0, 0, 0)
    zeros_place ← empty list
    worse_place ← empty list
    important_place ← empty list
    for i in range(size):
        w ← copy(original_weights)
        if back:
            w[2 * layer][..., i] ← 0 ;          /* We set the weights
              incoming to node i in layer to 0 */
        if forward:
            w[2 * layer + 2][..., i, :] ← 0 ;     /* We set the weights
              outgoing the node i in layer to 0 */
        w[2 * layer + 1][i] ← 0 ;  /* We set the bias incoming to
         node i in layer to 0 */
        weights of tester_model ← w
        new_loss ← the loss of tester_model when evaluated on x
         and y
        change ← loss − new_loss
```

```
15
16
17
18          if change ≤ tol_high and change ≥ tol_low:
19              num_zeros ← num_zeros + 1
20              zeros_place append i
21          elif change > tol_high:
22              num_worse ← num_worse + 1
23              worse_place append i
24          else:
25              num_important ← num_important + 1
26              important_place append i
27      amounts ← [num_zeros, num_worse, num_important]
28      places ← [zeros_place, worse_place, important_place]
29  return amount, places
```

Algorithm 7: This algorithm re-randomizes the weights till a layer have a higher fraction of important nodes than asked. It does this for each layer going forward in the network.

```
1  def optimizeWeights(model, tester_model, layer_sizes, filter_sizes,
     tol_low, tol_high, x, y, input_size, output_size, min_imp_frac,
     input_nodes):
       Input:
       model is the TensorFlow model of the trained neural network
       used;
       tester_model is a untrained copy of model, this will be used as a
       tester model to test different network weights;
       layer is the layer to consider;
       layer_sizes is the number of nodes/filters in each layer of the
       considered network;
       filter_sizes is the dimension of the filters in each convolutional
       layer of the considered network (for a Dense layer, we have the
       value None. We assume that all filters are square);
       tol_low is the tolerance for which node importance values below
       it, categorize the nodes as important;
       tol_high is the tolerance for which node importance values
       above it, categorize the nodes as worse;
       x is the dataset to consider when estimating the node
       importance;
       y is the labels of x;
       input_size is a number of inputs/filters for the input layers;
       output_size is the number of outputs in the output layer (this is
       usually the number of classes in the dataset);
       min_imp_frac is the fraction of the nodes in a layer that need to
       be important;
       input_nodes if the input is an image, then this represents the
       number of inputs for the image. Otherwise this value is not
       used;
       Output: Updated weights
       /* Start of the code                                          */
2      for layer, size in enumerate(layer_sizes):
3          filter ← filter_sizes[layer]
4          if layer < len(layer_sizes) − 1:
5              filter_out ← filter_sizes[layer + 1] ;     /* The size of
                the filter in the next layer */
6              next_size ← layer_sizes[layer + 1]
7          else:
8              filter_out ← None
9              next_size ← output_size
10         if layer == 0:
11             tmp_amount, tmp_place ←
                estimateNodeImportanceSingleLayer(model,
                tester_model, layer, size, tol_low, tol_high, x, y, True, True)
```

```
10
11
12      else:
13          tmp_amount, tmp_place ←
             estimateNodeImportanceSingleLayer(model,
             tester_model, layer, size, tol_low, tol_high, x, y, False, True)
14      imp_frac ← tmp_amount[2] / size
15      while imp_frac ≤ min_imp_frac:
16          w ← weights of model if filter:
17              if filter_out:
18                  limit_out ← sqrt(6 / ((size*filter^2)+(next_size*filter_out^2))) ;
                     /* Using Glorot uniform to initialize new
                     weights.  */
19              else:
20                  limit_out ← sqrt(6 / (size+next_size))
21          if layer == 0:
22              if input_nodes:
23                  limit_in ← sqrt(6 / (input_nodes+size))
24              else:
25                  limit_in ← sqrt(6 / (input_size+size))
26          if tmp_amount[1]:
                /* Checking if there are any worse nodes, and
                   replacing them                              */
27              if filter_out:
28                  size_out ←
                     (filter_out, filter_out, tmp_amount[1], next_size)
29              else:
30                  size_out ← (tmp_amount[1], next_size)
31              w[2 * layer + 2][..., tmp_place[1], :] ← new random
                 weights chosen from a uniform distribution in
                 (−limit_out, limit_out)
32              if layer == 0:
33                  size_out ←
                     (filter, filter, input_size, tmp_amount[1])
34              else:
35                  size_out ← (input_size, tmp_amount[1])
36              w[2 * layer][..., tmp_place[1] ← new random weights
                 chosen from a uniform distribution in
                 (−limit_in, limit_in)
```

```
35
36
37
38      if tmp_amount[0]: ;     /* Checking if there are any
         zero nodes, and replacing them */
39
40          if filter_out:
41              size_out ←
                 (filter_out, filter_out, tmp_amount[0], next_size)
42          else:
43              size_out ← (tmp_amount[0], next_size)
44          w[2 * layer + 2][..., tmp_place[0], :] ← new random
             weights chosen from a uniform distribution in
             (−limit_out, limit_out)
45          if layer == 0:
46              size_out ←
                 (filter, filter, input_size, tmp_amount[0])
47          else:
48              size_out ← (input_size, tmp_amount[0])
49          w[2 * layer][..., tmp_place[0] ← new random weights
             chosen from a uniform distribution in
             (−limit_in, limit_in)
50      model weights ← w if layer == 0:
51          tmp_amount, tmp_place ←
             estimateNodeImportanceSingleLayer(model,
             tester_model, layer, size, tol_low, tol_high, x, y, True,
             True)
52      else:
53          tmp_amount, tmp_place ←
             estimateNodeImportanceSingleLayer(model,
             tester_model, layer, size, tol_low, tol_high, x, y, False,
             True)
54      imp_frac ← tmp_amount[2] / size
```

Algorithm 8: This algorithm starts by looking for the worst node in a layer and prunes it. It continues doing so till all nodes left in the layer are above a given threshold. It then goes to the next layer till all layers are pruned.

```
1  def pruneNodesExhaustively(model, tester_model, x, y, layer_sizes,
     tol):
       Input:
       model is the TensorFlow model of the trained neural network
       used;
       tester_model is a untrained copy of model, this will be used as a
       tester model to test different network weights;
       layer_sizes is the number of nodes/filters in each layer of the
       considered network (organized in a backward direction);
       x is the dataset to consider when estimating the node
       importance;
       y is the labels of x;
       tol is the tolerance for which nodes with a node importance
       value above it will be removed;
       Output:
       weights: Pruned weights (where incoming weights are set to 0
       for the pruned nodes);
       ev_acc: The evolution of the model accuracy as nodes are
       removed;
       ev_loss: The evolution of the model loss as nodes are removed;
       amounts: The number of nodes/filters removed in each layer;
       places: The places of the nodes/filters removed in each layer;
       /* Start of the code                                          */
2      old_loss ← the loss of model when evaluated on x and y
3      old_acc ← the accuracy of model when evaluated on x and y
4      ev_loss ← a list containaing old_loss
5      ev_acc ← a list containaing old_acc
6      weights ← copy of orignial_weights
7      weight_len ← len(weights) − 3 ; /* We remove 3 for the bias
        and weight value to the output layer plus 1 because
        python lists start at 0 */
8      best_acc ← 0
9      best_loss ← a large number
10     amounts ← empty list
11     places ← empty list
12     for layer, size in enumerate(layer):
13         end_not_reached ← True ;      /* Whether or not we have
            reached the end of the layer */
14         current_pos ← 0 ;    /* The current position of in the
            layer */
15         num_removed ← 0
16         best_pos ← −1 ;   /* The position of the current best
            node (highest node importance value) to remove on
            this pass */
```

```
15
16
17      best_change ← tol ;   /* Node importance of the current
          best node */
18      node_removed ← empty list
19      improved ← False ; /* Whether or not we have removed a
          node in this pass */
20      while end_not_reached or improved:
21          if not (end_not_reached:
22              end_not_reached ← True
23              improved ← False
24              current_pos ← 0
25              size ← size − 1
26              nodes_removed append best_pos
27              weights[weight_len − (2 ∗ layer + 1)][..., best_pos] ← 0
28              weights[weight_len − 2 ∗ layer][best_pos] ← 0
29              best_pos ← −1
30              old_loss ← best_loss
31              old_acc ← best_acc
32              ev_loss append best_loss
33              ev_acc append best_acc
34              best_change ← tol
35              num_removed ← num_removed + 1
36          if current_pos in nodes_removed:
                /* if the nodes has already been removed,
                   skip it                                    */
37              current_pos ← current_pos + 1
38              if current_pos − num_removed ≥ size:
                    /* Check whether we have reached the end
                       of the layer                           */
39                  end_not_reached ← False
40              continue
41          w ← copy(weights)
42          w[weight_len − (2 ∗ layer + 1)][..., best_pos] ← 0
43          w[weight_len − 2 ∗ layer][best_pos] ← 0
44          weights of tester_model ← w
45          new_loss ← the loss of tester_model when evaluated on x
              and y
46          new_acc ← the accuracy of tester_model when evaluated
              on x and y
47          if old_loss − new_loss ≥ best_change:
48              best_change ← old_loss − new_loss
49              best_pos ← current_pos
50              improved ← True
51              best_loss ← new_loss
52              best_acc ← new_acc
53          current_pos ← current_pos + 1
```

```
51
52
53
54          if current_pos − num_removed ≥ size:
55              end_not_reached ← False
56      amounts ← num_removed
57      places ← nodes_removed
58  return weights, ev_acc, ev_loss, amounts, places
```

Algorithm 9: This algorithm prunes the first node under a given node importance threshold it encounters. It continues doing so till all nodes left in the layer are above a given threshold. It then goes to the next layer till all layers are pruned.

```
1  def pruneNodesGreedy(model, tester_model, x, y, layer_sizes, tol,
     ignore_cutoff):
       Input:
       model is the TensorFlow model of the trained neural network
       used;
       tester_model is a untrained copy of model, this will be used as a
       tester model to test different network weights;
       layer_sizes is the number of nodes/filters in each layer of the
       considered network (organized in a backward direction);
       x is the dataset to consider when estimating the node
       importance;
       y is the labels of x;
       tol is the tolerance for which nodes with a node importance
       value above it will be removed;
       ignore_cutoff is the node importance for which nodes with a
       value under it will be ignored in future passes;
       Output:
       weights: Pruned weights (where incoming weights are set to 0
       for the pruned nodes);
       ev_acc: The evolution of the model accuracy as nodes are
       removed;
       ev_loss: The evolution of the model loss as nodes are removed;
       amounts: The number of nodes/filters removed in each layer;
       places: The places of the nodes/filters removed in each layer;
       /* Start of the code                                          */
2      old_loss ← the loss of model when evaluated on x and y
3      old_acc ← the accuracy of model when evaluated on x and y
4      ev_loss ← a list containaing old_loss
5      ev_acc ← a list containaing old_acc
6      weights ← copy of orignial_weights
7      weight_len ← len(weights) − 3 ; /* We remove 3 for the bias
        and weight value to the output layer plus 1 because
        python lists start at 0 */
8      best_acc ← 0
9      best_loss ← a large number
10     amounts ← empty list
11     places ← empty list
12     for layer, size in enumerate(layer):
13         end_not_reached ← True
14         node_removed ← empty list
15         num_removed ← 0
```

```
14
15
16      nodes_to_estimate ← list of integers from 0 to the number of
         nodes - 1 ;  /* This will be a list of the nodes that
         have not been removed or ignored */
17      current_pos ← nodes_to_estimate[0]
18      idx ← 0 ;    /* Position in the array for the node to
         consider */
19      while end_not_reached:
20          w ← copy(weights)
21          w[weight_len − (2 ∗ layer + 1)][..., best_pos] ← 0
22          w[weight_len − 2 ∗ layer][best_pos] ← 0
23          weights of tester_model ← w
24          new_loss ← the loss of tester_model when evaluated on x
             and y
25          new_acc ← the accuracy of tester_model when evaluated
             on x and y
26          if old_loss − new_loss ≥ tol:
27              best_loss ← new_loss
28              best_acc ← new_acc
29              size ← size − 1
30              nodes_removed append current_pos
31              nodes_to_estimate remove current_pos
32              weights[weight_len − (2 ∗ layer + 1)][..., best_pos] ← 0
33              weights[weight_len − 2 ∗ layer][best_pos] ← 0
34              ev_loss append new_loss
35              ev_acc append old_acc
36              num_removed ← num_removed + 1
37              idx ← 0
38          elif old_loss − new_loss ≤ ignore_cutoff:
39              size ← size − 1
40              nodes_to_estimate remove current_pos
41          else:
42              idx ← idx + 1
43          if idx ≥ size:
44              end_not_reached ← False
45          else:
46              current_pos ← nodes_to_estimate[idx]
47      amounts ← num_removed
48      places ← nodes_removed
49  return weights, ev_acc, ev_loss, amounts, places
```

Appendix B

Single-layer ANN - Extra Figures and Tables

B.1 Estimating Node Importance

B.1.1 MNIST

Tables - Training set

	Zero Nodes	Worse Nodes	Important Nodes
mean	2.26	13.86	111.88
std	1.43	6.00	5.90
min	0.00	4.00	100.00
25%	1.00	9.00	107.25
50%	2.50	13.00	112.00
75%	3.00	18.00	116.75
max	5.00	27.00	123.00

Table B.1: Number of nodes in each class of nodes

Tables - Validation set

	Zero Nodes	Worse Nodes	Important Nodes
mean	2.12	24.64	101.24
std	1.60	6.47	6.20
min	0.00	11.00	86.00
25%	1.00	19.00	96.25
50%	2.00	24.00	102.00
75%	3.00	29.75	105.00
max	5.00	39.00	112.00

Table B.2: Number of nodes in each class of nodes

B.1.2 Fashion MNIST

Tables - Training set

	Zero Nodes	Worse Nodes	Important Nodes
mean	13.96	20.92	93.12
std	3.36	6.50	7.02
min	5.00	8.00	80.00
25%	12.00	16.00	89.00
50%	14.00	19.50	92.00
75%	16.00	25.75	97.75
max	24.00	36.00	115.00

Table B.3: Number of nodes in each class of nodes

Tables - Validation set

	Zero Nodes	Worse Nodes	Important Nodes
mean	15.32	27.54	85.14
std	3.04	6.79	6.46
min	10.00	14.00	71.00
25%	13.00	22.25	80.50
50%	15.00	26.50	86.00
75%	18.00	32.75	90.75
max	21.00	41.00	97.00

Table B.4: Number of nodes in each class of nodes

B.2 Effects of batch size on Node Importance

B.2.1 MNIST

Tables

	1			8			32			256			1024		
	Zero Nodes	Worse Nodes	Important Nodes	Zero Nodes	Worse Nodes	Important Nodes	Zero Nodes	Worse Nodes	Important Nodes	Zero Nodes	Worse Nodes	Important Nodes	Zero Nodes	Worse Nodes	Important Nodes
mean	10.16	29.56	88.28	3.96	26.44	97.6	1.96	26.04	100.00	1.76	13.60	112.64	3.16	3.96	120.88
std	3.24	4.44	4.48	2.24	4.97	5.9	1.43	4.89	5.39	1.76	7.05	7.52	1.75	3.66	4.08
min	3.00	22.00	78.00	0.00	19.00	87.0	0.00	17.00	88.00	0.00	3.00	96.00	1.00	0.00	112.00
25%	8.00	27.00	86.00	2.00	22.00	93.0	1.00	23.00	96.00	1.00	10.00	107.00	2.00	1.00	118.00
50%	10.00	30.00	88.00	4.00	27.00	98.0	2.00	25.00	101.00	1.00	12.00	114.00	3.00	3.00	122.00
75%	12.00	32.00	91.00	6.00	29.00	103.0	2.00	30.00	104.00	3.00	17.00	117.00	5.00	5.00	124.00
max	16.00	42.00	97.00	8.00	36.00	106.0	7.00	38.00	110.00	6.00	29.00	124.00	6.00	14.00	127.00

Table B.5: Number of nodes in each class of nodes

	1			8			32			256			1024		
	Zero Nodes	Worse Nodes	Important Nodes	Zero Nodes	Worse Nodes	Important Nodes	Zero Nodes	Worse Nodes	Important Nodes	Zero Nodes	Worse Nodes	Important Nodes	Zero Nodes	Worse Nodes	Important Nodes
mean	-0.0	0.0018	-0.0043	-0.0	0.0008	-0.0023	-0.0	0.0007	-0.0025	-0.0	0.0006	-0.0027	-0.0	0.0003	-0.0028
std	0.0	0.0006	0.0008	0.0	0.0003	0.0003	0.0	0.0002	0.0003	0.0	0.0003	0.0003	0.0	0.0003	0.0002
min	-0.0	0.0009	-0.0072	-0.0	0.0004	-0.0030	-0.0	0.0003	-0.0036	-0.0	0.0002	-0.0034	-0.0	0.0000	-0.0031
25%	-0.0	0.0013	-0.0046	-0.0	0.0006	-0.0023	-0.0	0.0007	-0.0027	-0.0	0.0004	-0.0028	-0.0	0.0001	-0.0029
50%	0.0	0.0018	-0.0040	0.0	0.0007	-0.0021	0.0	0.0008	-0.0024	0.0	0.0005	-0.0026	-0.0	0.0002	-0.0027
75%	0.0	0.0020	-0.0038	0.0	0.0009	-0.0021	0.0	0.0008	-0.0023	0.0	0.0008	-0.0024	0.0	0.0004	-0.0026
max	0.0	0.0034	-0.0033	0.0	0.0017	-0.0019	0.0	0.0016	-0.0020	0.0	0.0013	-0.0023	0.0	0.0008	-0.0025

Table B.6: Average importance for each node class

B.2.2 Fashion MNIST

Tables

	1			8			32			256			1024		
	Zero Nodes	Worse Nodes	Important Nodes	Zero Nodes	Worse Nodes	Important Nodes	Zero Nodes	Worse Nodes	Important Nodes	Zero Nodes	Worse Nodes	Important Nodes	Zero Nodes	Worse Nodes	Important Nodes
mean	55.28	24.88	47.84	28.24	23.88	75.88	14.80	26.68	86.52	10.56	27.2	90.24	12.56	13.04	102.4
std	4.76	5.18	5.23	3.82	5.39	5.47	4.02	4.34	5.24	3.23	8.5	8.87	2.97	5.84	6.0
min	43.00	15.00	37.00	20.00	17.00	66.00	6.00	19.00	73.00	5.00	9.0	76.00	8.00	1.00	93.0
25%	53.00	22.00	44.00	27.00	19.00	72.00	12.00	23.00	85.00	8.00	21.0	84.00	11.00	11.00	98.0
50%	54.00	25.00	49.00	28.00	22.00	77.00	15.00	26.00	88.00	11.00	28.0	87.00	13.00	13.00	103.0
75%	56.00	28.00	52.00	30.00	27.00	81.00	18.00	30.00	89.00	12.00	33.0	96.00	14.00	15.00	107.0
max	63.00	35.00	55.00	36.00	34.00	85.00	22.00	34.00	94.00	16.00	41.0	110.00	19.00	24.00	112.0

Table B.7: Number of nodes in each class of nodes

	1			8			32			256			1024		
	Zero Nodes	Worse Nodes	Important Nodes	Zero Nodes	Worse Nodes	Important Nodes	Zero Nodes	Worse Nodes	Important Nodes	Zero Nodes	Worse Nodes	Important Nodes	Zero Nodes	Worse Nodes	Important Nodes
mean	-0.0	0.0020	-0.0121	-0.0	0.0016	-0.0040	-0.0	0.0016	-0.0032	-0.0	0.0038	-0.0075	-0.0	0.0017	-0.0076
std	0.0	0.0010	0.0026	0.0	0.0008	0.0006	0.0	0.0007	0.0004	0.0	0.0023	0.0018	0.0	0.0009	0.0007
min	-0.0	0.0007	-0.0175	-0.0	0.0003	-0.0054	-0.0	0.0004	-0.0045	-0.0	0.0009	-0.0109	-0.0	0.0006	-0.0092
25%	-0.0	0.0012	-0.0137	-0.0	0.0012	-0.0042	-0.0	0.0011	-0.0033	-0.0	0.0019	-0.0089	-0.0	0.0010	-0.0081
50%	0.0	0.0017	-0.0119	-0.0	0.0015	-0.0039	-0.0	0.0015	-0.0031	-0.0	0.0028	-0.0069	-0.0	0.0014	-0.0074
75%	0.0	0.0028	-0.0108	0.0	0.0018	-0.0036	0.0	0.0019	-0.0029	0.0	0.0055	-0.0063	0.0	0.0020	-0.0071
max	0.0	0.0042	-0.0081	0.0	0.0037	-0.0030	0.0	0.0036	-0.0027	0.0	0.0081	-0.0047	0.0	0.0037	-0.0066

Table B.8: Average importance for each node class

B.3 Effects of dropout on Node Importance

B.3.1 MNIST

Tables

	0.1			0.3			0.5			0.7			0.9		
	Zero Nodes	Worse Nodes	Important Nodes	Zero Nodes	Worse Nodes	Important Nodes	Zero Nodes	Worse Nodes	Important Nodes	Zero Nodes	Worse Nodes	Important Nodes	Zero Nodes	Worse Nodes	Important Nodes
mean	2.20	23.68	102.12	2.60	23.24	102.16	3.36	25.52	99.12	4.24	32.32	91.44	9.16	5.32	113.52
std	1.47	5.76	5.70	1.66	7.19	7.22	1.87	5.12	5.73	1.79	7.43	6.93	2.30	4.10	5.03
min	0.00	13.00	85.00	0.00	8.00	91.00	1.00	14.00	89.00	1.00	21.00	77.00	4.00	1.00	101.00
25%	1.00	21.00	100.00	2.00	19.00	96.00	1.00	22.00	94.00	3.00	27.00	87.00	8.00	3.00	112.00
50%	2.00	24.00	103.00	2.00	22.00	104.00	4.00	27.00	99.00	4.00	32.00	93.00	9.00	4.00	115.00
75%	3.00	25.00	105.00	3.00	30.00	105.00	4.00	29.00	103.00	6.00	38.00	95.00	11.00	6.00	116.00
max	6.00	41.00	110.00	7.00	36.00	120.00	7.00	33.00	113.00	7.00	45.00	102.00	14.00	17.00	122.00

Table B.9: Number of nodes in each class of nodes

	0.1			0.3			0.5			0.7			0.9		
	Zero Nodes	Worse Nodes	Important Nodes	Zero Nodes	Worse Nodes	Important Nodes	Zero Nodes	Worse Nodes	Important Nodes	Zero Nodes	Worse Nodes	Important Nodes	Zero Nodes	Worse Nodes	Important Nodes
mean	-0.0	0.0005	-0.0013	0.0	0.0004	-0.0009	-0.0	0.0004	-0.0007	0.0	0.0004	-0.0006	-0.0	0.0002	-0.0013
std	0.0	0.0002	0.0002	0.0	0.0001	0.0001	0.0	0.0001	0.0001	0.0	0.0001	0.0001	0.0	0.0001	0.0001
min	-0.0	0.0002	-0.0019	-0.0	0.0002	-0.0011	-0.0	0.0002	-0.0009	-0.0	0.0002	-0.0007	-0.0	0.0000	-0.0015
25%	-0.0	0.0004	-0.0013	-0.0	0.0003	-0.0009	-0.0	0.0003	-0.0007	-0.0	0.0003	-0.0007	-0.0	0.0001	-0.0013
50%	0.0	0.0005	-0.0012	0.0	0.0004	-0.0008	0.0	0.0003	-0.0007	0.0	0.0004	-0.0006	-0.0	0.0002	-0.0012
75%	0.0	0.0007	-0.0012	0.0	0.0005	-0.0008	0.0	0.0004	-0.0006	0.0	0.0004	-0.0006	0.0	0.0002	-0.0012
max	0.0	0.0011	-0.0010	0.0	0.0007	-0.0007	0.0	0.0006	-0.0006	0.0	0.0005	-0.0005	0.0	0.0006	-0.0011

Table B.10: Average importance for each node class

B.3.2 Fashion MNIST

Figures

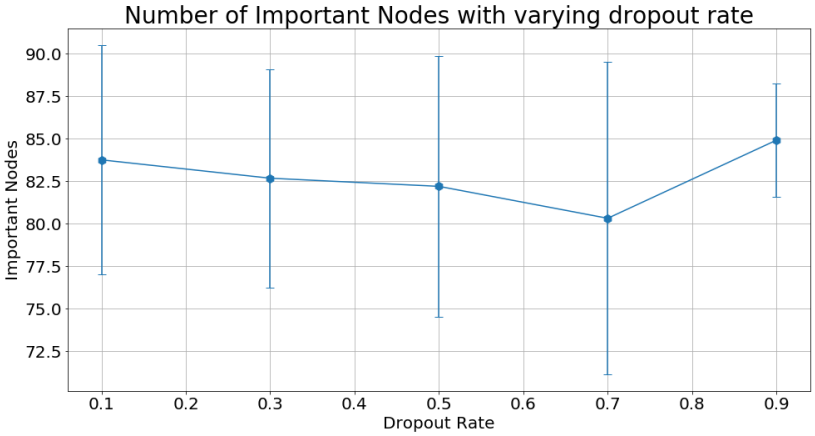

Figure B.1: Number of important nodes per dropout rate

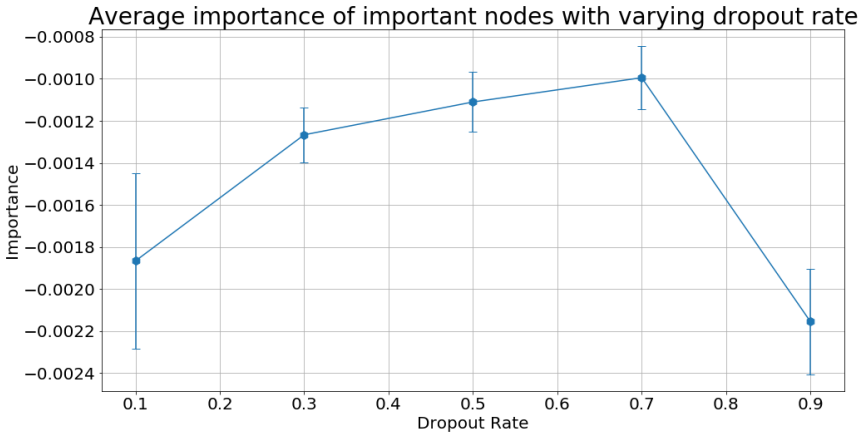

Figure B.2: Average importance for important nodes per dropout rate

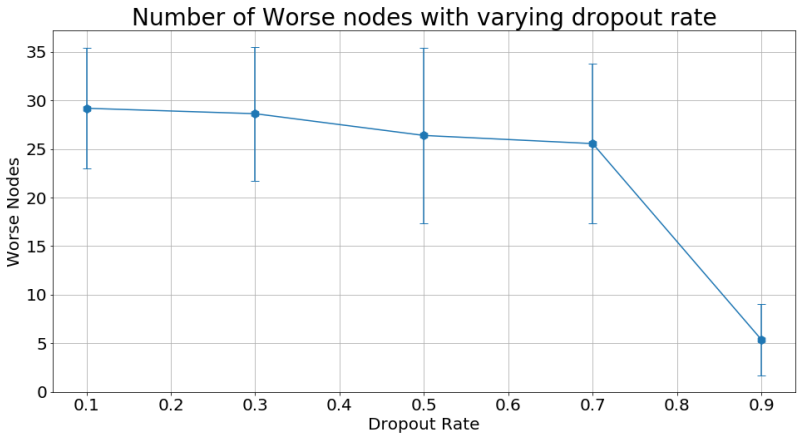

Figure B.3: Number of worse nodes per dropout rate

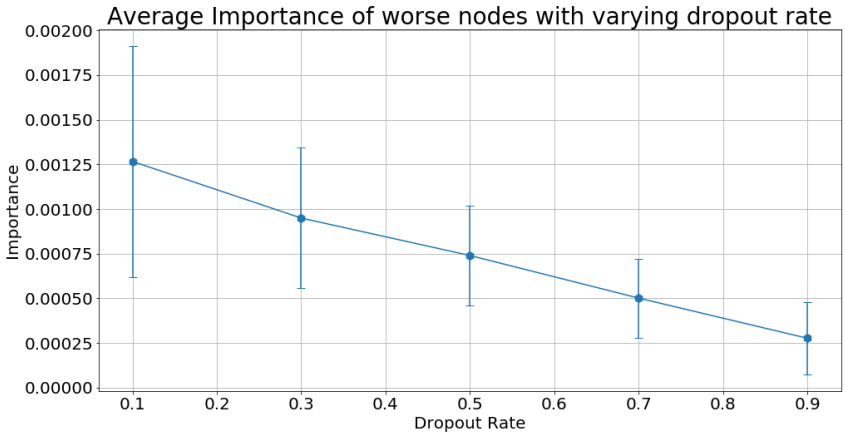

Figure B.4: Average importance for worse nodes per dropout rate

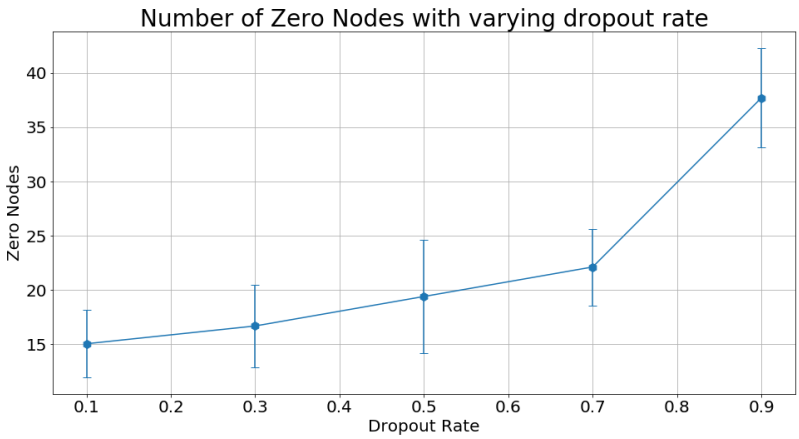

Figure B.5: Number of zero nodes per dropout rate

Tables

	0.1	0.3	0.5	0.7	0.9
mean	0.8697	0.8674	0.8599	0.8471	0.7813
std	0.0041	0.0029	0.0037	0.0034	0.0144
min	0.8596	0.8606	0.8517	0.8404	0.7392
25%	0.8677	0.8665	0.8581	0.8452	0.7807
50%	0.8706	0.8680	0.8600	0.8472	0.7855
75%	0.8726	0.8693	0.8622	0.8498	0.7894
max	0.8747	0.8730	0.8660	0.8522	0.8006

Table B.11: Statistics of accuracy of ANNs trained on Fashion MNIST

	0.1	0.3	0.5	0.7	0.9
mean	0.3610	0.3681	0.3889	0.4235	0.6109
std	0.0124	0.0067	0.0074	0.0058	0.0121
min	0.3450	0.3590	0.3764	0.4120	0.5844
25%	0.3546	0.3631	0.3848	0.4200	0.6032
50%	0.3565	0.3674	0.3882	0.4232	0.6128
75%	0.3650	0.3714	0.3921	0.4278	0.6189
max	0.4048	0.3859	0.4083	0.4349	0.6334

Table B.12: Statistics of loss of ANNs trained on Fashion MNIST

	0.1			0.3			0.5			0.7			0.9		
	Zero Nodes	Worse Nodes	Important Nodes	Zero Nodes	Worse Nodes	Important Nodes	Zero Nodes	Worse Nodes	Important Nodes	Zero Nodes	Worse Nodes	Important Nodes	Zero Nodes	Worse Nodes	Important Nodes
mean	15.04	29.20	83.76	16.68	28.64	82.68	19.40	26.40	82.20	22.12	25.56	80.32	37.72	5.36	84.92
std	3.12	6.22	6.76	3.78	6.88	6.43	5.21	9.01	7.69	3.54	8.21	9.20	4.59	3.71	3.32
min	7.00	16.00	66.00	10.00	13.00	71.00	11.00	14.00	64.00	17.00	14.00	61.00	26.00	0.00	80.00
25%	14.00	24.00	82.00	13.00	23.00	78.00	16.00	22.00	77.00	20.00	19.00	74.00	36.00	3.00	82.00
50%	15.00	30.00	84.00	17.00	29.00	82.00	20.00	26.00	82.00	22.00	23.00	83.00	38.00	5.00	85.00
75%	17.00	32.00	86.00	18.00	32.00	85.00	22.00	29.00	88.00	24.00	31.00	89.00	41.00	7.00	87.00
max	23.00	41.00	97.00	24.00	43.00	97.00	33.00	51.00	97.00	30.00	42.00	91.00	45.00	14.00	93.00

Table B.13: Number of nodes in each class of nodes

	0.1			0.3			0.5			0.7			0.9		
	Zero Nodes	Worse Nodes	Important Nodes	Zero Nodes	Worse Nodes	Important Nodes	Zero Nodes	Worse Nodes	Important Nodes	Zero Nodes	Worse Nodes	Important Nodes	Zero Nodes	Worse Nodes	Important Nodes
mean	-0.0	0.0013	-0.0019	0.0	0.0009	-0.0013	-0.0	0.0007	-0.0011	0.0	0.0005	-0.0010	0.0	0.0003	-0.0022
std	0.0	0.0006	0.0004	0.0	0.0004	0.0001	0.0	0.0003	0.0001	0.0	0.0002	0.0002	0.0	0.0002	0.0003
min	-0.0	0.0003	-0.0029	-0.0	0.0002	-0.0015	-0.0	0.0003	-0.0014	-0.0	0.0002	-0.0013	-0.0	0.0000	-0.0027
25%	-0.0	0.0008	-0.0020	-0.0	0.0007	-0.0014	-0.0	0.0005	-0.0012	-0.0	0.0004	-0.0011	-0.0	0.0002	-0.0023
50%	-0.0	0.0011	-0.0019	0.0	0.0009	-0.0013	-0.0	0.0007	-0.0011	0.0	0.0005	-0.0010	0.0	0.0002	-0.0022
75%	0.0	0.0016	-0.0016	0.0	0.0011	-0.0012	0.0	0.0009	-0.0010	0.0	0.0007	-0.0009	0.0	0.0003	-0.0020
max	0.0	0.0028	-0.0012	0.0	0.0020	-0.0010	0.0	0.0013	-0.0008	0.0	0.0012	-0.0007	0.0	0.0009	-0.0015

Table B.14: Average importance for each node class

B.4 Pre-calculated Pruning

B.4.1 MNIST

Tables - Training set

	Change in Accuracy
mean	-0.0155
std	0.0313
min	-0.1972
25%	-0.0165
50%	-0.0046
75%	-0.0010
max	0.0011

Table B.15: Statistics of change in accuracy of ANNs trained on MNIST

	Change in Loss
mean	0.0465
std	0.0928
min	-0.0025
25%	0.0047
50%	0.0145
75%	0.0494
max	0.5915

Table B.16: Statistics of change loss of ANNs trained on MNIST

Tables - Validation set

	Change in Accuracy
mean	-0.0361
std	0.0316
min	-0.1300
25%	-0.0469
50%	-0.0277
75%	-0.0113
max	-0.0009

Table B.17: Statistics of change accuracy of ANNs trained on MNIST

	Change in Loss
mean	0.1053
std	0.0926
min	0.0003
25%	0.0321
50%	0.0765
75%	0.1433
max	0.3680

Table B.18: Statistics of change loss of ANNs trained on MNIST

B.4.2 Fashion MNIST

Figures - Training set

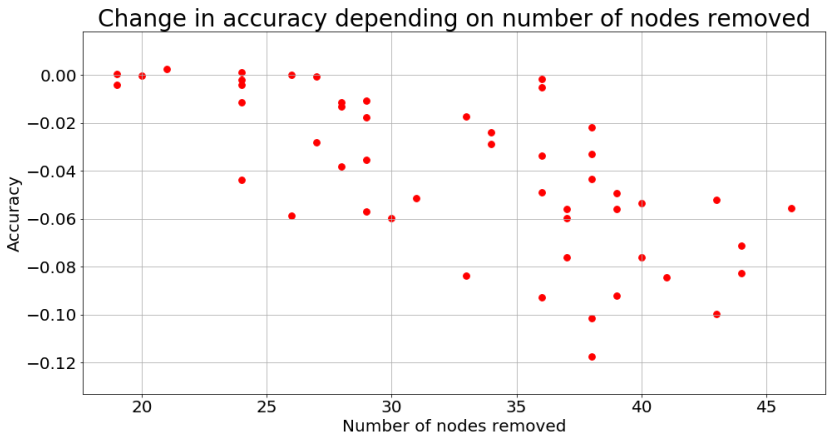

Figure B.6: Change in accuracy after pruning based on pre-calculated node importance

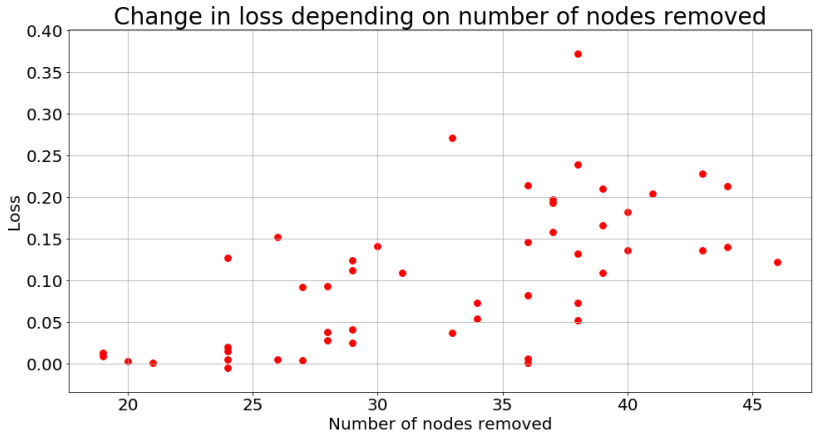

Figure B.7: Change in loss after pruning based on pre-calculated node importance

Figures - Validation set

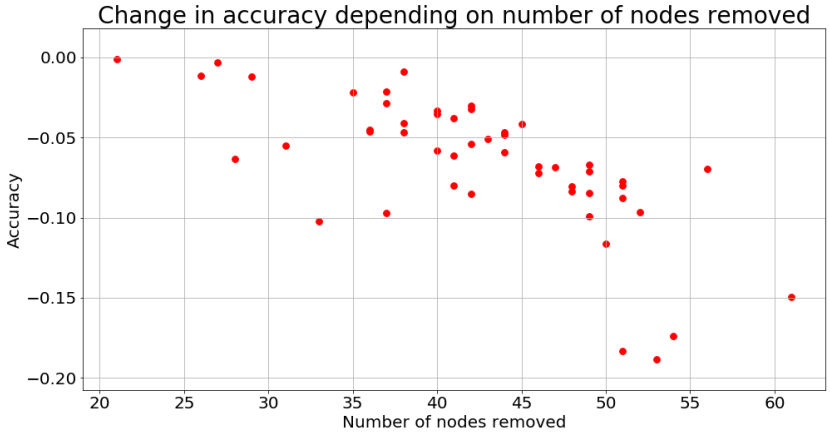

Figure B.8: Change in accuracy after pruning based on pre-calculated node importance

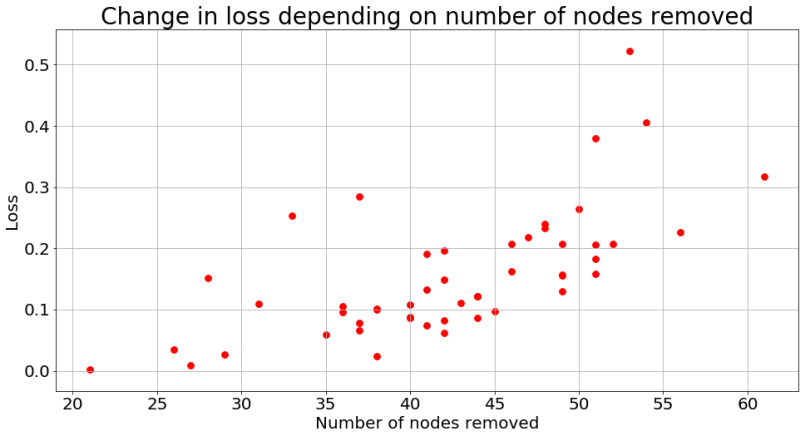

Figure B.9: Change in loss after pruning based on pre-calculated node importance

Tables - Training set

	Number of Nodes
mean	32.78
std	7.19
min	19.00
25%	27.25
50%	34.00
75%	38.00
max	46.00

Table B.19: Statistics of number of nodes pruned from ANNs trained on Fashion MNIST

	Change in Accuracy
mean	-0.0412
std	0.0328
min	-0.1173
25%	-0.0593
50%	-0.0407
75%	-0.0114
max	0.0024

Table B.20: Statistics of change in accuracy of ANNs trained on MNIST

	Change in Loss
mean	0.1062
std	0.0865
min	-0.0051
25%	0.0260
50%	0.1096
75%	0.1571
max	0.3725

Table B.21: Statistics of change loss of ANNs trained on MNIST

Tables - Validation set

	Number of Nodes
mean	42.38
std	8.33
min	21.00
25%	37.25
50%	42.00
75%	49.00
max	61.00

Table B.22: Statistics of number of nodes pruned from ANNs trained on Fashion MNIST

	Change in Accuracy
mean	-0.0656
std	0.0424
min	-0.1885
25%	-0.0827
50%	-0.0602
75%	-0.0387
max	-0.0011

Table B.23: Statistics of change accuracy of ANNs trained on MNIST

	Change in Loss
mean	0.1558
std	0.1035
min	0.0026
25%	0.0873
50%	0.1313
75%	0.2070
max	0.5218

Table B.24: Statistics of change loss of ANNs trained on MNIST

B.5 Exhaustive Pruning

B.5.1 MNIST

Tables - Training set

	Nodes removed
mean	6.65
std	1.95
min	4.00
25%	5.00
50%	6.50
75%	7.25
max	11.00

Table B.25: Statistics of number of nodes pruned from ANNs trained on MNIST

	Accuracy
mean	0.0015
std	0.0011
min	-0.0002
25%	0.0007
50%	0.0015
75%	0.0020
max	0.0035

Table B.26: Statistics of change in accuracy of ANNs trained on MNIST

	Loss
mean	-0.0050
std	0.0034
min	-0.0112
25%	-0.0081
50%	-0.0043
75%	-0.0021
max	-0.0009

Table B.27: Statistics of change loss of ANNs trained on MNIST

Tables - Validation set

	Nodes removed
mean	12.70
std	2.92
min	8.00
25%	11.50
50%	14.00
75%	15.00
max	17.00

Table B.28: Statistics of number of nodes pruned from ANNs trained on MNIST

	Accuracy
mean	0.0013
std	0.0016
min	-0.0013
25%	-0.0001
50%	0.0014
75%	0.0024
max	0.0040

Table B.29: Statistics of change accuracy of ANNs trained on MNIST

	Loss
mean	-0.0051
std	0.0040
min	-0.0119
25%	-0.0077
50%	-0.0060
75%	-0.0017
max	0.0029

Table B.30: Statistics of change loss of ANNs trained on MNIST

B.5.2 Fashion MNIST

Figures - Training set

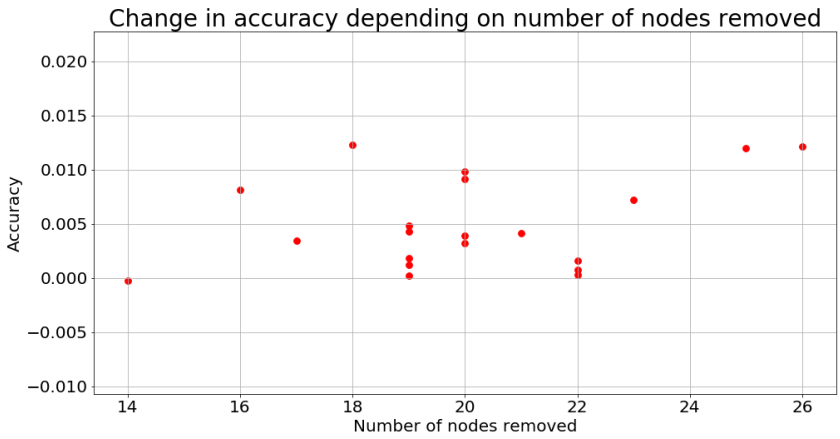

Figure B.10: Change in accuracy after pruning with exhaustive method

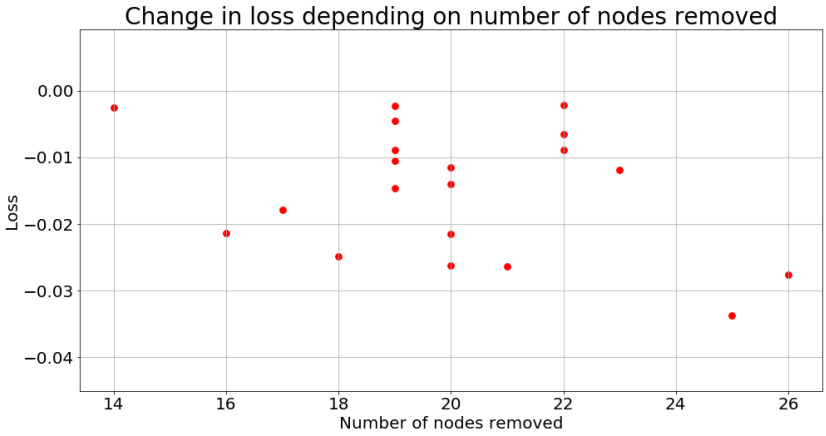

Figure B.11: Change in loss after pruning bwith exhaustive method

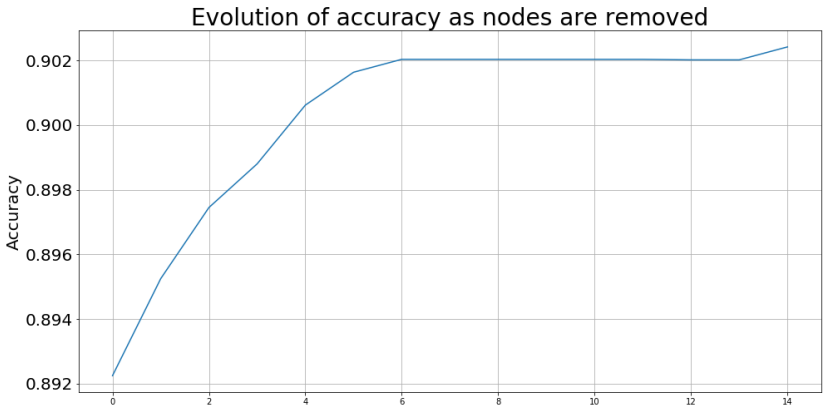

Figure B.12: Evolution of training accuracy as an ANN is pruned

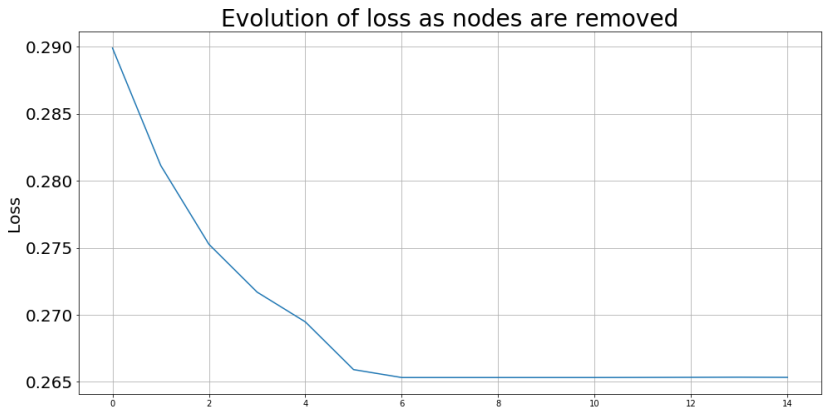

Figure B.13: Evolution of training loss as an ANN is pruned

Figures - Validation set

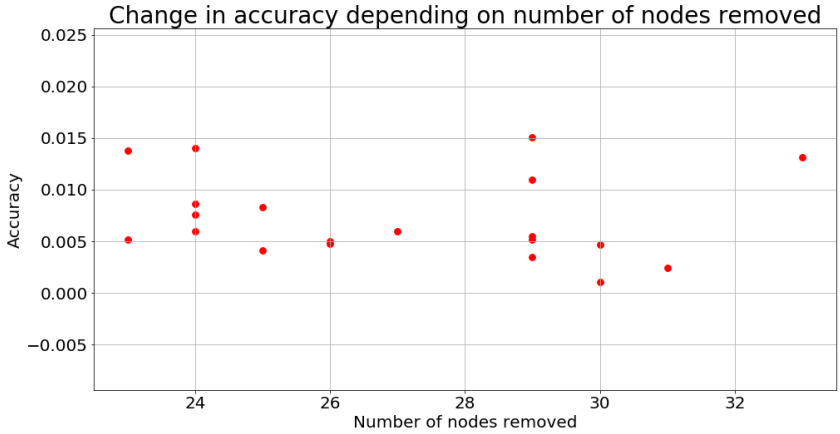

Figure B.14: Change in accuracy with exhaustive method

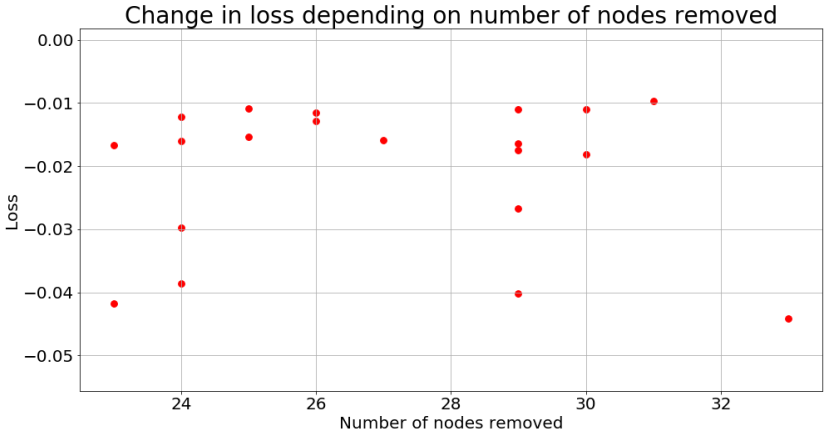

Figure B.15: Change in loss after pruning with exhaustive method

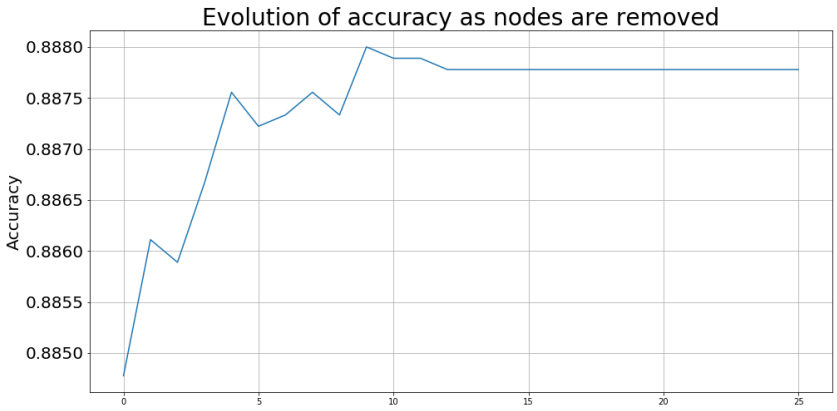

Figure B.16: Evolution of validation accuracy as an ANN is pruned

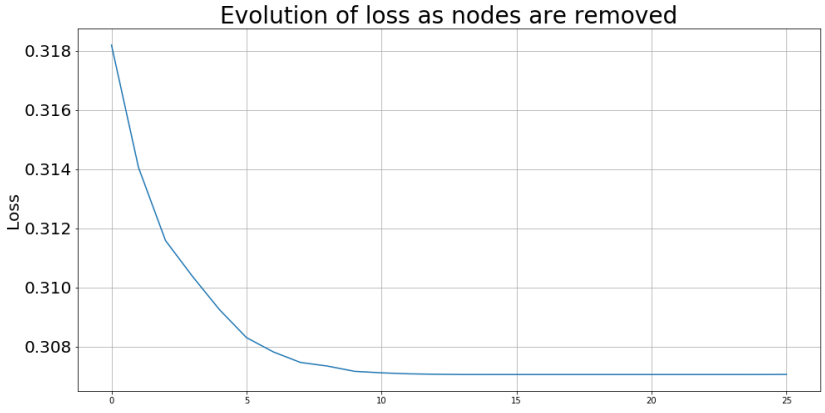

Figure B.17: Evolution of validation loss as an ANN is pruned

Tables - Training set

	Nodes removed
mean	20.05
std	2.84
min	14.00
25%	19.00
50%	20.00
75%	22.00
max	26.00

Table B.31: Statistics of number of nodes pruned from ANNs trained on Fashion MNIST

	Accuracy
mean	0.0050
std	0.0043
min	-0.0003
25%	0.0015
50%	0.0040
75%	0.0084
max	0.0123

Table B.32: Statistics of change in accuracy of ANNs trained on Fashion MNIST

	Loss
mean	-0.0149
std	0.0095
min	-0.0337
25%	-0.0224
50%	-0.0129
75%	-0.0083
max	-0.0022

Table B.33: Statistics of change loss of ANNs trained on Fashion MNIST

Tables - Validation set

	Nodes removed
mean	27.00
std	2.97
min	23.00
25%	24.00
50%	26.50
75%	29.00
max	33.00

Table B.34: Statistics of number of nodes pruned from ANNs trained on Fashion MNIST

	Accuracy
mean	0.0073
std	0.0041
min	0.0011
25%	0.0048
50%	0.0057
75%	0.0092
max	0.0151

Table B.35: Statistics of change accuracy of ANNs trained on Fashion MNIST

	Loss
mean	-0.0208
std	0.0116
min	-0.0441
25%	-0.0275
50%	-0.0162
75%	-0.0120
max	-0.0097

Table B.36: Statistics of change loss of ANNs trained on Fashion MNIST

B.6 Iterative Weight Initialization

B.6.1 Fashion MNIST

Tables

	Unoptimized Weights	Optimized Weights
mean	0.8689	0.8691
std	0.0060	0.0061
min	0.8571	0.8562
25%	0.8643	0.8669
50%	0.8691	0.8698
75%	0.8745	0.8731
max	0.8789	0.8782

Table B.37: Statistics of accuracy of ANNs trained on Fashion MNIST

	Unoptimized Weights	Optimized Weights
mean	0.3647	0.3628
std	0.0141	0.0163
min	0.3426	0.3417
25%	0.3572	0.3505
50%	0.3619	0.3596
75%	0.3710	0.3667
max	0.4027	0.4062

Table B.38: Statistics of loss of ANNs trained on Fashion MNIST

Appendix C

MLP - Extra Figures and Tables

C.1 Estimating Node Importance

C.1.1 MNIST

Tables - Training set

	Zero Nodes	Worse Nodes	Important Nodes
mean	2.98	18.48	106.54
std	1.86	6.20	6.41
min	0.00	2.00	89.00
25%	2.00	14.25	102.00
50%	3.00	19.00	107.00
75%	4.00	23.00	110.75
max	7.00	34.00	125.00

Table C.1: Number of nodes in each class of nodes for the first layer of the MLP (128-node layer)

	Zero Nodes	Worse Nodes	Important Nodes
mean	2.06	9.42	52.52
std	1.27	4.70	4.90
min	0.00	0.00	42.00
25%	1.00	6.00	49.00
50%	2.00	9.00	53.00
75%	3.00	12.75	56.00
max	5.00	20.00	62.00

Table C.2: Number of nodes in each class of nodes for the second layer of the MLP (64-node layer)

	Zero Nodes	Worse Nodes	Important Nodes
mean	2.04	2.18	27.78
std	1.64	2.06	2.57
min	0.00	0.00	22.00
25%	0.25	1.00	26.00
50%	2.00	2.00	28.00
75%	3.00	3.00	30.00
max	5.00	8.00	32.00

Table C.3: Number of nodes in each class of nodes for the third layer of the MLP (32-node layer)

Tables - Validation set

	Zero Nodes	Worse Nodes	Important Nodes
mean	3.14	33.46	91.40
std	1.74	6.51	6.24
min	0.00	21.00	78.00
25%	2.00	29.00	86.00
50%	3.00	33.00	92.00
75%	4.00	38.75	96.00
max	8.00	48.00	102.00

Table C.4: Number of nodes in each class of nodes for the first layer of the MLP (128-node layer)

	Zero Nodes	Worse Nodes	Important Nodes
mean	2.22	16.38	45.4
std	1.74	4.04	4.2
min	0.00	7.00	35.0
25%	1.00	14.00	42.0
50%	2.00	16.00	46.0
75%	3.00	19.00	48.0
max	9.00	27.00	56.0

Table C.5: Number of nodes in each class of nodes for the second layer of the MLP (64-node layer)

	Zero Nodes	Worse Nodes	Important Nodes
mean	2.4	4.58	25.02
std	1.5	2.42	2.32
min	0.0	1.00	19.00
25%	1.0	3.00	23.25
50%	2.0	4.50	25.50
75%	4.0	5.00	27.00
max	5.0	11.00	30.00

Table C.6: Number of nodes in each class of nodes for the third layer of the MLP (32-node layer)

C.1.2 Fashion MNIST

Tables - Training set

	Zero Nodes	Worse Nodes	Important Nodes
mean	17.66	18.46	91.88
std	3.59	7.02	7.12
min	10.00	3.00	75.00
25%	15.25	13.00	87.00
50%	17.50	19.00	92.00
75%	19.75	24.00	97.00
max	26.00	34.00	106.00

Table C.7: Number of nodes in each class of nodes for the first layer of the MLP (128-node layer)

	Zero Nodes	Worse Nodes	Important Nodes
mean	4.16	12.20	47.64
std	2.16	5.06	5.26
min	1.00	0.00	36.00
25%	2.25	9.00	44.00
50%	4.00	12.00	47.50
75%	6.00	15.00	51.00
max	10.00	23.00	58.00

Table C.8: Number of nodes in each class of nodes for the second layer of the MLP (64-node layer)

	Zero Nodes	Worse Nodes	Important Nodes
mean	2.78	3.52	25.70
std	1.73	2.37	2.38
min	0.00	0.00	20.00
25%	2.00	2.00	24.00
50%	2.00	3.00	26.00
75%	4.00	4.75	27.75
max	9.00	9.00	30.00

Table C.9: Number of nodes in each class of nodes for the third layer of the MLP (32-node layer)

Tables - Validation set

	Zero Nodes	Worse Nodes	Important Nodes
mean	17.78	27.22	83.00
std	3.44	6.27	6.88
min	12.00	7.00	66.00
25%	16.00	24.00	78.25
50%	17.00	27.00	84.00
75%	19.00	31.75	86.75
max	27.00	39.00	105.00

Table C.10: Number of nodes in each class of nodes for the first layer of the MLP (128-node layer)

	Zero Nodes	Worse Nodes	Important Nodes
mean	3.64	16.92	43.44
std	1.88	4.81	4.85
min	0.00	7.00	32.00
25%	3.00	14.00	39.00
50%	3.00	17.00	44.00
75%	4.00	20.00	46.00
max	8.00	27.00	55.00

Table C.11: Number of nodes in each class of nodes for the second layer of the MLP (64-node layer)

	Zero Nodes	Worse Nodes	Important Nodes
mean	2.80	5.14	24.06
std	1.65	2.29	2.44
min	0.00	1.00	18.00
25%	2.00	3.00	22.00
50%	2.00	5.00	24.50
75%	4.00	6.75	26.00
max	6.00	10.00	29.00

Table C.12: Number of nodes in each class of nodes for the third layer of the MLP (32-node layer)

C.2 Effects of batch size on Node Importance

C.2.1 MNIST

Figures

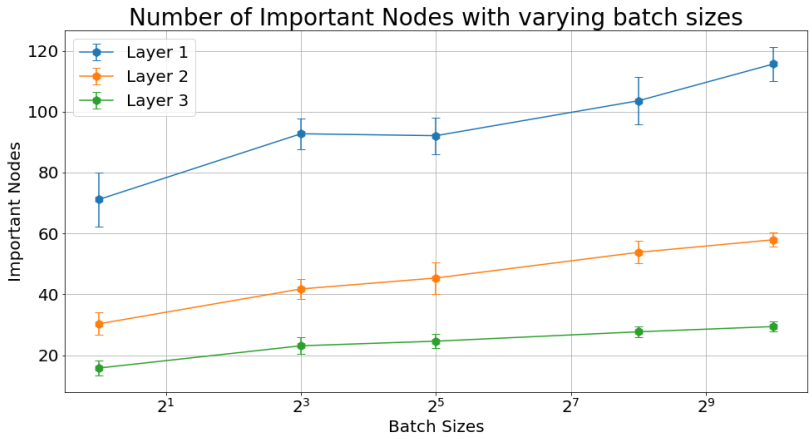

Figure C.1: Number of important nodes per batch size

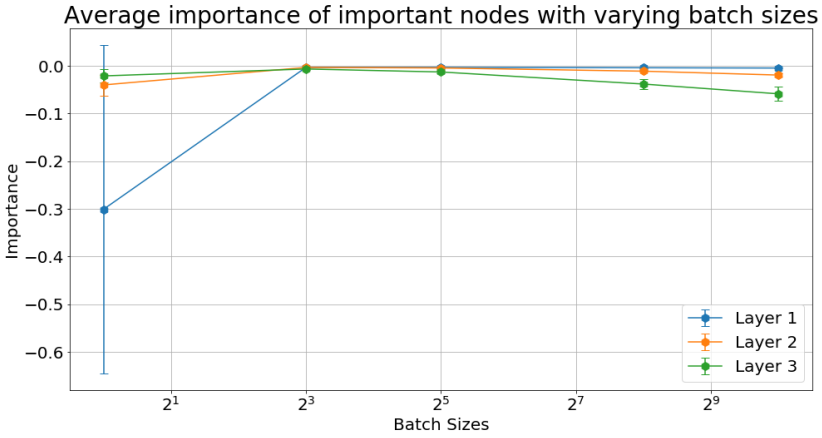

Figure C.2: Average node importance of important nodes per batch size

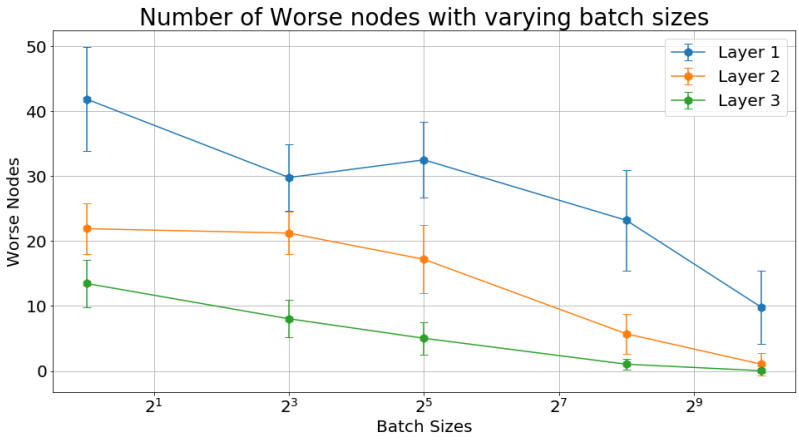

Figure C.3: Number of worse nodes per batch size

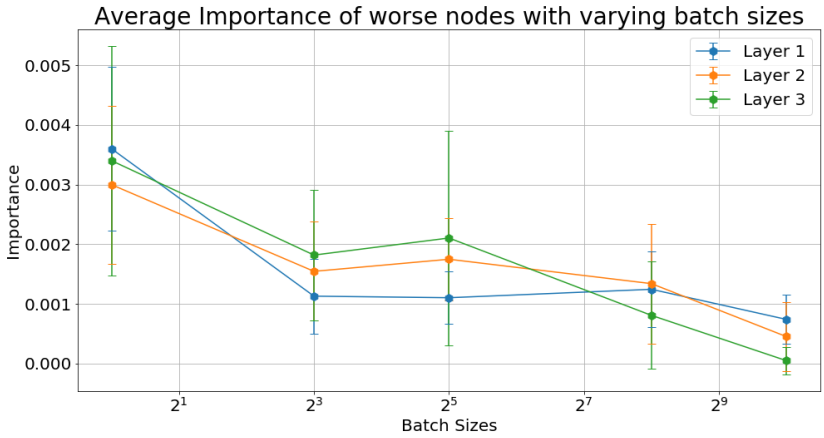

Figure C.4: Average node importance of worse nodes per batch size

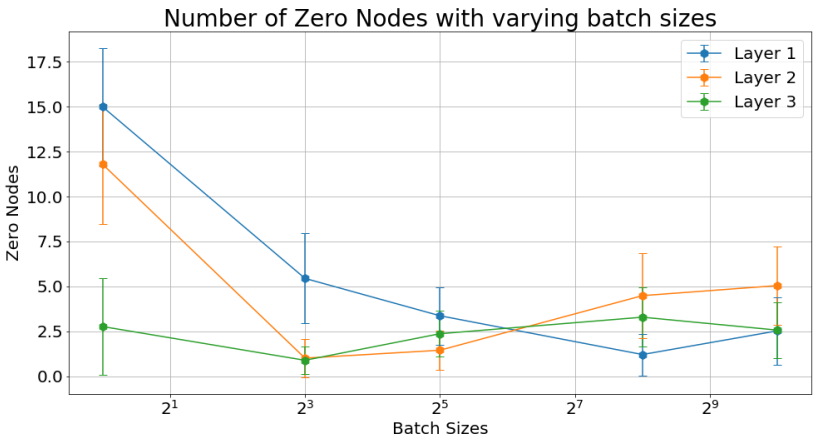

Figure C.5: Number of zero nodes per batch size

Tables

	1	8	32	256	1024
mean	0.9632	0.9729	0.9731	0.9696	0.9567
std	0.0041	0.0028	0.0026	0.0021	0.0020
min	0.9529	0.9674	0.9674	0.9637	0.9507
25%	0.9599	0.9709	0.9716	0.9683	0.9558
50%	0.9639	0.9723	0.9733	0.9700	0.9570
75%	0.9663	0.9743	0.9750	0.9712	0.9576
max	0.9695	0.9775	0.9782	0.9735	0.9599

Table C.13: Statistics of accuracy of MLPs trained on MNIST at each batch size

	1	8	32	256	1024
mean	0.2217	0.1008	0.0932	0.0998	0.1450
std	0.0312	0.0102	0.0091	0.0064	0.0063
min	0.1803	0.0828	0.0721	0.0875	0.1367
25%	0.2045	0.0937	0.0884	0.0967	0.1408
50%	0.2156	0.0993	0.0934	0.0997	0.1439
75%	0.2275	0.1075	0.0965	0.1038	0.1462
max	0.3225	0.1261	0.1176	0.1162	0.1603

Table C.14: Statistics of loss of MLPs trained on MNIST at each batch size

	1			8			32			256			1024		
	Zero Nodes	Worse Nodes	Important Nodes	Zero Nodes	Worse Nodes	Important Nodes	Zero Nodes	Worse Nodes	Important Nodes	Zero Nodes	Worse Nodes	Important Nodes	Zero Nodes	Worse Nodes	Important Nodes
mean	15.00	41.88	71.12	5.44	29.80	92.76	3.36	32.52	92.12	1.20	23.20	103.60	2.52	9.80	115.68
std	3.27	8.04	8.80	2.50	5.13	5.11	1.60	5.80	6.04	1.15	7.74	7.86	1.87	5.68	5.61
min	9.00	20.00	56.00	2.00	21.00	83.00	0.00	19.00	80.00	0.00	8.00	92.00	0.00	1.00	104.00
25%	14.00	36.00	65.00	3.00	26.00	91.00	2.00	30.00	88.00	0.00	16.00	98.00	1.00	5.00	112.00
50%	14.00	44.00	70.00	5.00	29.00	92.00	3.00	32.00	92.00	1.00	25.00	102.00	2.00	10.00	115.00
75%	17.00	49.00	75.00	7.00	32.00	97.00	5.00	36.00	95.00	2.00	28.00	110.00	3.00	13.00	119.00
max	21.00	54.00	89.00	11.00	41.00	102.00	6.00	45.00	107.00	4.00	35.00	119.00	7.00	23.00	125.00

Table C.15: Number of nodes in each class of nodes for the first layer of the MLP (128-node layer)

	1			8			32			256			1024		
	Zero Nodes	Worse Nodes	Important Nodes	Zero Nodes	Worse Nodes	Important Nodes	Zero Nodes	Worse Nodes	Important Nodes	Zero Nodes	Worse Nodes	Important Nodes	Zero Nodes	Worse Nodes	Important Nodes
mean	11.80	21.92	30.28	1.00	21.24	41.76	1.44	17.20	45.36	4.48	5.72	53.80	5.04	1.04	57.92
std	3.32	3.96	3.68	1.08	3.28	3.36	1.08	5.28	5.23	2.35	3.10	3.66	2.17	1.77	2.31
min	7.00	13.00	24.00	0.00	16.00	35.00	0.00	6.00	33.00	1.00	1.00	46.00	1.00	0.00	52.00
25%	9.00	19.00	28.00	0.00	19.00	38.00	1.00	14.00	42.00	3.00	3.00	52.00	3.00	0.00	57.00
50%	11.00	22.00	30.00	1.00	21.00	43.00	1.00	16.00	46.00	4.00	6.00	53.00	5.00	1.00	58.00
75%	14.00	25.00	34.00	1.00	23.00	43.00	2.00	20.00	48.00	6.00	8.00	56.00	6.00	1.00	59.00
max	18.00	28.00	37.00	3.00	28.00	47.00	3.00	31.00	57.00	9.00	13.00	61.00	11.00	7.00	62.00

Table C.16: Number of nodes in each class of nodes for the second layer of the MLP (64-node layer)

	1			8			32			256			1024		
	Zero Nodes	Worse Nodes	Important Nodes	Zero Nodes	Worse Nodes	Important Nodes	Zero Nodes	Worse Nodes	Important Nodes	Zero Nodes	Worse Nodes	Important Nodes	Zero Nodes	Worse Nodes	Important Nodes
mean	2.76	13.48	15.76	0.88	8.04	23.08	2.36	5.04	24.60	3.28	1.04	27.68	2.56	0.04	29.40
std	2.68	3.62	2.50	0.78	2.89	2.71	1.29	2.51	2.35	1.65	0.84	1.73	1.56	0.20	1.55
min	0.00	3.00	10.00	0.00	3.00	19.00	0.00	0.00	21.00	1.00	0.00	24.00	0.00	0.00	25.00
25%	1.00	11.00	14.00	0.00	6.00	21.00	2.00	3.00	23.00	2.00	1.00	27.00	2.00	0.00	29.00
50%	1.00	14.00	16.00	1.00	8.00	23.00	2.00	5.00	25.00	3.00	1.00	28.00	3.00	0.00	29.00
75%	4.00	15.00	17.00	1.00	10.00	26.00	3.00	7.00	26.00	5.00	1.00	29.00	3.00	0.00	30.00
max	9.00	22.00	20.00	3.00	13.00	28.00	6.00	9.00	29.00	7.00	3.00	31.00	7.00	1.00	32.00

Table C.17: Number of nodes in each class of nodes for the third layer of the MLP (32-node layer)

	1			8			32			256			1024		
	Zero Nodes	Worse Nodes	Important Nodes	Zero Nodes	Worse Nodes	Important Nodes	Zero Nodes	Worse Nodes	Important Nodes	Zero Nodes	Worse Nodes	Important Nodes	Zero Nodes	Worse Nodes	Important Nodes
mean	-0.0	0.0036	-0.3009	0.0	0.0011	-0.0031	0.0	0.0011	-0.0032	0.0	0.0012	-0.0039	-0.0	0.0007	-0.0045
std	0.0	0.0014	0.3449	0.0	0.0006	0.0008	0.0	0.0004	0.0007	0.0	0.0006	0.0006	0.0	0.0004	0.0005
min	-0.0	0.0014	-1.3614	-0.0	0.0004	-0.0052	-0.0	0.0003	-0.0048	-0.0	0.0004	-0.0060	-0.0	0.0001	-0.0057
25%	-0.0	0.0023	-0.3223	-0.0	0.0008	-0.0032	0.0	0.0008	-0.0035	0.0	0.0008	-0.0043	-0.0	0.0004	-0.0047
50%	0.0	0.0037	-0.2173	0.0	0.0009	-0.0029	0.0	0.0011	-0.0030	0.0	0.0012	-0.0038	-0.0	0.0007	-0.0045
75%	0.0	0.0046	-0.0535	0.0	0.0013	-0.0027	0.0	0.0013	-0.0028	0.0	0.0016	-0.0035	0.0	0.0010	-0.0043
max	0.0	0.0066	-0.0223	0.0	0.0035	-0.0022	0.0	0.0019	-0.0020	0.0	0.0028	-0.0030	0.0	0.0017	-0.0037

Table C.18: Average importance for each node class for the first layer of the MLP (128-node layer)

	1			8			32			256			1024		
	Zero Nodes	Worse Nodes	Important Nodes	Zero Nodes	Worse Nodes	Important Nodes	Zero Nodes	Worse Nodes	Important Nodes	Zero Nodes	Worse Nodes	Important Nodes	Zero Nodes	Worse Nodes	Important Nodes
mean	-0.0	0.0036	-0.3009	0.0	0.0011	-0.0031	0.0	0.0011	-0.0032	0.0	0.0012	-0.0039	-0.0	0.0007	-0.0045
std	0.0	0.0014	0.3449	0.0	0.0006	0.0008	0.0	0.0004	0.0007	0.0	0.0006	0.0006	0.0	0.0004	0.0005
min	-0.0	0.0014	-1.3614	-0.0	0.0004	-0.0052	-0.0	0.0003	-0.0048	-0.0	0.0004	-0.0060	-0.0	0.0001	-0.0057
25%	-0.0	0.0023	-0.3223	-0.0	0.0008	-0.0032	0.0	0.0008	-0.0035	0.0	0.0008	-0.0043	-0.0	0.0004	-0.0047
50%	0.0	0.0037	-0.2173	0.0	0.0009	-0.0029	0.0	0.0011	-0.0030	0.0	0.0012	-0.0038	-0.0	0.0007	-0.0045
75%	0.0	0.0046	-0.0535	0.0	0.0013	-0.0027	0.0	0.0013	-0.0028	0.0	0.0016	-0.0035	0.0	0.0010	-0.0043
max	0.0	0.0066	-0.0223	0.0	0.0035	-0.0022	0.0	0.0019	-0.0020	0.0	0.0028	-0.0030	0.0	0.0017	-0.0037

Table C.19: Average importance for each node class for the second layer of the MLP (64-node layer)

	1			8			32			256			1024		
	Zero Nodes	Worse Nodes	Important Nodes	Zero Nodes	Worse Nodes	Important Nodes	Zero Nodes	Worse Nodes	Important Nodes	Zero Nodes	Worse Nodes	Important Nodes	Zero Nodes	Worse Nodes	Important Nodes
mean	-0.0	0.0036	-0.3009	0.0	0.0011	-0.0031	0.0	0.0011	-0.0032	0.0	0.0012	-0.0039	-0.0	0.0007	-0.0045
std	0.0	0.0014	0.3449	0.0	0.0006	0.0008	0.0	0.0004	0.0007	0.0	0.0006	0.0006	0.0	0.0004	0.0005
min	-0.0	0.0014	-1.3614	-0.0	0.0004	-0.0052	-0.0	0.0003	-0.0048	-0.0	0.0004	-0.0060	-0.0	0.0001	-0.0057
25%	-0.0	0.0023	-0.3223	-0.0	0.0008	-0.0032	0.0	0.0008	-0.0035	0.0	0.0008	-0.0043	-0.0	0.0004	-0.0047
50%	0.0	0.0037	-0.2173	0.0	0.0009	-0.0029	0.0	0.0011	-0.0030	0.0	0.0012	-0.0038	-0.0	0.0007	-0.0045
75%	0.0	0.0046	-0.0535	0.0	0.0013	-0.0027	0.0	0.0013	-0.0028	0.0	0.0016	-0.0035	0.0	0.0010	-0.0043
max	0.0	0.0066	-0.0223	0.0	0.0035	-0.0022	0.0	0.0019	-0.0020	0.0	0.0028	-0.0030	0.0	0.0017	-0.0037

Table C.20: Average importance for each node class for the third layer of the MLP (32-node layer)

C.2.2 Fashion MNIST

Tables

	1			8			32			256			1024		
	Zero Nodes	Worse Nodes	Important Nodes	Zero Nodes	Worse Nodes	Important Nodes	Zero Nodes	Worse Nodes	Important Nodes	Zero Nodes	Worse Nodes	Important Nodes	Zero Nodes	Worse Nodes	Important Nodes
mean	57.48	29.68	40.84	33.16	24.00	70.84	17.84	28.64	81.52	10.60	27.60	89.80	10.68	18.16	99.16
std	4.19	7.08	6.85	4.37	6.63	6.69	3.64	6.61	7.57	3.15	6.56	6.44	3.26	8.74	10.05
min	47.00	20.00	27.00	22.00	11.00	54.00	12.00	15.00	64.00	4.00	17.00	79.00	6.00	3.00	73.00
25%	54.00	24.00	37.00	32.00	20.00	68.00	16.00	24.00	78.00	9.00	23.00	86.00	8.00	13.00	94.00
50%	57.00	27.00	40.00	33.00	24.00	71.00	18.00	31.00	80.00	11.00	29.00	89.00	11.00	17.00	100.00
75%	61.00	37.00	46.00	35.00	29.00	76.00	19.00	34.00	84.00	13.00	31.00	96.00	12.00	24.00	106.00
max	64.00	43.00	54.00	40.00	37.00	81.00	26.00	38.00	100.00	16.00	44.00	102.00	19.00	36.00	116.00

Table C.21: Number of nodes in each class of nodes for the first layer of the MLP (128-node layer)

	1			8			32			256			1024		
	Zero Nodes	Worse Nodes	Important Nodes	Zero Nodes	Worse Nodes	Important Nodes	Zero Nodes	Worse Nodes	Important Nodes	Zero Nodes	Worse Nodes	Important Nodes	Zero Nodes	Worse Nodes	Important Nodes
mean	24.12	20.64	19.24	3.0	20.84	40.16	4.40	16.88	42.72	6.64	8.56	48.80	9.00	3.84	51.16
std	5.15	4.77	3.87	1.8	4.89	5.11	2.68	5.11	5.16	2.36	4.16	4.16	2.92	2.56	3.68
min	15.00	12.00	14.00	0.0	12.00	30.00	0.00	6.00	37.00	2.00	3.00	39.00	5.00	0.00	44.00
25%	20.00	17.00	17.00	2.0	18.00	36.00	3.00	15.00	38.00	5.00	6.00	47.00	7.00	2.00	49.00
50%	24.00	20.00	19.00	3.0	19.00	41.00	4.00	17.00	42.00	7.00	8.00	49.00	8.00	4.00	52.00
75%	28.00	24.00	21.00	4.0	25.00	43.00	6.00	21.00	45.00	7.00	10.00	52.00	11.00	5.00	54.00
max	33.00	29.00	28.00	6.0	32.00	50.00	10.00	25.00	53.00	12.00	21.00	57.00	16.00	12.00	56.00

Table C.22: Number of nodes in each class of nodes for the second layer of the MLP (64-node layer)

	1			8			32			256			1024		
	Zero Nodes	Worse Nodes	Important Nodes	Zero Nodes	Worse Nodes	Important Nodes	Zero Nodes	Worse Nodes	Important Nodes	Zero Nodes	Worse Nodes	Important Nodes	Zero Nodes	Worse Nodes	Important Nodes
mean	5.52	12.52	13.96	1.68	8.76	21.56	3.48	5.00	23.52	3.40	1.88	26.72	5.40	0.56	26.04
std	3.16	3.60	2.79	1.22	2.96	3.14	1.94	1.98	2.45	1.76	1.13	1.99	2.25	0.65	2.52
min	1.00	5.00	11.00	0.00	3.00	14.00	1.00	1.00	20.00	1.00	0.00	22.00	1.00	0.00	21.00
25%	4.00	9.00	12.00	1.00	7.00	20.00	2.00	4.00	22.00	2.00	1.00	25.00	4.00	0.00	24.00
50%	4.00	13.00	14.00	1.00	9.00	22.00	3.00	5.00	23.00	3.00	2.00	27.00	5.00	0.00	26.00
75%	6.00	14.00	14.00	2.00	11.00	24.00	5.00	6.00	25.00	5.00	2.00	28.00	7.00	1.00	28.00
max	13.00	19.00	21.00	4.00	15.00	27.00	7.00	9.00	29.00	7.00	5.00	30.00	11.00	2.00	31.00

Table C.23: Number of nodes in each class of nodes for the third layer of the MLP (32-node layer)

	1			8			32			256			1024		
	Zero Nodes	Worse Nodes	Important Nodes	Zero Nodes	Worse Nodes	Important Nodes	Zero Nodes	Worse Nodes	Important Nodes	Zero Nodes	Worse Nodes	Important Nodes	Zero Nodes	Worse Nodes	Important Nodes
mean	0.0	0.0039	-0.4106	-0.0	0.0015	-0.0055	-0.0	0.0017	-0.0035	-0.0	0.0028	-0.0065	-0.0	0.0028	-0.0130
std	0.0	0.0019	0.4056	0.0	0.0011	0.0016	0.0	0.0009	0.0007	0.0	0.0015	0.0012	0.0	0.0022	0.0034
min	-0.0	0.0013	-1.4856	-0.0	0.0005	-0.0103	-0.0	0.0003	-0.0057	-0.0	0.0008	-0.0099	-0.0	0.0002	-0.0272
25%	-0.0	0.0026	-0.6050	-0.0	0.0008	-0.0057	-0.0	0.0010	-0.0039	-0.0	0.0015	-0.0071	-0.0	0.0015	-0.0134
50%	0.0	0.0036	-0.2745	-0.0	0.0012	-0.0055	-0.0	0.0014	-0.0035	-0.0	0.0029	-0.0061	-0.0	0.0020	-0.0123
75%	0.0	0.0049	-0.1248	0.0	0.0016	-0.0040	0.0	0.0024	-0.0030	0.0	0.0036	-0.0058	0.0	0.0035	-0.0113
max	0.0	0.0084	-0.0703	0.0	0.0053	-0.0036	0.0	0.0034	-0.0022	0.0	0.0069	-0.0048	0.0	0.0110	-0.0097

Table C.24: Average importance for each node class for the first layer of the MLP (128-node layer)

	1			8			32			256			1024		
	Zero Nodes	Worse Nodes	Important Nodes	Zero Nodes	Worse Nodes	Important Nodes	Zero Nodes	Worse Nodes	Important Nodes	Zero Nodes	Worse Nodes	Important Nodes	Zero Nodes	Worse Nodes	Important Nodes
mean	0.0	0.0039	-0.4106	-0.0	0.0015	-0.0055	-0.0	0.0017	-0.0035	-0.0	0.0028	-0.0065	-0.0	0.0028	-0.0130
std	0.0	0.0019	0.4056	0.0	0.0011	0.0016	0.0	0.0009	0.0007	0.0	0.0015	0.0012	0.0	0.0022	0.0034
min	-0.0	0.0013	-1.4856	-0.0	0.0005	-0.0103	-0.0	0.0003	-0.0057	-0.0	0.0008	-0.0099	-0.0	0.0002	-0.0272
25%	-0.0	0.0026	-0.6050	-0.0	0.0008	-0.0057	-0.0	0.0010	-0.0039	-0.0	0.0015	-0.0071	-0.0	0.0015	-0.0134
50%	0.0	0.0036	-0.2745	-0.0	0.0012	-0.0055	-0.0	0.0014	-0.0035	-0.0	0.0029	-0.0061	-0.0	0.0020	-0.0123
75%	0.0	0.0049	-0.1248	0.0	0.0016	-0.0040	0.0	0.0024	-0.0030	0.0	0.0036	-0.0058	0.0	0.0035	-0.0113
max	0.0	0.0084	-0.0703	0.0	0.0053	-0.0036	0.0	0.0034	-0.0022	0.0	0.0069	-0.0048	0.0	0.0110	-0.0097

Table C.25: Average importance for each node class for the second layer of the MLP (64-node layer)

	1			8			32			256			1024		
	Zero Nodes	Worse Nodes	Important Nodes	Zero Nodes	Worse Nodes	Important Nodes	Zero Nodes	Worse Nodes	Important Nodes	Zero Nodes	Worse Nodes	Important Nodes	Zero Nodes	Worse Nodes	Important Nodes
mean	0.0	0.0039	-0.4106	-0.0	0.0015	-0.0055	-0.0	0.0017	-0.0035	-0.0	0.0028	-0.0065	-0.0	0.0028	-0.0130
std	0.0	0.0019	0.4056	0.0	0.0011	0.0016	0.0	0.0009	0.0007	0.0	0.0015	0.0012	0.0	0.0022	0.0034
min	-0.0	0.0013	-1.4856	-0.0	0.0005	-0.0103	-0.0	0.0003	-0.0057	-0.0	0.0008	-0.0099	-0.0	0.0002	-0.0272
25%	-0.0	0.0026	-0.6050	-0.0	0.0008	-0.0057	-0.0	0.0010	-0.0039	-0.0	0.0015	-0.0071	-0.0	0.0015	-0.0134
50%	0.0	0.0036	-0.2745	-0.0	0.0012	-0.0055	-0.0	0.0014	-0.0035	-0.0	0.0029	-0.0061	-0.0	0.0020	-0.0123
75%	0.0	0.0049	-0.1248	0.0	0.0016	-0.0040	0.0	0.0024	-0.0030	0.0	0.0036	-0.0058	0.0	0.0035	-0.0113
max	0.0	0.0084	-0.0703	0.0	0.0053	-0.0036	0.0	0.0034	-0.0022	0.0	0.0069	-0.0048	0.0	0.0110	-0.0097

Table C.26: Average importance for each node class for the third layer of the MLP (32-node layer)

C.3 Effects of dropout on Node Importance

C.3.1 MNIST

Figures

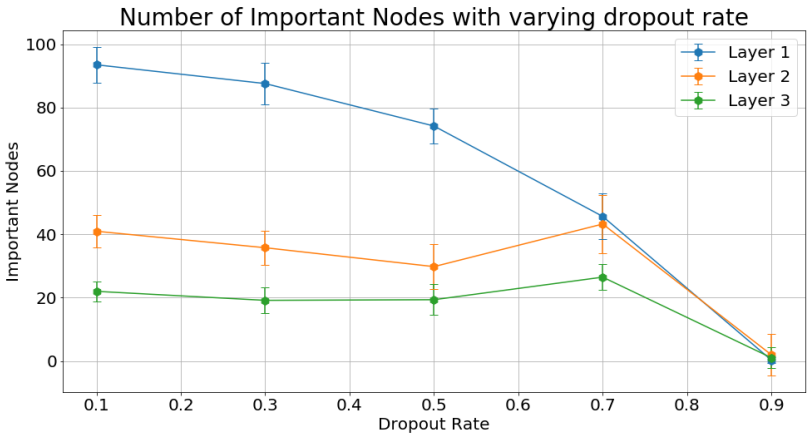

Figure C.6: Number of important nodes per dropout rate

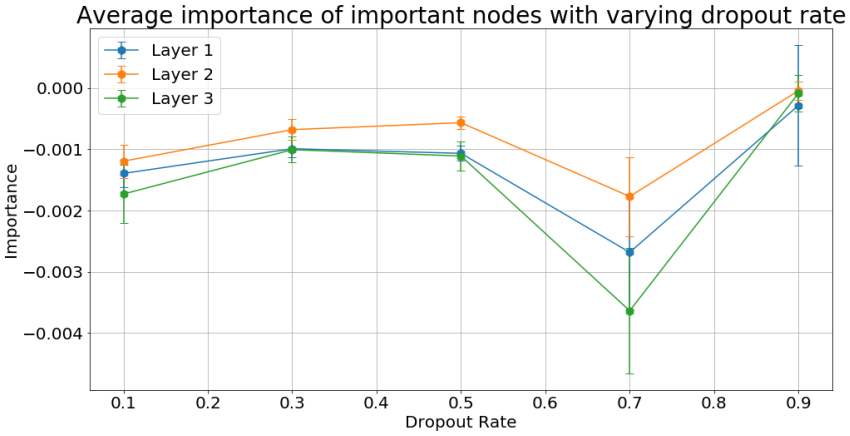

Figure C.7: Average node importance of important nodes per dropout rate

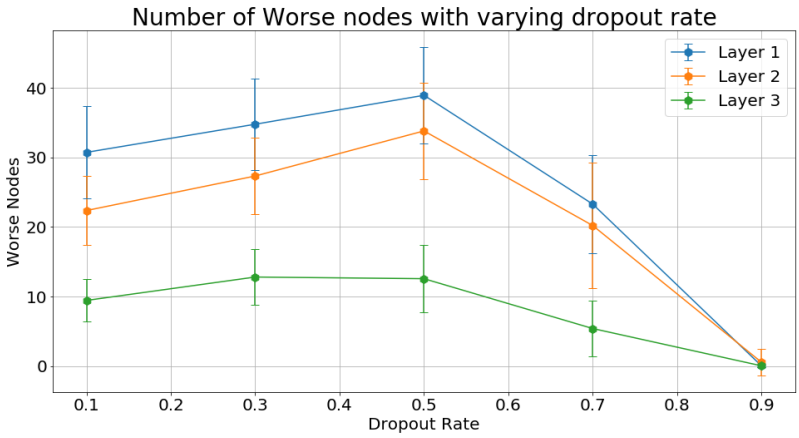

Figure C.8: Number of worse nodes per dropout rate

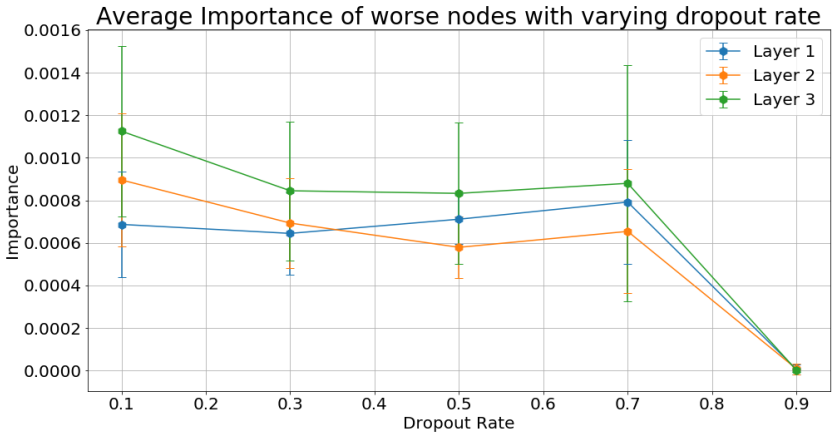

Figure C.9: Average node importance of worse nodes per dropout rate

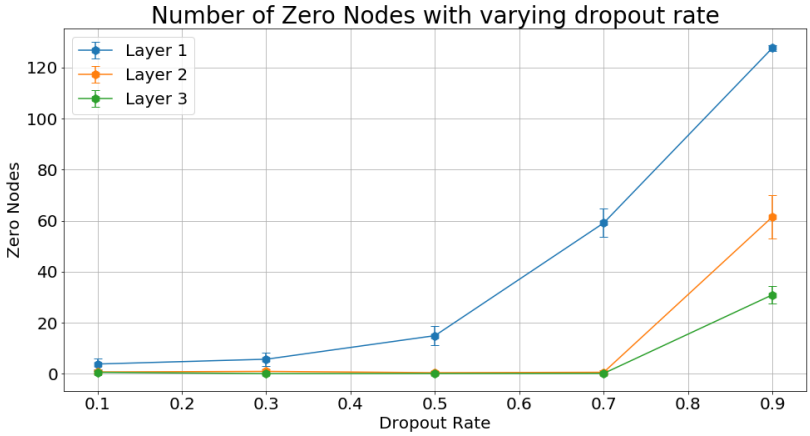

Figure C.10: Number of zero nodes per dropout rate

Tables

	0.1	0.3	0.5	0.7	0.9
mean	0.9743	0.9696	0.9564	0.7289	0.1135
std	0.0017	0.0014	0.0018	0.0575	0.0000
min	0.9700	0.9664	0.9538	0.6527	0.1135
25%	0.9739	0.9688	0.9552	0.6785	0.1135
50%	0.9746	0.9697	0.9561	0.7092	0.1135
75%	0.9755	0.9705	0.9576	0.7580	0.1135
max	0.9766	0.9718	0.9600	0.8546	0.1135

Table C.27: Statistics of accuracy of MLPs trained on MNIST

	0.1	0.3	0.5	0.7	0.9
mean	0.0887	0.1108	0.1697	0.7803	2.3001
std	0.0049	0.0055	0.0063	0.0866	0.0037
min	0.0802	0.0990	0.1564	0.6178	2.2858
25%	0.0861	0.1094	0.1667	0.7299	2.3010
50%	0.0882	0.1114	0.1692	0.7903	2.3011
75%	0.0910	0.1131	0.1744	0.8502	2.3011
max	0.1019	0.1254	0.1812	0.8890	2.3012

Table C.28: Statistics of loss of MLPs trained on MNIST

	0.1			0.3			0.5			0.7			0.9		
	Zero Nodes	Worse Nodes	Important Nodes	Zero Nodes	Worse Nodes	Important Nodes	Zero Nodes	Worse Nodes	Important Nodes	Zero Nodes	Worse Nodes	Important Nodes	Zero Nodes	Worse Nodes	Important Nodes
mean	3.8	30.72	93.48	5.68	34.76	87.56	14.92	38.92	74.16	59.08	23.28	45.64	127.68	0.04	0.28
std	2.0	6.67	5.64	2.53	6.54	6.50	3.65	6.95	5.60	5.53	7.00	7.23	1.11	0.20	0.98
min	1.0	15.00	82.00	0.00	25.00	68.00	10.00	25.00	60.00	49.00	8.00	35.00	124.00	0.00	0.00
25%	3.0	26.00	91.00	5.00	31.00	84.00	12.00	35.00	72.00	55.00	19.00	39.00	128.00	0.00	0.00
50%	3.0	28.00	95.00	5.00	34.00	87.00	14.00	39.00	74.00	60.00	24.00	46.00	128.00	0.00	0.00
75%	5.0	35.00	97.00	7.00	37.00	92.00	18.00	43.00	77.00	63.00	27.00	51.00	128.00	0.00	0.00
max	9.0	45.00	104.00	11.00	57.00	99.00	21.00	54.00	84.00	69.00	39.00	62.00	128.00	1.00	4.00

Table C.29: Number of nodes in each class of nodes for the first layer of the MLP (128-node layer)

	0.1			0.3			0.5			0.7			0.9		
	Zero Nodes	Worse Nodes	Important Nodes	Zero Nodes	Worse Nodes	Important Nodes	Zero Nodes	Worse Nodes	Important Nodes	Zero Nodes	Worse Nodes	Important Nodes	Zero Nodes	Worse Nodes	Important Nodes
mean	0.68	22.36	40.96	0.92	27.32	35.76	0.40	33.80	29.80	0.56	20.2	43.24	61.48	0.56	1.96
std	0.75	5.00	5.13	0.86	5.46	5.46	0.71	6.93	7.01	0.77	9.0	9.19	8.47	1.94	6.57
min	0.00	13.00	29.00	0.00	16.00	27.00	0.00	18.00	20.00	0.00	4.0	18.00	28.00	0.00	0.00
25%	0.00	19.00	38.00	0.00	24.00	32.00	0.00	27.00	24.00	0.00	15.0	38.00	64.00	0.00	0.00
50%	1.00	21.00	42.00	1.00	27.00	35.00	0.00	36.00	27.00	0.00	19.0	44.00	64.00	0.00	0.00
75%	1.00	26.00	45.00	1.00	32.00	39.00	1.00	40.00	36.00	1.00	26.0	49.00	64.00	0.00	0.00
max	2.00	33.00	51.00	3.00	37.00	48.00	2.00	44.00	46.00	2.00	44.0	60.00	64.00	9.00	27.00

Table C.30: Number of nodes in each class of nodes for the second layer of the MLP (64-node layer)

	0.1			0.3			0.5			0.7			0.9		
	Zero Nodes	Worse Nodes	Important Nodes	Zero Nodes	Worse Nodes	Important Nodes	Zero Nodes	Worse Nodes	Important Nodes	Zero Nodes	Worse Nodes	Important Nodes	Zero Nodes	Worse Nodes	Important Nodes
mean	0.56	9.44	22.0	0.04	12.80	19.16	0.08	12.56	19.36	0.12	5.4	26.48	30.92	0.04	1.04
std	0.58	3.07	3.2	0.20	4.04	4.02	0.28	4.83	4.84	0.44	4.0	4.02	3.32	0.20	3.32
min	0.00	3.00	17.0	0.00	3.00	11.00	0.00	5.00	10.00	0.00	0.0	17.00	19.00	0.00	0.00
25%	0.00	7.00	20.0	0.00	11.00	17.00	0.00	8.00	16.00	0.00	2.0	24.00	32.00	0.00	0.00
50%	1.00	10.00	21.0	0.00	13.00	19.00	0.00	13.00	19.00	0.00	5.0	27.00	32.00	0.00	0.00
75%	1.00	11.00	25.0	0.00	15.00	21.00	0.00	16.00	23.00	0.00	7.0	30.00	32.00	0.00	0.00
max	2.00	14.00	29.0	1.00	21.00	29.00	1.00	22.00	27.00	2.00	15.0	32.00	32.00	1.00	13.00

Table C.31: Number of nodes in each class of nodes for the third layer of the MLP (32-node layer)

	0.1			0.3			0.5			0.7			0.9		
	Zero Nodes	Worse Nodes	Important Nodes	Zero Nodes	Worse Nodes	Important Nodes	Zero Nodes	Worse Nodes	Important Nodes	Zero Nodes	Worse Nodes	Important Nodes	Zero Nodes	Worse Nodes	Important Nodes
mean	-0.0	0.0007	-0.0014	0.0	0.0006	-0.0010	-0.0	0.0007	-0.0011	-0.0	0.0008	-0.0027	0.0	0.0000	-0.0003
std	0.0	0.0002	0.0002	0.0	0.0002	0.0001	0.0	0.0001	0.0001	0.0	0.0003	0.0010	0.0	0.0000	0.0010
min	-0.0	0.0003	-0.0022	-0.0	0.0003	-0.0014	-0.0	0.0005	-0.0013	-0.0	0.0003	-0.0044	-0.0	0.0000	-0.0037
25%	-0.0	0.0005	-0.0015	-0.0	0.0005	-0.0010	-0.0	0.0006	-0.0012	-0.0	0.0006	-0.0032	-0.0	0.0000	0.0000
50%	0.0	0.0006	-0.0013	0.0	0.0007	-0.0010	-0.0	0.0007	-0.0010	-0.0	0.0008	-0.0027	0.0	0.0000	0.0000
75%	0.0	0.0009	-0.0012	0.0	0.0008	-0.0009	0.0	0.0008	-0.0010	0.0	0.0010	-0.0019	0.0	0.0000	0.0000
max	0.0	0.0013	-0.0011	0.0	0.0010	-0.0008	0.0	0.0010	-0.0008	0.0	0.0015	-0.0013	0.0	0.0001	0.0000

Table C.32: Average importance for each node class for the first layer of the MLP (128-node layer)

	0.1			0.3			0.5			0.7			0.9		
	Zero Nodes	Worse Nodes	Important Nodes	Zero Nodes	Worse Nodes	Important Nodes	Zero Nodes	Worse Nodes	Important Nodes	Zero Nodes	Worse Nodes	Important Nodes	Zero Nodes	Worse Nodes	Important Nodes
mean	-0.0	0.0007	-0.0014	0.0	0.0006	-0.0010	-0.0	0.0007	-0.0011	-0.0	0.0008	-0.0027	0.0	0.0000	-0.0003
std	0.0	0.0002	0.0002	0.0	0.0002	0.0001	0.0	0.0001	0.0001	0.0	0.0003	0.0010	0.0	0.0000	0.0010
min	-0.0	0.0003	-0.0022	-0.0	0.0003	-0.0014	-0.0	0.0005	-0.0013	-0.0	0.0003	-0.0044	-0.0	0.0000	-0.0037
25%	-0.0	0.0005	-0.0015	-0.0	0.0005	-0.0010	-0.0	0.0006	-0.0012	-0.0	0.0006	-0.0032	-0.0	0.0000	0.0000
50%	0.0	0.0006	-0.0013	0.0	0.0007	-0.0010	-0.0	0.0007	-0.0010	-0.0	0.0008	-0.0027	0.0	0.0000	0.0000
75%	0.0	0.0009	-0.0012	0.0	0.0008	-0.0009	0.0	0.0008	-0.0010	0.0	0.0010	-0.0019	0.0	0.0000	0.0000
max	0.0	0.0013	-0.0011	0.0	0.0010	-0.0008	0.0	0.0010	-0.0008	0.0	0.0015	-0.0013	0.0	0.0001	0.0000

Table C.33: Average importance for each node class for the second layer of the MLP (64-node layer)

	0.1			0.3			0.5			0.7			0.9		
	Zero Nodes	Worse Nodes	Important Nodes	Zero Nodes	Worse Nodes	Important Nodes	Zero Nodes	Worse Nodes	Important Nodes	Zero Nodes	Worse Nodes	Important Nodes	Zero Nodes	Worse Nodes	Important Nodes
mean	-0.0	0.0007	-0.0014	0.0	0.0006	-0.0010	-0.0	0.0007	-0.0011	-0.0	0.0008	-0.0027	0.0	0.0000	-0.0003
std	0.0	0.0002	0.0002	0.0	0.0002	0.0001	0.0	0.0001	0.0001	0.0	0.0003	0.0010	0.0	0.0000	0.0010
min	-0.0	0.0003	-0.0022	-0.0	0.0003	-0.0014	-0.0	0.0005	-0.0013	-0.0	0.0003	-0.0044	-0.0	0.0000	-0.0037
25%	-0.0	0.0005	-0.0015	-0.0	0.0005	-0.0010	-0.0	0.0006	-0.0012	-0.0	0.0006	-0.0032	-0.0	0.0000	0.0000
50%	0.0	0.0006	-0.0013	0.0	0.0007	-0.0010	-0.0	0.0007	-0.0010	-0.0	0.0008	-0.0027	0.0	0.0000	0.0000
75%	0.0	0.0009	-0.0012	0.0	0.0008	-0.0009	0.0	0.0008	-0.0010	0.0	0.0010	-0.0019	0.0	0.0000	0.0000
max	0.0	0.0013	-0.0011	0.0	0.0010	-0.0008	0.0	0.0010	-0.0008	0.0	0.0015	-0.0013	0.0	0.0001	0.0000

Table C.34: Average importance for each node class for the third layer of the MLP (32-node layer)

C.3.2 Fashion MNIST

Tables

	0.1			0.3			0.5			0.7			0.9		
	Zero Nodes	Worse Nodes	Important Nodes	Zero Nodes	Worse Nodes	Important Nodes	Zero Nodes	Worse Nodes	Important Nodes	Zero Nodes	Worse Nodes	Important Nodes	Zero Nodes	Worse Nodes	Important Nodes
mean	20.44	23.80	83.76	30.08	25.80	72.12	47.08	21.56	59.36	88.40	3.08	36.52	123.96	0.32	3.72
std	3.08	6.01	6.69	3.38	5.01	5.29	5.53	7.29	8.08	9.59	3.12	9.19	4.89	0.85	4.37
min	15.00	13.00	71.00	21.00	16.00	65.00	34.00	9.00	45.00	77.00	0.00	14.00	112.00	0.00	0.00
25%	18.00	19.00	80.00	28.00	23.00	67.00	43.00	16.00	52.00	83.00	1.00	33.00	122.00	0.00	0.00
50%	21.00	23.00	83.00	31.00	27.00	72.00	47.00	20.00	61.00	85.00	2.00	39.00	126.00	0.00	2.00
75%	22.00	29.00	89.00	32.00	29.00	76.00	51.00	27.00	65.00	90.00	3.00	43.00	128.00	0.00	6.00
max	28.00	35.00	94.00	34.00	35.00	81.00	57.00	36.00	76.00	111.00	11.00	45.00	128.00	4.00	15.00

Table C.35: Number of nodes in each class of nodes for the first layer of the MLP (128-node layer)

	0.1			0.3			0.5			0.7			0.9		
	Zero Nodes	Worse Nodes	Important Nodes	Zero Nodes	Worse Nodes	Important Nodes	Zero Nodes	Worse Nodes	Important Nodes	Zero Nodes	Worse Nodes	Important Nodes	Zero Nodes	Worse Nodes	Important Nodes
mean	0.76	18.6	44.64	0.68	21.52	41.80	0.28	19.08	44.64	1.08	2.68	60.24	44.08	4.60	15.32
std	0.88	5.2	5.37	0.85	5.45	5.39	0.54	6.26	6.18	1.29	5.66	6.44	20.45	4.96	16.28
min	0.00	7.0	36.00	0.00	13.00	33.00	0.00	7.00	32.00	0.00	0.00	40.00	8.00	0.00	0.00
25%	0.00	14.0	41.00	0.00	16.00	38.00	0.00	15.00	40.00	0.00	0.00	61.00	21.00	0.00	0.00
50%	1.00	18.0	45.00	0.00	22.00	41.00	0.00	19.00	45.00	1.00	0.00	63.00	54.00	4.00	8.00
75%	1.00	23.0	48.00	1.00	26.00	46.00	0.00	23.00	48.00	2.00	2.00	64.00	64.00	7.00	29.00
max	3.00	27.0	57.00	3.00	31.00	51.00	2.00	32.00	56.00	4.00	20.00	64.00	64.00	19.00	46.00

Table C.36: Number of nodes in each class of nodes for the second layer of the MLP (64-node layer)

	0.1			0.3			0.5			0.7			0.9		
	Zero Nodes	Worse Nodes	Important Nodes	Zero Nodes	Worse Nodes	Important Nodes	Zero Nodes	Worse Nodes	Important Nodes	Zero Nodes	Worse Nodes	Important Nodes	Zero Nodes	Worse Nodes	Important Nodes
mean	0.36	6.60	25.04	0.16	6.72	25.12	0.16	4.92	26.92	0.0	0.36	31.64	23.8	0.08	8.12
std	0.70	3.21	3.28	0.37	3.32	3.27	0.37	2.72	2.72	0.0	1.22	1.22	8.5	0.40	8.57
min	0.00	0.00	18.00	0.00	2.00	17.00	0.00	0.00	20.00	0.0	0.00	26.00	8.0	0.00	0.00
25%	0.00	4.00	23.00	0.00	5.00	23.00	0.00	3.00	26.00	0.0	0.00	32.00	16.0	0.00	0.00
50%	0.00	6.00	25.00	0.00	6.00	26.00	0.00	5.00	27.00	0.0	0.00	32.00	28.0	0.00	4.00
75%	1.00	9.00	28.00	0.00	9.00	27.00	0.00	6.00	29.00	0.0	0.00	32.00	32.0	0.00	16.00
max	3.00	14.00	32.00	1.00	15.00	30.00	1.00	12.00	32.00	0.0	6.00	32.00	32.0	2.00	24.00

Table C.37: Number of nodes in each class of nodes for the third layer of the MLP (32-node layer)

	0.1			0.3			0.5			0.7			0.9		
	Zero Nodes	Worse Nodes	Important Nodes	Zero Nodes	Worse Nodes	Important Nodes	Zero Nodes	Worse Nodes	Important Nodes	Zero Nodes	Worse Nodes	Important Nodes	Zero Nodes	Worse Nodes	Important Nodes
mean	-0.0	0.0012	-0.0019	-0.0	0.0009	-0.0016	0.0	0.0008	-0.0018	-0.0	0.0007	-0.0051	-0.0	0.0000	-0.0006
std	0.0	0.0009	0.0003	0.0	0.0004	0.0001	0.0	0.0004	0.0003	0.0	0.0020	0.0039	0.0	0.0001	0.0009
min	-0.0	0.0003	-0.0025	-0.0	0.0004	-0.0019	-0.0	0.0002	-0.0024	-0.0	0.0000	-0.0164	-0.0	0.0000	-0.0025
25%	-0.0	0.0007	-0.0019	-0.0	0.0006	-0.0017	-0.0	0.0006	-0.0020	-0.0	0.0001	-0.0059	-0.0	0.0000	-0.0012
50%	-0.0	0.0010	-0.0018	-0.0	0.0009	-0.0016	0.0	0.0008	-0.0018	-0.0	0.0002	-0.0044	-0.0	0.0000	-0.0001
75%	0.0	0.0013	-0.0017	0.0	0.0009	-0.0015	0.0	0.0010	-0.0016	0.0	0.0005	-0.0020	0.0	0.0000	0.0000
max	0.0	0.0044	-0.0014	0.0	0.0024	-0.0014	0.0	0.0020	-0.0013	0.0	0.0098	-0.0012	0.0	0.0004	0.0000

Table C.38: Average importance for each node class for the first layer of the MLP (128-node layer)

	0.1			0.3			0.5			0.7			0.9		
	Zero Nodes	Worse Nodes	Important Nodes	Zero Nodes	Worse Nodes	Important Nodes	Zero Nodes	Worse Nodes	Important Nodes	Zero Nodes	Worse Nodes	Important Nodes	Zero Nodes	Worse Nodes	Important Nodes
mean	-0.0	0.0012	-0.0019	-0.0	0.0009	-0.0016	0.0	0.0008	-0.0018	-0.0	0.0007	-0.0051	-0.0	0.0000	-0.0006
std	0.0	0.0009	0.0003	0.0	0.0004	0.0001	0.0	0.0004	0.0003	0.0	0.0020	0.0039	0.0	0.0001	0.0009
min	-0.0	0.0003	-0.0025	-0.0	0.0004	-0.0019	-0.0	0.0002	-0.0024	-0.0	0.0000	-0.0164	-0.0	0.0000	-0.0025
25%	-0.0	0.0007	-0.0019	-0.0	0.0006	-0.0017	-0.0	0.0006	-0.0020	-0.0	0.0001	-0.0059	-0.0	0.0000	-0.0012
50%	-0.0	0.0010	-0.0018	-0.0	0.0009	-0.0016	0.0	0.0008	-0.0018	-0.0	0.0002	-0.0044	-0.0	0.0000	-0.0001
75%	0.0	0.0013	-0.0017	0.0	0.0009	-0.0015	0.0	0.0010	-0.0016	0.0	0.0005	-0.0020	0.0	0.0000	0.0000
max	0.0	0.0044	-0.0014	0.0	0.0024	-0.0014	0.0	0.0020	-0.0013	0.0	0.0098	-0.0012	0.0	0.0004	0.0000

Table C.39: Average importance for each node class for the second layer of the MLP (64-node layer)

	0.1			0.3			0.5			0.7			0.9		
	Zero Nodes	Worse Nodes	Important Nodes	Zero Nodes	Worse Nodes	Important Nodes	Zero Nodes	Worse Nodes	Important Nodes	Zero Nodes	Worse Nodes	Important Nodes	Zero Nodes	Worse Nodes	Important Nodes
mean	-0.0	0.0012	-0.0019	-0.0	0.0009	-0.0016	0.0	0.0008	-0.0018	-0.0	0.0007	-0.0051	-0.0	0.0000	-0.0006
std	0.0	0.0009	0.0003	0.0	0.0004	0.0001	0.0	0.0004	0.0003	0.0	0.0020	0.0039	0.0	0.0001	0.0009
min	-0.0	0.0003	-0.0025	-0.0	0.0004	-0.0019	-0.0	0.0002	-0.0024	-0.0	0.0000	-0.0164	-0.0	0.0000	-0.0025
25%	-0.0	0.0007	-0.0019	-0.0	0.0006	-0.0017	-0.0	0.0006	-0.0020	-0.0	0.0001	-0.0059	-0.0	0.0000	-0.0012
50%	-0.0	0.0010	-0.0018	-0.0	0.0009	-0.0016	0.0	0.0008	-0.0018	-0.0	0.0002	-0.0044	-0.0	0.0000	-0.0001
75%	0.0	0.0013	-0.0017	0.0	0.0009	-0.0015	0.0	0.0010	-0.0016	0.0	0.0005	-0.0020	0.0	0.0000	0.0000
max	0.0	0.0044	-0.0014	0.0	0.0024	-0.0014	0.0	0.0020	-0.0013	0.0	0.0098	-0.0012	0.0	0.0004	0.0000

Table C.40: Average importance for each node class for the third layer of the MLP (32-node layer)

C.4 Pre-calculated Pruning

C.4.1 MNIST

Figures- Training set

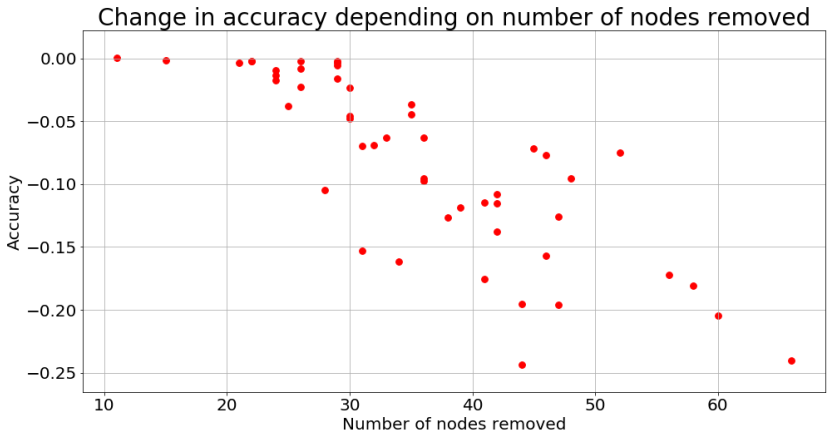

Figure C.11: Change in accuracy after pruning based on pre-calculated node importance

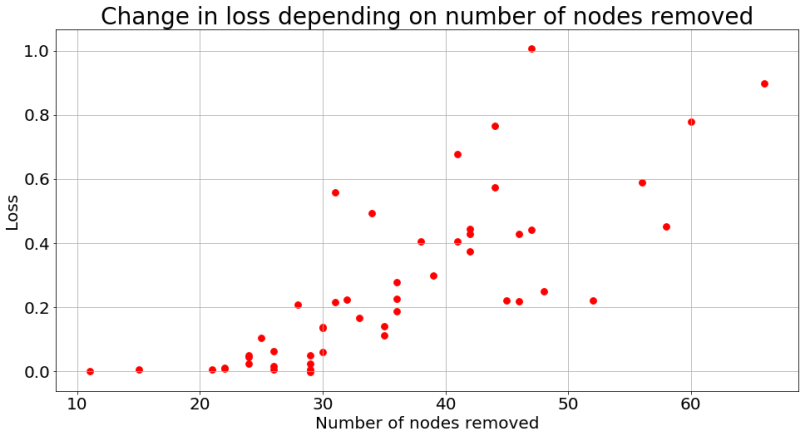

Figure C.12: Change in loss after pruning based on pre-calculated node importance

Figures- Validation set

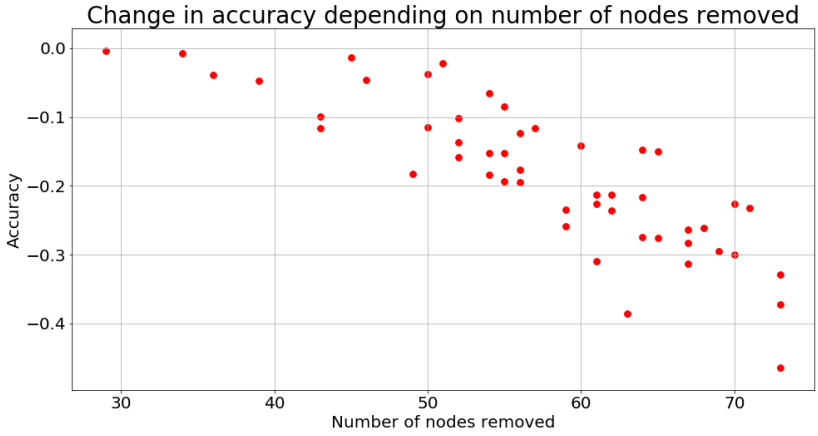

Figure C.13: Change in accuracy after pruning based on pre-calculated node importance

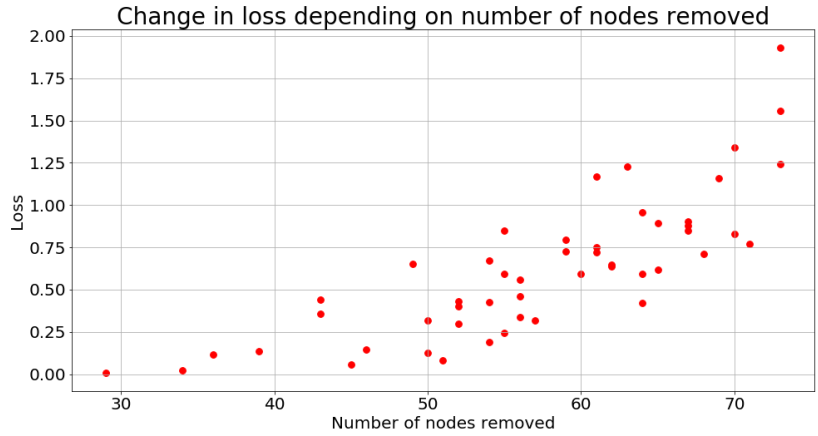

Figure C.14: Change in loss after pruning based on pre-calculated node importance

Tables - Training set

	Layer 3	Layer 2	Layer 1
mean	4.42	10.50	20.74
std	1.73	4.01	7.62
min	2.00	3.00	5.00
25%	3.00	7.25	16.00
50%	4.00	11.00	20.00
75%	5.00	13.00	24.75
max	9.00	19.00	39.00

Table C.41: Statistics of number of nodes pruned from MLPs trained on MNIST

	Change in Accuracy
mean	-0.0831
std	0.0706
min	-0.2434
25%	-0.1265
50%	-0.0707
75%	-0.0166
max	0.0001

Table C.42: Statistics of change in accuracy of MLPs trained on MNIST

	Change in Loss
mean	0.2689
std	0.2576
min	-0.0014
25%	0.0490
50%	0.2175
75%	0.4286
max	1.0061

Table C.43: Statistics of change loss of MLPs trained on MNIST

Tables - Validation set

	Layer 3	Layer 2	Layer 1
mean	6.52	17.06	33.64
std	1.87	4.02	6.64
min	1.00	4.00	16.00
25%	6.00	15.00	30.25
50%	7.00	17.00	33.50
75%	7.75	20.00	39.75
max	10.00	25.00	44.00

Table C.44: Statistics of number of nodes pruned from MLPs trained on MNIST

	Change in Accuracy
mean	-0.1837
std	0.1064
min	-0.4639
25%	-0.2606
50%	-0.1829
75%	-0.1154
max	-0.0041

Table C.45: Statistics of change in accuracy of MLPs trained on MNIST

	Change in Loss
mean	0.6232
std	0.4126
min	0.0059
25%	0.3227
50%	0.6055
75%	0.8457
max	1.9321

Table C.46: Statistics of change loss of MLPs trained on MNIST

C.4.2 Fashion MNIST

Figures - Training set

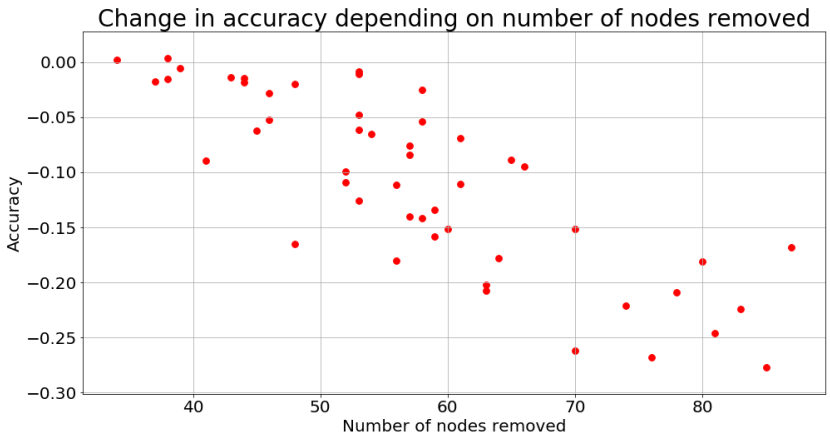

Figure C.15: Change in accuracy after pruning based on pre-calculated node importance

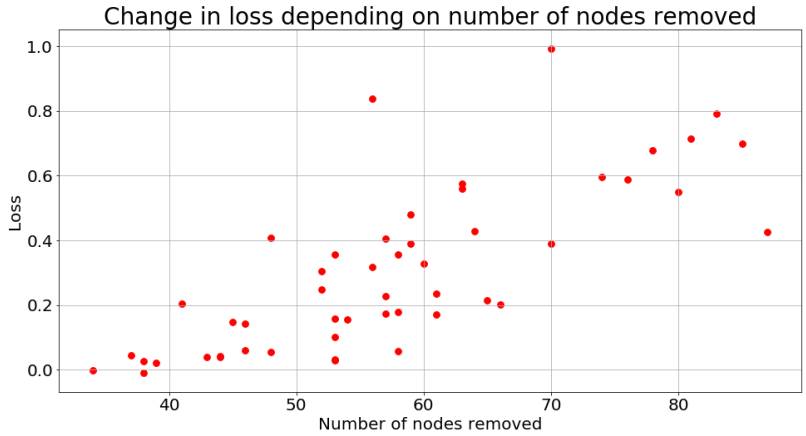

Figure C.16: Change in loss after pruning based on pre-calculated node importance

Figures - Validation set

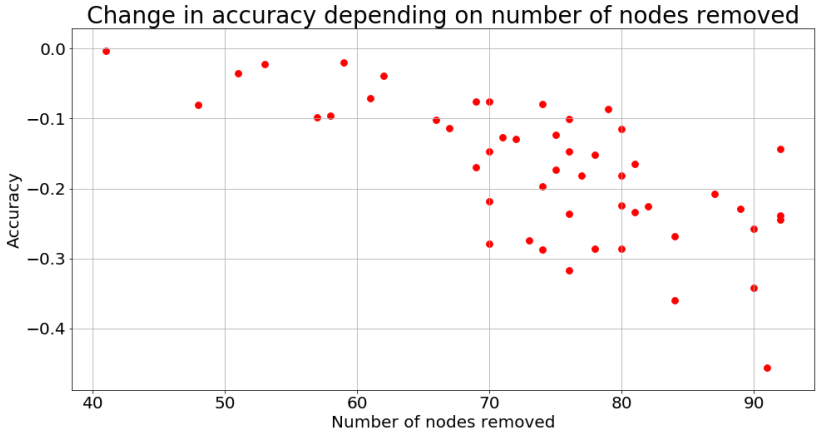

Figure C.17: Change in accuracy after pruning based on pre-calculated node importance

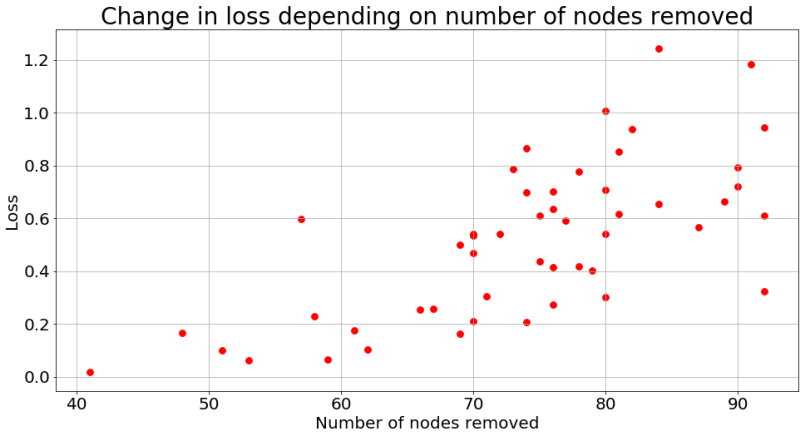

Figure C.18: Change in loss after pruning based on pre-calculated node importance

Tables - Training set

	Layer 3	Layer 2	Layer 1
mean	6.34	15.62	35.56
std	1.98	5.40	7.86
min	2.00	5.00	19.00
25%	5.00	12.00	31.00
50%	6.00	15.00	35.00
75%	7.00	19.00	40.00
max	12.00	29.00	50.00

Table C.47: Statistics of number of nodes pruned from MLPs trained on Fashion MNIST

	Change in Accuracy
mean	-0.1087
std	0.0810
min	-0.2773
25%	-0.1674
50%	-0.0967
75%	-0.0329
max	0.0038

Table C.48: Statistics of change in accuracy of MLPs trained on Fashion MNIST

	Change in Loss
mean	0.3035
std	0.2519
min	-0.0090
25%	0.0695
50%	0.2315
75%	0.4288
max	0.9918

Table C.49: Statistics of change loss of MLPs trained on Fashion MNIST

Tables - Validation set

	Layer 3	Layer 2	Layer 1
mean	7.50	20.40	46.1
std	2.48	4.61	7.0
min	2.00	8.00	27.0
25%	6.00	18.00	44.0
50%	8.00	20.50	47.5
75%	9.00	23.00	50.0
max	13.00	28.00	58.0

Table C.50: Statistics of number of nodes pruned from MLPs trained on Fashion MNIST

	Change in Accuracy
mean	-0.1744
std	0.0983
min	-0.4558
25%	-0.2380
50%	-0.1670
75%	-0.0987
max	-0.0036

Table C.51: Statistics of change in accuracy of MLPs trained on Fashion MNIST

	Change in Loss
mean	0.5159
std	0.2955
min	0.0174
25%	0.2610
50%	0.5405
75%	0.7009
max	1.2444

Table C.52: Statistics of change loss of MLPs trained on Fashion MNIST

C.5 Exhaustive Pruning

C.5.1 MNIST

Figures - Training set

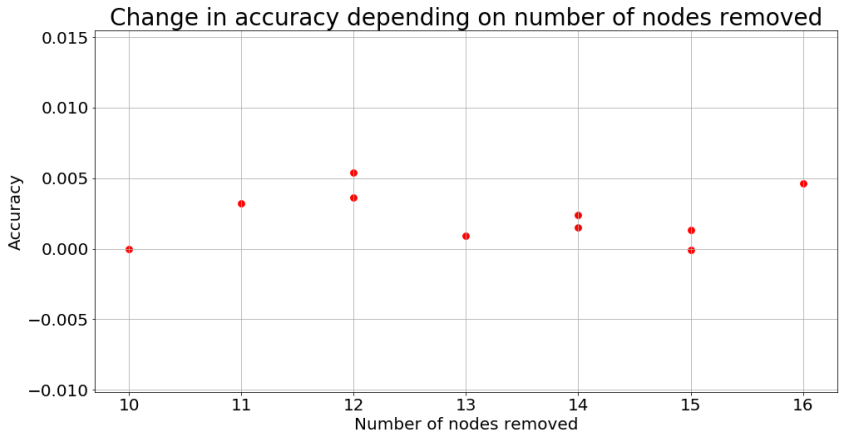

Figure C.19: Change in accuracy after pruning with exhaustive method

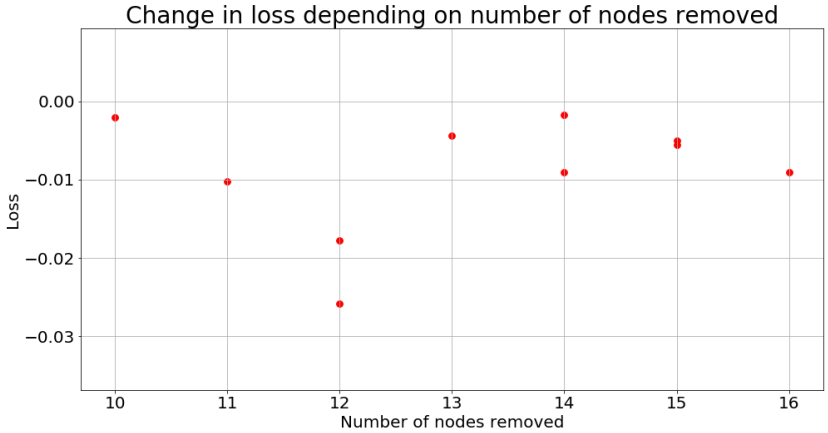

Figure C.20: Change in loss after pruning with exhaustive method

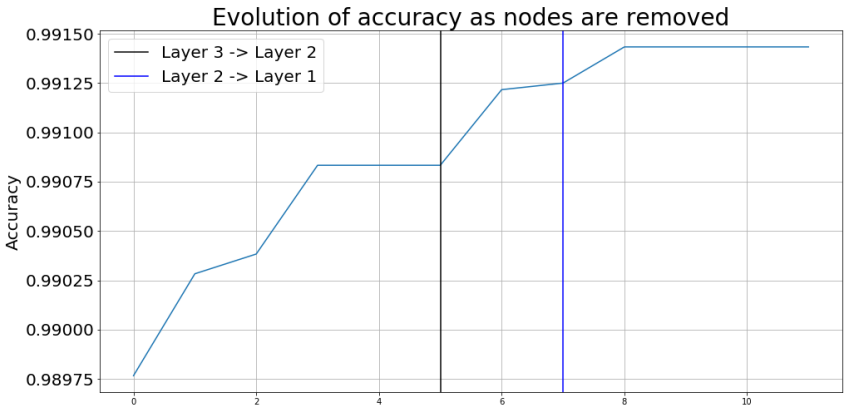

Figure C.21: Evolution of training accuracy as an MLP is pruned

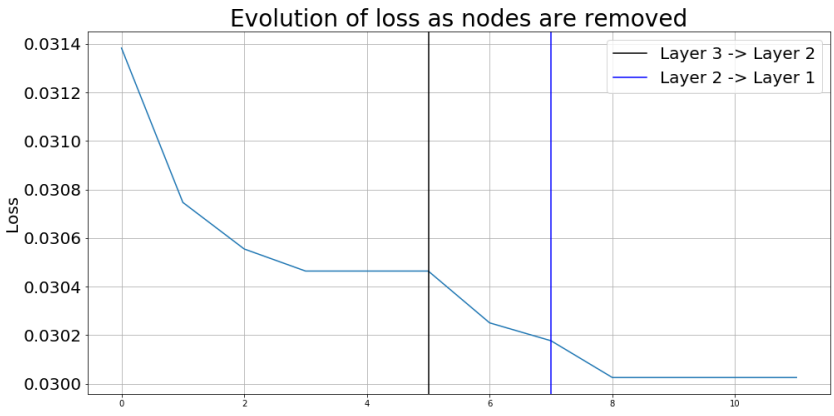

Figure C.22: Evolution of training loss as an MLP is pruned

Figures - Validation set

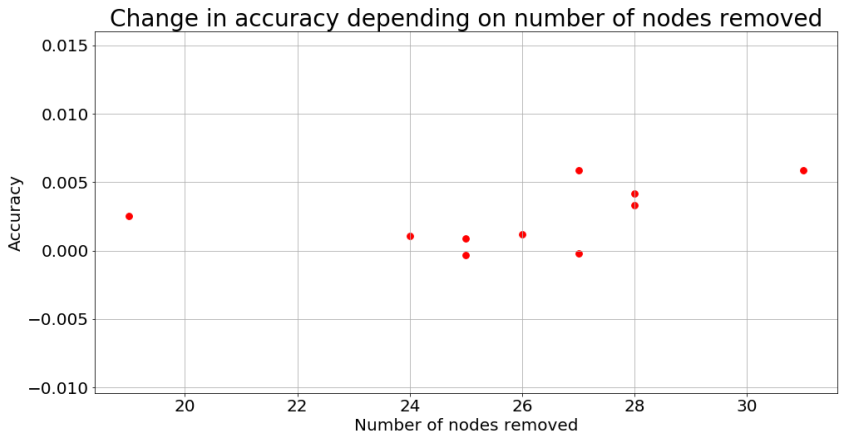

Figure C.23: Change in accuracy after pruning with exhaustive method

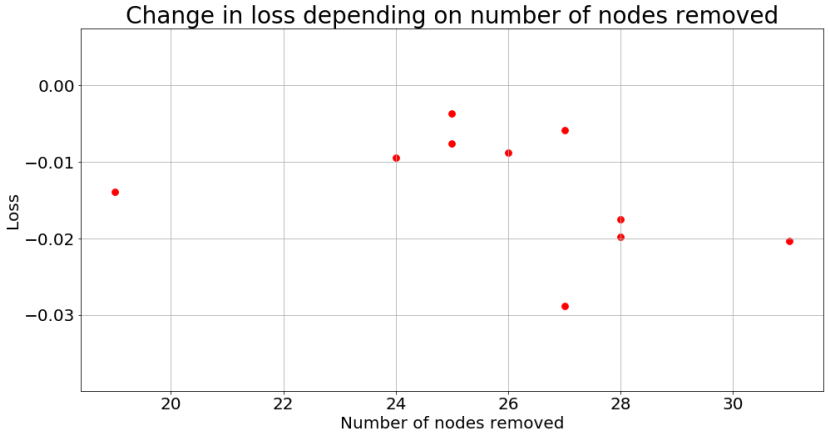

Figure C.24: Change in accuracy after pruning with exhaustive method

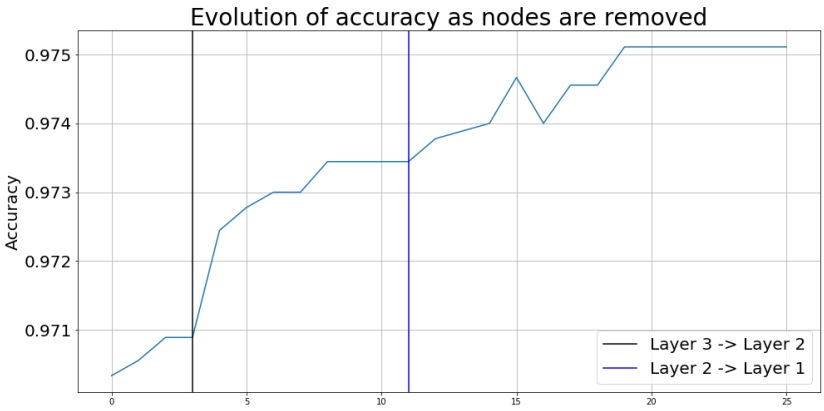

Figure C.25: Evolution of validation accuracy as an MLP is pruned

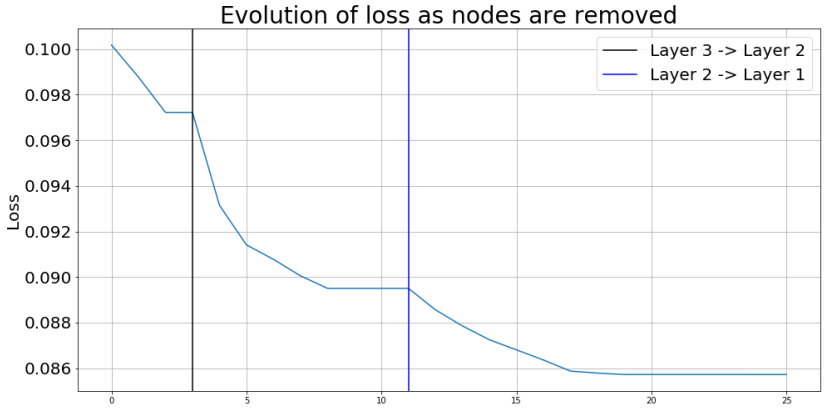

Figure C.26: Evolution of validation loss as an MLP is pruned

Tables - Training set

	Layer 3	Layer 2	Layer 1
mean	3.40	4.00	5.80
std	1.51	1.76	2.97
min	1.00	0.00	2.00
25%	2.00	3.25	3.25
50%	4.00	4.00	5.50
75%	4.75	5.00	8.25
max	5.00	6.00	10.00

Table C.53: Statistics of number of nodes pruned from MLPs trained on MNIST

	Accuracy
mean	0.0023
std	0.0019
min	-0.0001
25%	0.0010
50%	0.0020
75%	0.0035
max	0.0054

Table C.54: Statistics of change in accuracy of MLPs trained on MNIST

	Loss
mean	-0.0091
std	0.0075
min	-0.0259
25%	-0.0099
50%	-0.0073
75%	-0.0046
max	-0.0017

Table C.55: Statistics of change loss of MLPs trained on MNIST

Tables - Validation set

	Layer 3	Layer 2	Layer 1
mean	5.50	8.40	12.10
std	2.17	2.07	3.07
min	3.00	5.00	9.00
25%	3.50	7.50	10.00
50%	5.00	9.00	10.50
75%	7.00	9.75	14.75
max	9.00	11.00	17.00

Table C.56: Statistics of number of nodes pruned from MLPs trained on MNIST

	Accuracy
mean	0.0024
std	0.0023
min	-0.0003
25%	0.0009
50%	0.0018
75%	0.0040
max	0.0059

Table C.57: Statistics of change accuracy of MLPs trained on MNIST

	Loss
mean	-0.0136
std	0.0080
min	-0.0289
25%	-0.0193
50%	-0.0117
75%	-0.0079
max	-0.0036

Table C.58: Statistics of change loss of MLPs trained on MNIST

C.5.2 Fashion MNIST

Tables - Training set

	Accuracy
mean	0.0059
std	0.0023
min	0.0017
25%	0.0049
50%	0.0055
75%	0.0071
max	0.0097

Table C.59: Statistics of change in accuracy of MLPs trained on Fashion MNIST

	Loss
mean	-0.0186
std	0.0068
min	-0.0281
25%	-0.0248
50%	-0.0171
75%	-0.0143
max	-0.0075

Table C.60: Statistics of change loss of MLPs trained on Fashion MNIST

Tables - Validation set

	Accuracy
mean	0.0085
std	0.0030
min	0.0057
25%	0.0063
50%	0.0067
75%	0.0102
max	0.0140

Table C.61: Statistics of change accuracy of MLPs trained on Fashion MNIST

	Loss
mean	-0.0182
std	0.0069
min	-0.0271
25%	-0.0241
50%	-0.0176
75%	-0.0146
max	-0.0071

Table C.62: Statistics of change loss of MLPs trained on Fashion MNIST

C.6 Greedy Pruning

C.6.1 MNIST

Figures - ignore cutoff: $-1e-2$

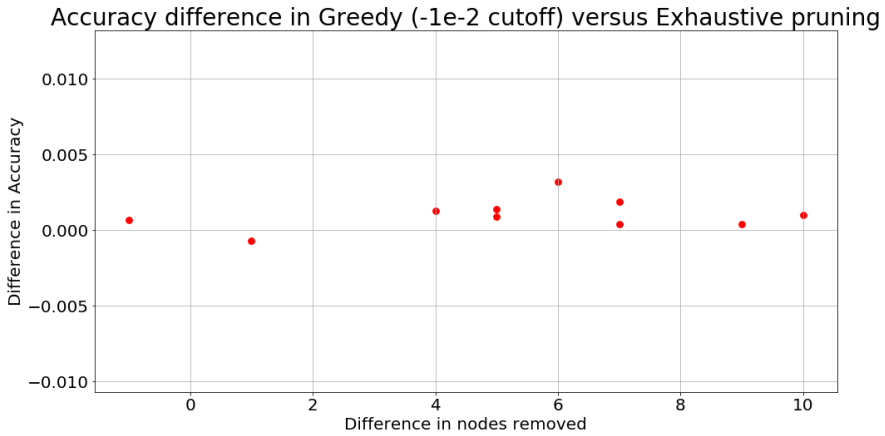

Figure C.27: Change in accuracy after pruning with greedy method and an ignore cutoff of $-1e-2$

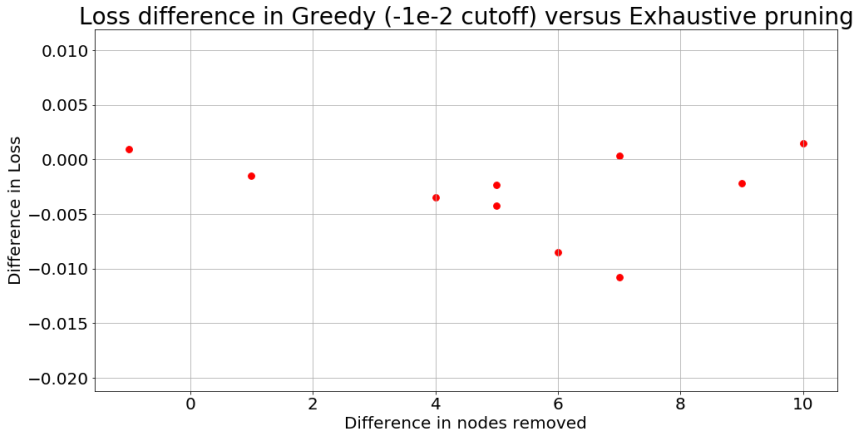

Figure C.28: Change in loss after pruning with greedy method and an ignore cutoff of $-1e-2$

Figures - ignore cutoff: $-1e-3$

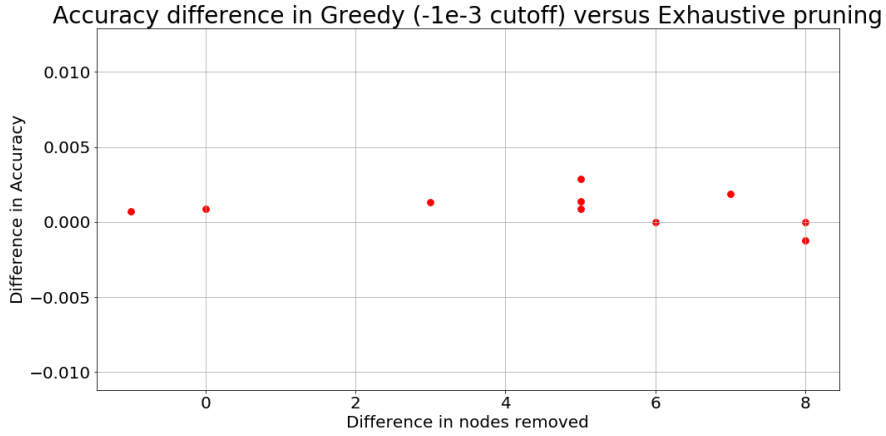

Figure C.29: Change in accuracy after pruning with greedy method and an ignore cutoff of $-1e-3$

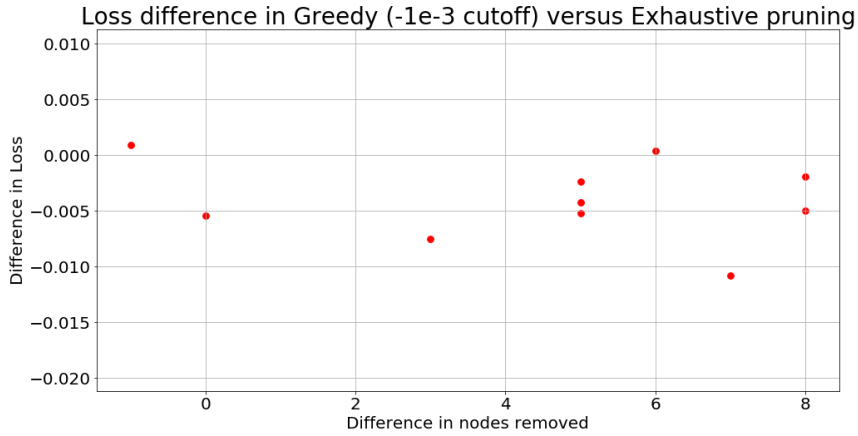

Figure C.30: Change in loss after pruning with greedy method and an ignore cutoff of $-1e-3$

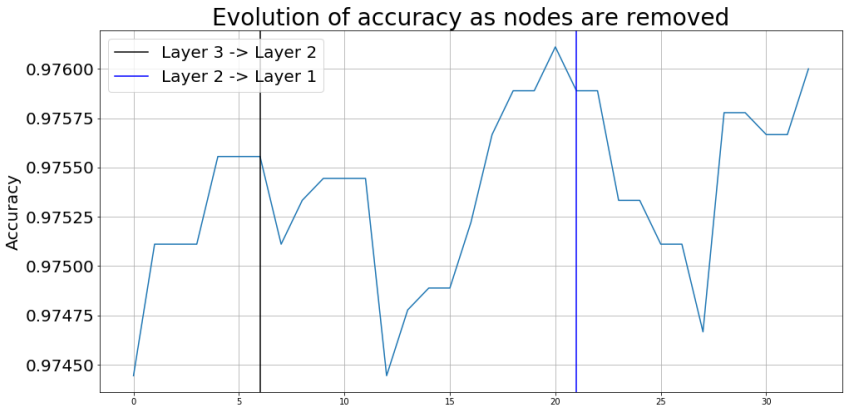

Figure C.31: Evolution of validation accuracy as an MLP is pruned

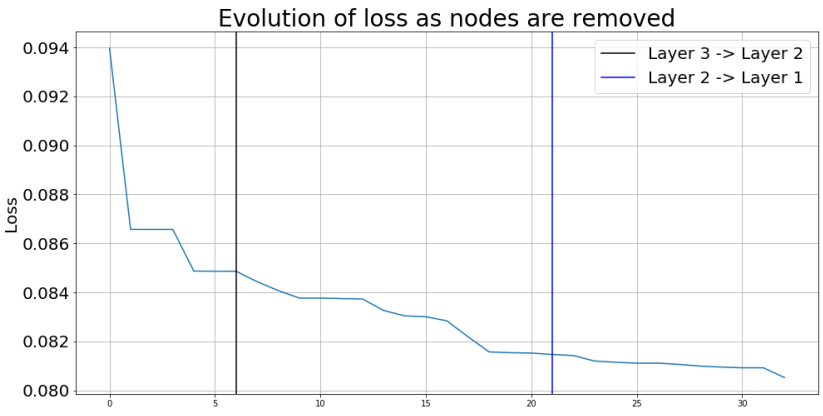

Figure C.32: Evolution of validation loss as an MLP is pruned

Tables - ignore cutoff: $-1e-2$

	Layer 3	Layer 2	Layer 1
mean	0.20	2.80	2.30
std	0.42	1.75	4.57
min	0.00	1.00	-4.00
25%	0.00	1.25	-0.75
50%	0.00	2.50	2.00
75%	0.00	3.75	6.00
max	1.00	6.00	8.00

Table C.63: Statistics of the difference in the number of nodes pruned from MLPs trained on MNIST

Tables - ignore cutoff: $-1e-3$

	Layer 3	Layer 2	Layer 1
mean	0.30	2.40	1.90
std	0.48	1.90	4.46
min	0.00	-1.00	-4.00
25%	0.00	1.25	-2.00
50%	0.00	2.50	2.00
75%	0.75	3.00	5.50
max	1.00	6.00	8.00

Table C.64: Statistics of the difference in the number of nodes pruned from MLPs trained on MNIST

	Exhaustive	Greedy (-1e-2 cutoff)	Greedy (-1e-3 cutoff)
mean	0.0015	0.0004	0.0006
std	0.0015	0.0013	0.0015
min	-0.0008	-0.0022	-0.0022
25%	0.0009	0.0004	-0.0004
50%	0.0015	0.0010	0.0004
75%	0.0024	0.0011	0.0015
max	0.0041	0.0016	0.0028

Table C.65: Statistics of change accuracy of MLPs trained on MNIST

	Exhaustive	Greedy (-1e-2 cutoff)	Greedy (-1e-3 cutoff)
mean	-0.0099	-0.0069	-0.0058
std	0.0051	0.0037	0.0043
min	-0.0174	-0.0117	-0.0118
25%	-0.0135	-0.0094	-0.0095
50%	-0.0102	-0.0078	-0.0056
75%	-0.0086	-0.0042	-0.0032
max	-0.0003	-0.0013	0.0016

Table C.66: Statistics of change loss of MLPs trained on MNIST

	Exhaustive	Greedy (-1e-2 cutoff)	Greedy (-1e-3 cutoff)
mean	150.75	73.60	30.09
std	20.71	16.16	3.07
min	118.61	56.29	23.58
25%	139.51	58.84	28.95
50%	143.73	68.98	29.91
75%	166.01	88.64	31.81
max	189.81	95.32	34.45

Table C.67: Time taken by different pruning algorithms to prune MLPs trained on MNIST

C.6.2 Fashion MNIST

Tables - ignore cutoff: $-1e-2$

	Layer 3	Layer 2	Layer 1
mean	0.40	0.40	1.60
std	1.07	1.43	3.03
min	-1.00	-2.00	-5.00
25%	0.00	0.00	0.25
50%	0.00	0.00	2.00
75%	0.75	1.00	4.00
max	3.00	3.00	5.00

Table C.68: Statistics of the difference in the number of nodes pruned from MLPs trained on Fashion MNIST

Tables - ignore cutoff: $-1e-3$

	Layer 3	Layer 2	Layer 1
mean	0.40	0.40	1.50
std	1.07	1.43	3.14
min	-1.00	-2.00	-5.00
25%	0.00	0.00	0.25
50%	0.00	0.00	2.00
75%	0.75	1.00	4.00
max	3.00	3.00	5.00

Table C.69: Statistics of the difference in the number of nodes pruned from MLPs trained on Fashion MNIST

Tables - general

	Exhaustive	Greedy (-1e-2 cutoff)	Greedy (-1e-3 cutoff)
mean	0.0041	0.0041	0.0041
std	0.0038	0.0025	0.0025
min	0.0000	0.0009	0.0009
25%	0.0011	0.0018	0.0018
50%	0.0029	0.0043	0.0043
75%	0.0075	0.0062	0.0062
max	0.0102	0.0073	0.0073

Table C.70: Statistics of change accuracy of MLPs trained on Fashion MNIST

	Exhaustive	Greedy (-1e-2 cutoff)	Greedy (-1e-3 cutoff)
mean	-0.0124	-0.0102	-0.0101
std	0.0077	0.0059	0.0059
min	-0.0218	-0.0189	-0.0189
25%	-0.0192	-0.0150	-0.0150
50%	-0.0130	-0.0089	-0.0084
75%	-0.0062	-0.0058	-0.0060
max	-0.0007	-0.0015	-0.0015

Table C.71: Statistics of change loss of MLPs trained on Fashion MNIST

C.7 Iterative Weight Initialization

C.7.1 MNIST

Tables

	Unoptimized Weights	Optimized Weights
mean	0.9741	0.9736
std	0.0020	0.0025
min	0.9685	0.9673
25%	0.9734	0.9726
50%	0.9743	0.9741
75%	0.9757	0.9748
max	0.9773	0.9778

Table C.72: Statistics of accuracy of MLPs trained on MNIST

	Unoptimized Weights	Optimized Weights
mean	0.0908	0.0900
std	0.0077	0.0092
min	0.0778	0.0763
25%	0.0869	0.0841
50%	0.0906	0.0894
75%	0.0955	0.0933
max	0.1087	0.1178

Table C.73: Statistics of loss of MLPs trained on MNIST

C.7.2 Fashion MNIST

Tables

	Unoptimized Weights	Optimized Weights
mean	0.8688	0.8695
std	0.0044	0.0064
min	0.8592	0.8505
25%	0.8655	0.8674
50%	0.8693	0.8705
75%	0.8716	0.8743
max	0.8759	0.8782

Table C.74: Statistics of accuracy of MLPs trained on Fashion MNIST

	Unoptimized Weights	Optimized Weights
mean	0.3670	0.3641
std	0.0115	0.0170
min	0.3471	0.3412
25%	0.3588	0.3559
50%	0.3663	0.3618
75%	0.3725	0.3686
max	0.3919	0.4265

Table C.75: Statistics of loss of MLPs trained on Fashion MNIST

Appendix D

CNN - Extra Figures and Tables

D.1 Estimating Node Importance

D.1.1 MNIST

Tables- Training set

	Zero Nodes	Worse Nodes	Important Nodes
mean	25.70	9.30	29.00
std	6.11	5.31	4.50
min	19.00	2.00	20.00
25%	21.50	6.00	27.25
50%	24.50	8.00	29.50
75%	29.00	12.25	32.00
max	38.00	18.00	35.00

Table D.1: Number of nodes in each class of nodes for the first layer of the CNN (32-filter layer)

	Zero Nodes	Worse Nodes	Important Nodes
mean	73.60	62.30	120.10
std	13.57	21.06	19.89
min	51.00	27.00	90.00
25%	66.25	48.00	107.50
50%	71.50	62.50	118.50
75%	81.50	75.75	135.00
max	98.00	96.00	148.00

Table D.2: Number of nodes in each class of nodes for the second layer of the CNN (64-filter layer)

	Zero Nodes	Worse Nodes	Important Nodes
mean	14.20	31.40	82.40
std	5.05	11.40	9.20
min	9.00	11.00	66.00
25%	10.50	29.25	76.75
50%	13.00	33.00	83.00
75%	15.75	36.75	87.50
max	26.00	50.00	98.00

Table D.3: Number of nodes in each class of nodes for the third layer of the CNN (128-filter layer)

	Zero Nodes	Worse Nodes	Important Nodes
mean	3.00	16.60	44.40
std	1.63	6.67	6.29
min	0.00	8.00	35.00
25%	2.00	12.25	38.50
50%	3.00	16.00	46.00
75%	4.00	22.00	48.00
max	6.00	27.00	54.00

Table D.4: Number of nodes in each class of nodes for the fourth layer of the CNN (256-filter layer)

	Zero Nodes	Worse Nodes	Important Nodes
mean	4.8	6.20	21.00
std	2.3	4.49	4.52
min	1.0	1.00	13.00
25%	4.0	3.25	18.00
50%	5.0	5.00	21.50
75%	5.0	8.25	23.75
max	10.0	15.00	28.00

Table D.5: Number of nodes in each class of nodes for the fifth layer of the CNN (32-node layer)

Tables- Validation set

	Zero Nodes	Worse Nodes	Important Nodes
mean	2.50	14.80	14.70
std	1.43	3.12	3.77
min	0.00	10.00	9.00
25%	2.00	14.00	12.50
50%	2.50	14.50	14.50
75%	3.00	17.50	15.75
max	5.00	19.00	22.00

Table D.6: Number of nodes in each class of nodes for the first layer of the CNN (32-filter layer)

	Zero Nodes	Worse Nodes	Important Nodes
mean	1.70	29.20	33.10
std	1.16	3.61	4.07
min	0.00	23.00	28.00
25%	1.00	26.50	30.25
50%	2.00	30.00	32.00
75%	2.75	32.00	36.00
max	3.00	34.00	41.00

Table D.7: Number of nodes in each class of nodes for the second layer of the CNN (64-filter layer)

	Zero Nodes	Worse Nodes	Important Nodes
mean	10.90	54.50	62.60
std	4.58	8.63	8.71
min	2.00	40.00	47.00
25%	9.00	50.25	57.25
50%	11.00	54.00	66.00
75%	13.50	58.00	68.75
max	18.00	72.00	71.00

Table D.8: Number of nodes in each class of nodes for the third layer of the CNN (128-filter layer)

	Zero Nodes	Worse Nodes	Important Nodes
mean	55.80	120.70	79.50
std	8.99	13.83	16.85
min	38.00	103.00	50.00
25%	50.50	111.50	70.50
50%	57.50	117.50	83.00
75%	59.50	128.00	89.75
max	68.00	148.00	105.00

Table D.9: Number of nodes in each class of nodes for the fourth layer of the CNN (256-filter layer)

	Zero Nodes	Worse Nodes	Important Nodes
mean	25.00	16.70	22.30
std	3.97	4.97	4.08
min	20.00	11.00	13.00
25%	22.25	14.00	21.00
50%	24.50	15.50	23.00
75%	28.25	18.75	24.00
max	31.00	28.00	28.00

Table D.10: Number of nodes in each class of nodes for the fifth layer of the CNN (32-node layer)

D.1.2 Fashion MNIST

Tables- Training set

	Zero Nodes	Worse Nodes	Important Nodes
mean	25.70	6.30	32.00
std	4.79	2.87	4.06
min	19.00	2.00	26.00
25%	22.25	4.00	28.75
50%	25.50	6.50	32.50
75%	29.50	8.00	33.75
max	32.00	11.00	40.00

Table D.11: Number of nodes in each class of nodes for the first layer of the CNN (32-filter layer)

	Zero Nodes	Worse Nodes	Important Nodes
mean	49.10	48.30	158.60
std	3.45	20.69	20.43
min	44.00	12.00	124.00
25%	46.75	32.50	151.75
50%	50.00	50.50	155.50
75%	50.75	60.25	173.75
max	54.00	81.00	190.00

Table D.12: Number of nodes in each class of nodes for the second layer of the CNN (64-filter layer)

	Zero Nodes	Worse Nodes	Important Nodes
mean	11.30	23.20	93.50
std	2.95	7.81	8.59
min	6.00	10.00	82.00
25%	9.50	18.00	86.25
50%	11.00	25.00	92.00
75%	12.75	28.75	99.75
max	16.00	34.00	109.00

Table D.13: Number of nodes in each class of nodes for the third layer of the CNN (128-filter layer)

	Zero Nodes	Worse Nodes	Important Nodes
mean	3.00	16.60	44.40
std	1.63	6.67	6.29
min	0.00	8.00	35.00
25%	2.00	12.25	38.50
50%	3.00	16.00	46.00
75%	4.00	22.00	48.00
max	6.00	27.00	54.00

Table D.14: Number of nodes in each class of nodes for the fourth layer of the CNN (256-filter layer)

	Zero Nodes	Worse Nodes	Important Nodes
mean	0.60	3.90	27.50
std	0.84	3.63	3.72
min	0.00	0.00	21.00
25%	0.00	0.50	26.00
50%	0.00	3.50	28.50
75%	1.00	5.75	30.00
max	2.00	10.00	32.00

Table D.15: Number of nodes in each class of nodes for the fifth layer of the CNN (32-node layer)

Tables- Validation set

	Zero Nodes	Worse Nodes	Important Nodes
mean	0.40	12.20	19.40
std	0.70	2.66	2.50
min	0.00	8.00	16.00
25%	0.00	11.00	17.50
50%	0.00	11.50	20.00
75%	0.75	14.25	20.75
max	2.00	16.00	24.00

Table D.16: Number of nodes in each class of nodes for the first layer of the CNN (32-filter layer)

	Zero Nodes	Worse Nodes	Important Nodes
mean	1.40	25.60	37.00
std	1.17	6.26	5.83
min	0.00	16.00	29.00
25%	1.00	22.25	34.25
50%	1.00	26.00	35.50
75%	2.00	29.00	39.75
max	4.00	34.00	46.00

Table D.17: Number of nodes in each class of nodes for the second layer of the CNN (64-filter layer)

	Zero Nodes	Worse Nodes	Important Nodes
mean	10.80	47.80	69.40
std	2.90	7.80	6.42
min	7.00	36.00	58.00
25%	8.25	43.25	66.00
50%	11.50	45.00	71.00
75%	12.75	53.50	72.75
max	15.00	60.00	79.00

Table D.18: Number of nodes in each class of nodes for the third layer of the CNN (128-filter layer)

	Zero Nodes	Worse Nodes	Important Nodes
mean	44.90	133.60	77.50
std	5.15	21.81	19.65
min	36.00	103.00	42.00
25%	41.50	119.00	66.50
50%	45.50	131.50	79.50
75%	49.00	149.25	85.75
max	52.00	170.00	106.00

Table D.19: Number of nodes in each class of nodes for the second layer of the CNN (256-filter layer)

	Zero Nodes	Worse Nodes	Important Nodes
mean	27.10	12.50	24.40
std	4.33	3.75	2.46
min	20.00	6.00	20.00
25%	24.50	9.75	23.25
50%	26.50	13.00	24.50
75%	29.75	14.00	25.75
max	35.00	19.00	29.00

Table D.20: Number of nodes in each class of nodes for the third layer of the CNN (32-node layer)

D.1.3 CIFAR-10

Tables- Training set

	Zero Nodes	Worse Nodes	Important Nodes
mean	1.60	1.20	29.20
std	1.65	0.79	1.48
min	0.00	0.00	27.00
25%	0.25	1.00	29.00
50%	1.00	1.00	29.00
75%	2.75	2.00	30.50
max	5.00	2.00	31.00

Table D.21: Number of nodes in each class of nodes for the first layer of the CNN (32-filter layer)

	Zero Nodes	Worse Nodes	Important Nodes
mean	10.00	3.90	50.10
std	2.16	3.45	3.84
min	8.00	1.00	43.00
25%	8.25	2.00	47.75
50%	9.50	2.50	51.50
75%	11.00	3.75	52.75
max	15.00	12.00	54.00

Table D.22: Number of nodes in each class of nodes for the second layer of the CNN (64-filter layer)

	Zero Nodes	Worse Nodes	Important Nodes
mean	27.50	10.80	89.70
std	7.20	6.94	8.49
min	10.00	3.00	75.00
25%	26.00	5.25	85.50
50%	27.50	8.50	89.00
75%	31.75	17.25	93.75
max	35.00	22.00	107.00

Table D.23: Number of nodes in each class of nodes for the third layer of the CNN (128-filter layer)

	Zero Nodes	Worse Nodes	Important Nodes
mean	75.70	23.90	156.40
std	10.50	12.72	19.29
min	61.00	9.00	115.00
25%	69.25	14.00	148.50
50%	73.00	21.50	162.50
75%	83.75	30.25	169.75
max	92.00	49.00	176.00

Table D.24: Number of nodes in each class of nodes for the fourth layer of the CNN (256-filter layer)

	Zero Nodes	Worse Nodes	Important Nodes
mean	38.10	0.80	25.10
std	3.67	0.92	3.70
min	33.00	0.00	19.00
25%	35.50	0.00	23.00
50%	38.50	0.50	24.50
75%	40.00	1.75	28.25
max	44.00	2.00	31.00

Table D.25: Number of nodes in each class of nodes for the fifth layer of the CNN (32-node layer)

Tables- Validation set

	Zero Nodes	Worse Nodes	Important Nodes
mean	41.40	3.60	19.00
std	6.00	1.26	5.33
min	33.00	2.00	9.00
25%	37.25	3.00	17.25
50%	41.00	3.50	19.50
75%	44.00	4.00	23.25
max	52.00	6.00	25.00

Table D.26: Number of nodes in each class of nodes for the first layer of the CNN (32-filter layer)

	Zero Nodes	Worse Nodes	Important Nodes
mean	70.90	91.00	94.10
std	8.01	8.88	7.22
min	61.00	71.00	82.00
25%	63.25	87.00	90.25
50%	71.00	93.50	93.00
75%	78.75	97.25	98.25
max	80.00	99.00	107.00

Table D.27: Number of nodes in each class of nodes for the second layer of the CNN (64-filter layer)

	Zero Nodes	Worse Nodes	Important Nodes
mean	22.50	29.90	75.60
std	8.75	6.15	5.25
min	14.00	19.00	66.00
25%	16.25	25.75	73.25
50%	21.00	30.00	74.50
75%	25.50	33.25	78.50
max	43.00	40.00	85.00

Table D.28: Number of nodes in each class of nodes for the third layer of the CNN (128-filter layer)

	Zero Nodes	Worse Nodes	Important Nodes
mean	10.90	11.40	41.70
std	4.68	7.41	7.90
min	2.00	1.00	25.00
25%	9.00	7.50	37.25
50%	10.50	11.00	43.00
75%	14.25	13.75	47.75
max	18.00	28.00	51.00

Table D.29: Number of nodes in each class of nodes for the second layer of the CNN (256-filter layer)

	Zero Nodes	Worse Nodes	Important Nodes
mean	2.30	5.90	23.80
std	1.25	1.45	2.10
min	0.00	4.00	20.00
25%	2.00	5.00	23.00
50%	2.00	6.00	24.00
75%	3.00	6.75	24.75
max	4.00	8.00	28.00

Table D.30: Number of nodes in each class of nodes for the third layer of the CNN (32-node layer)

D.2 Effects of batch size on Node Importance

D.2.1 MNIST

Figures

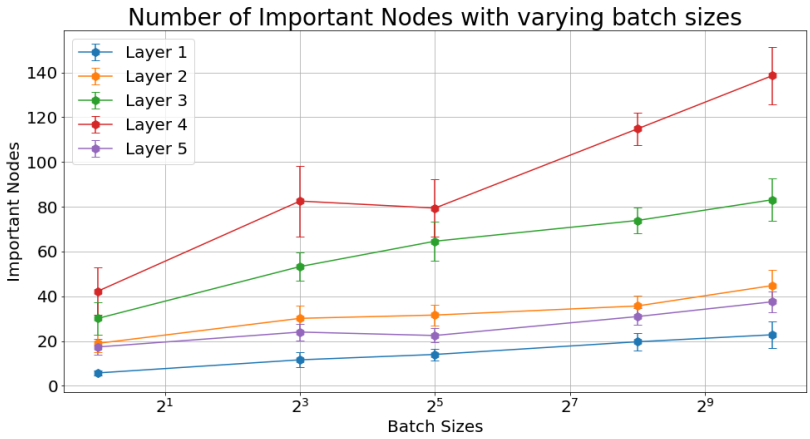

Figure D.1: Number of important nodes per batch size

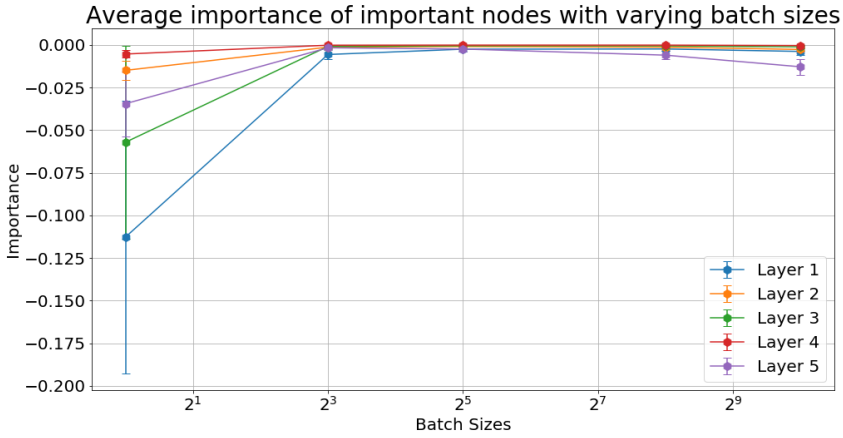

Figure D.2: Average node importance of important nodes per batch size

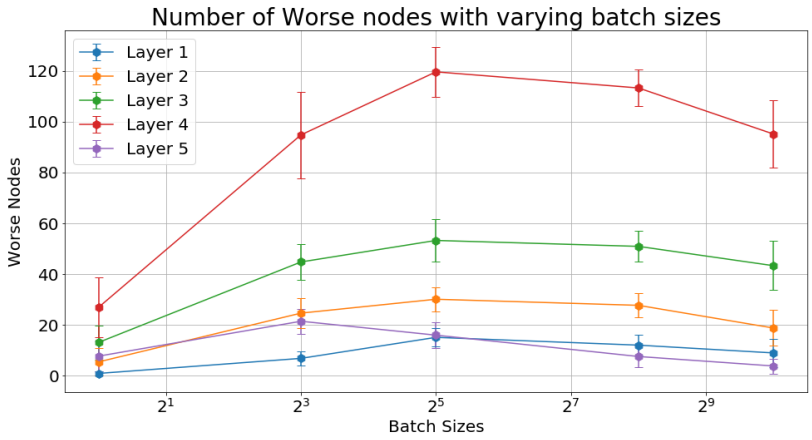

Figure D.3: Number of worse nodes per batch size

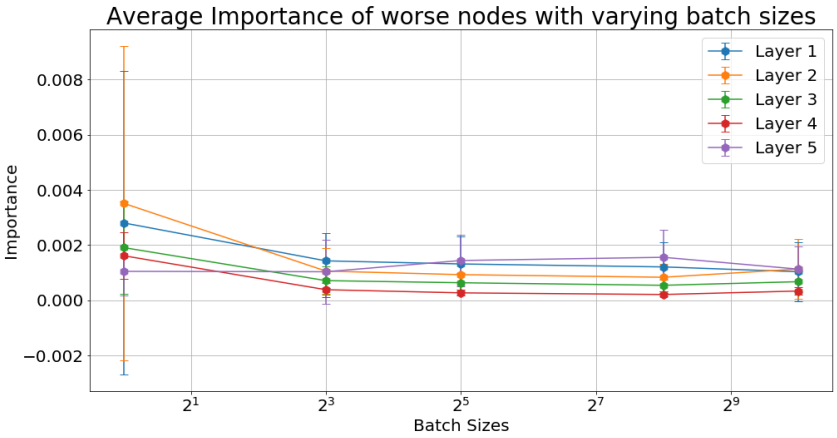

Figure D.4: Average node importance of worse nodes per batch size

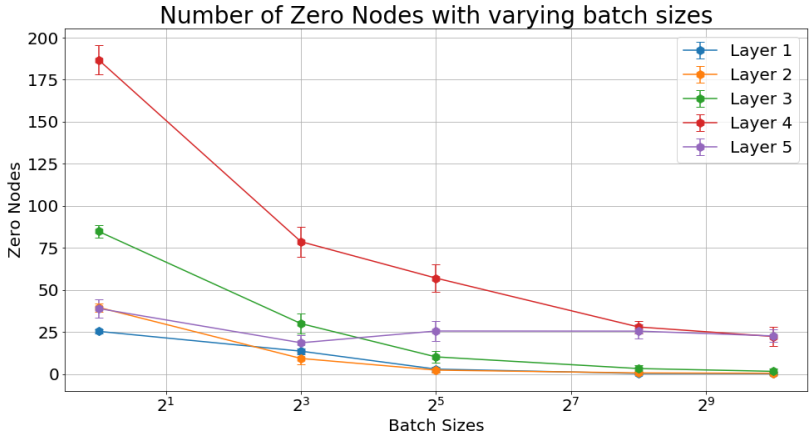

Figure D.5: Number of zero nodes per batch size

Tables

	1	8	32	256	1024
mean	0.9755	0.9883	0.9892	0.9897	0.9882
std	0.0059	0.0023	0.0022	0.0016	0.0012
min	0.9664	0.9839	0.9846	0.9869	0.9861
25%	0.9703	0.9873	0.9883	0.9884	0.9875
50%	0.9755	0.9890	0.9893	0.9902	0.9881
75%	0.9803	0.9900	0.9904	0.9912	0.9890
max	0.9844	0.9909	0.9932	0.9915	0.9909

Table D.31: Statistics of accuracy of CNNs trained on MNIST at each batch size

	1	8	32	256	1024
mean	0.1014	0.0466	0.0386	0.0314	0.0360
std	0.0285	0.0127	0.0088	0.0054	0.0031
min	0.0586	0.0328	0.0247	0.0246	0.0302
25%	0.0840	0.0368	0.0324	0.0271	0.0339
50%	0.0979	0.0433	0.0374	0.0310	0.0368
75%	0.1088	0.0526	0.0429	0.0337	0.0374
max	0.1744	0.0760	0.0547	0.0416	0.0421

Table D.32: Statistics of loss of CNNs trained on MNIST at each batch size

	1			8			32			256			1024		
	Zero Nodes	Worse Nodes	Important Nodes	Zero Nodes	Worse Nodes	Important Nodes	Zero Nodes	Worse Nodes	Important Nodes	Zero Nodes	Worse Nodes	Important Nodes	Zero Nodes	Worse Nodes	Important Nodes
mean	25.33	0.93	5.73	13.53	6.87	11.60	2.87	15.13	14.00	0.27	12.07	19.67	0.20	9.00	22.80
std	1.11	0.88	1.22	1.73	2.88	3.33	1.55	3.62	2.62	0.59	4.13	3.83	0.56	5.57	5.81
min	23.00	0.00	4.00	10.00	3.00	7.00	0.00	10.00	10.00	0.00	5.00	11.00	0.00	0.00	14.00
25%	25.00	0.00	5.00	13.00	5.00	9.00	2.00	12.00	12.00	0.00	9.50	18.00	0.00	4.50	18.50
50%	25.00	1.00	6.00	14.00	6.00	11.00	3.00	16.00	13.00	0.00	12.00	20.00	0.00	9.00	23.00
75%	26.00	1.00	7.00	15.00	8.50	13.50	3.50	18.00	16.00	0.00	14.00	22.50	0.00	13.50	27.50
max	27.00	3.00	7.00	16.00	14.00	18.00	6.00	20.00	19.00	2.00	21.00	25.00	2.00	18.00	32.00

Table D.33: Number of nodes in each class of nodes for the first layer of the CNN (32-filter layer)

	1			8			32			256			1024		
	Zero Nodes	Worse Nodes	Important Nodes	Zero Nodes	Worse Nodes	Important Nodes	Zero Nodes	Worse Nodes	Important Nodes	Zero Nodes	Worse Nodes	Important Nodes	Zero Nodes	Worse Nodes	Important Nodes
mean	39.60	5.47	18.93	9.20	24.67	30.13	2.27	30.13	31.60	0.60	27.73	35.67	0.40	18.80	44.80
std	2.44	5.40	3.92	3.47	5.73	5.84	1.28	4.84	4.76	0.74	4.65	4.73	0.63	7.03	6.84
min	34.00	0.00	9.00	4.00	12.00	20.00	0.00	22.00	25.00	0.00	18.00	29.00	0.00	4.00	37.00
25%	39.00	1.50	17.00	6.50	21.00	27.00	1.00	26.50	27.00	0.00	26.00	33.50	0.00	14.00	39.00
50%	40.00	5.00	19.00	8.00	26.00	29.00	2.00	30.00	31.00	0.00	27.00	36.00	0.00	21.00	43.00
75%	41.00	7.50	22.00	12.00	28.50	32.50	3.00	35.00	36.00	1.00	30.50	37.50	1.00	25.00	48.50
max	43.00	21.00	24.00	15.00	32.00	45.00	4.00	37.00	39.00	2.00	35.00	46.00	2.00	26.00	60.00

Table D.34: Number of nodes in each class of nodes for the second layer of the CNN (64-filter layer)

	1			8			32			256			1024		
	Zero Nodes	Worse Nodes	Important Nodes	Zero Nodes	Worse Nodes	Important Nodes	Zero Nodes	Worse Nodes	Important Nodes	Zero Nodes	Worse Nodes	Important Nodes	Zero Nodes	Worse Nodes	Important Nodes
mean	84.80	13.13	30.07	29.93	44.80	53.27	10.13	53.27	64.60	3.20	50.93	73.87	1.53	43.33	83.13
std	3.61	6.69	7.30	6.12	7.16	6.19	3.29	8.37	8.74	1.78	5.99	5.71	1.64	9.66	9.50
min	79.00	4.00	14.00	21.00	31.00	42.00	4.00	37.00	50.00	1.00	42.00	66.00	0.00	23.00	65.00
25%	82.50	7.50	28.00	24.50	39.50	50.00	8.00	47.00	60.00	2.00	45.50	69.50	0.00	37.00	79.00
50%	85.00	13.00	31.00	31.00	43.00	53.00	11.00	55.00	64.00	3.00	53.00	73.00	1.00	44.00	83.00
75%	87.00	17.00	36.00	34.50	52.00	57.00	12.00	59.50	67.00	4.50	55.50	78.50	2.50	47.50	87.00
max	91.00	26.00	39.00	40.00	55.00	62.00	16.00	67.00	87.00	7.00	60.00	85.00	5.00	61.00	105.00

Table D.35: Number of nodes in each class of nodes for the third layer of the CNN (128-filter layer)

	1			8			32			256			1024		
	Zero Nodes	Worse Nodes	Important Nodes	Zero Nodes	Worse Nodes	Important Nodes	Zero Nodes	Worse Nodes	Important Nodes	Zero Nodes	Worse Nodes	Important Nodes	Zero Nodes	Worse Nodes	Important Nodes
mean	186.87	26.93	42.20	78.67	94.80	82.53	57.00	119.60	79.40	27.93	113.27	114.8	22.27	95.13	138.60
std	8.75	11.68	10.66	9.04	16.97	15.87	8.41	9.81	12.93	3.71	7.24	7.3	5.50	13.37	12.81
min	163.00	9.00	26.00	65.00	62.00	61.00	49.00	106.00	44.00	23.00	98.00	105.0	15.00	74.00	109.00
25%	184.50	18.50	38.50	73.00	84.50	72.50	51.00	112.00	74.00	25.00	109.50	111.5	18.50	87.50	131.50
50%	189.00	25.00	40.00	77.00	99.00	76.00	53.00	118.00	80.00	27.00	114.00	115.0	21.00	94.00	142.00
75%	190.50	32.50	49.50	83.00	107.00	93.00	60.50	128.00	86.00	31.00	117.00	116.0	26.50	99.50	146.50
max	199.00	51.00	58.00	98.00	120.00	117.00	80.00	133.00	99.00	33.00	126.00	135.0	34.00	130.00	154.00

Table D.36: Number of nodes in each class of nodes for the fourth layer of the CNN (256-filter layer)

	1			8			32			256			1024		
	Zero Nodes	Worse Nodes	Important Nodes	Zero Nodes	Worse Nodes	Important Nodes	Zero Nodes	Worse Nodes	Important Nodes	Zero Nodes	Worse Nodes	Important Nodes	Zero Nodes	Worse Nodes	Important Nodes
mean	38.93	7.73	17.33	18.53	21.47	24.0	25.53	16.00	22.47	25.47	7.60	30.93	22.60	3.87	37.53
std	5.48	5.95	3.60	4.44	4.93	3.7	5.99	4.96	3.18	4.16	4.19	3.77	3.74	2.92	4.66
min	28.00	0.00	11.00	13.00	12.00	19.0	16.00	8.00	18.00	19.00	0.00	23.00	16.00	0.00	29.00
25%	37.00	4.50	16.00	15.00	18.00	21.0	21.00	12.50	20.50	22.50	5.00	28.00	20.50	1.50	34.50
50%	40.00	7.00	17.00	18.00	22.00	24.0	25.00	15.00	22.00	25.00	7.00	31.00	22.00	4.00	36.00
75%	43.00	8.00	18.50	20.50	26.00	27.5	30.50	19.00	23.50	28.00	11.00	33.50	25.50	5.00	41.50
max	46.00	22.00	27.00	29.00	29.00	31.0	34.00	26.00	30.00	33.00	14.00	37.00	28.00	9.00	44.00

Table D.37: Number of nodes in each class of nodes for the fifth layer of the CNN (32-node layer)

	1			8			32			256			1024		
	Zero Nodes	Worse Nodes	Important Nodes	Zero Nodes	Worse Nodes	Important Nodes	Zero Nodes	Worse Nodes	Important Nodes	Zero Nodes	Worse Nodes	Important Nodes	Zero Nodes	Worse Nodes	Important Nodes
mean	-0.0	0.0028	-0.1127	0.0	0.0014	-0.0057	0.0	0.0013	-0.0026	0.0	0.0012	-0.0024	-0.0	0.0010	-0.0039
std	0.0	0.0055	0.0801	0.0	0.0010	0.0026	0.0	0.0010	0.0015	0.0	0.0009	0.0012	0.0	0.0011	0.0020
min	-0.0	0.0000	-0.3357	-0.0	0.0003	-0.0113	-0.0	0.0004	-0.0063	-0.0	0.0003	-0.0050	-0.0	0.0000	-0.0087
25%	0.0	0.0000	-0.1490	-0.0	0.0008	-0.0076	-0.0	0.0005	-0.0033	0.0	0.0004	-0.0034	0.0	0.0004	-0.0051
50%	0.0	0.0002	-0.0915	0.0	0.0011	-0.0048	0.0	0.0009	-0.0020	0.0	0.0010	-0.0020	0.0	0.0005	-0.0026
75%	0.0	0.0032	-0.0599	0.0	0.0015	-0.0038	0.0	0.0017	-0.0017	0.0	0.0020	-0.0016	0.0	0.0013	-0.0024
max	0.0	0.0212	-0.0251	0.0	0.0040	-0.0027	0.0	0.0033	-0.0007	0.0	0.0030	-0.0009	0.0	0.0038	-0.0020

Table D.38: Average importance for each node class for the first layer of the CNN (32-filter layer)

	1			8			32			256			1024		
	Zero Nodes	Worse Nodes	Important Nodes	Zero Nodes	Worse Nodes	Important Nodes	Zero Nodes	Worse Nodes	Important Nodes	Zero Nodes	Worse Nodes	Important Nodes	Zero Nodes	Worse Nodes	Important Nodes
mean	-0.0	0.0028	-0.1127	0.0	0.0014	-0.0057	0.0	0.0013	-0.0026	0.0	0.0012	-0.0024	-0.0	0.0010	-0.0039
std	0.0	0.0055	0.0801	0.0	0.0010	0.0026	0.0	0.0010	0.0015	0.0	0.0009	0.0012	0.0	0.0011	0.0020
min	-0.0	0.0000	-0.3357	-0.0	0.0003	-0.0113	-0.0	0.0004	-0.0063	-0.0	0.0003	-0.0050	-0.0	0.0000	-0.0087
25%	0.0	0.0000	-0.1490	-0.0	0.0008	-0.0076	-0.0	0.0005	-0.0033	0.0	0.0004	-0.0034	0.0	0.0004	-0.0051
50%	0.0	0.0002	-0.0915	0.0	0.0011	-0.0048	0.0	0.0009	-0.0020	0.0	0.0010	-0.0020	0.0	0.0005	-0.0026
75%	0.0	0.0032	-0.0599	0.0	0.0015	-0.0038	0.0	0.0017	-0.0017	0.0	0.0020	-0.0016	0.0	0.0013	-0.0024
max	0.0	0.0212	-0.0251	0.0	0.0040	-0.0027	0.0	0.0033	-0.0007	0.0	0.0030	-0.0009	0.0	0.0038	-0.0020

Table D.39: Average importance for each node class for the second layer of the CNN (64-filter layer)

	1			8			32			256			1024		
	Zero Nodes	Worse Nodes	Important Nodes	Zero Nodes	Worse Nodes	Important Nodes	Zero Nodes	Worse Nodes	Important Nodes	Zero Nodes	Worse Nodes	Important Nodes	Zero Nodes	Worse Nodes	Important Nodes
mean	-0.0	0.0028	-0.1127	0.0	0.0014	-0.0057	0.0	0.0013	-0.0026	0.0	0.0012	-0.0024	-0.0	0.0010	-0.0039
std	0.0	0.0055	0.0801	0.0	0.0010	0.0026	0.0	0.0010	0.0015	0.0	0.0009	0.0012	0.0	0.0011	0.0020
min	-0.0	0.0000	-0.3357	-0.0	0.0003	-0.0113	-0.0	0.0004	-0.0063	-0.0	0.0003	-0.0050	-0.0	0.0000	-0.0087
25%	0.0	0.0000	-0.1490	-0.0	0.0008	-0.0076	-0.0	0.0005	-0.0033	0.0	0.0004	-0.0034	0.0	0.0004	-0.0051
50%	0.0	0.0002	-0.0915	0.0	0.0011	-0.0048	0.0	0.0009	-0.0020	0.0	0.0010	-0.0020	0.0	0.0005	-0.0026
75%	0.0	0.0032	-0.0599	0.0	0.0015	-0.0038	0.0	0.0017	-0.0017	0.0	0.0020	-0.0016	0.0	0.0013	-0.0024
max	0.0	0.0212	-0.0251	0.0	0.0040	-0.0027	0.0	0.0033	-0.0007	0.0	0.0030	-0.0009	0.0	0.0038	-0.0020

Table D.40: Average importance for each node class for the third layer of the CNN (128-filter layer)

	1			8			32			256			1024		
	Zero Nodes	Worse Nodes	Important Nodes	Zero Nodes	Worse Nodes	Important Nodes	Zero Nodes	Worse Nodes	Important Nodes	Zero Nodes	Worse Nodes	Important Nodes	Zero Nodes	Worse Nodes	Important Nodes
mean	-0.0	0.0028	-0.1127	0.0	0.0014	-0.0057	0.0	0.0013	-0.0026	0.0	0.0012	-0.0024	-0.0	0.0010	-0.0039
std	0.0	0.0055	0.0801	0.0	0.0010	0.0026	0.0	0.0010	0.0015	0.0	0.0009	0.0012	0.0	0.0011	0.0020
min	-0.0	0.0000	-0.3357	-0.0	0.0003	-0.0113	-0.0	0.0004	-0.0063	-0.0	0.0003	-0.0050	-0.0	0.0000	-0.0087
25%	0.0	0.0000	-0.1490	-0.0	0.0008	-0.0076	-0.0	0.0005	-0.0033	0.0	0.0004	-0.0034	0.0	0.0004	-0.0051
50%	0.0	0.0002	-0.0915	0.0	0.0011	-0.0048	0.0	0.0009	-0.0020	0.0	0.0010	-0.0020	0.0	0.0005	-0.0026
75%	0.0	0.0032	-0.0599	0.0	0.0015	-0.0038	0.0	0.0017	-0.0017	0.0	0.0020	-0.0016	0.0	0.0013	-0.0024
max	0.0	0.0212	-0.0251	0.0	0.0040	-0.0027	0.0	0.0033	-0.0007	0.0	0.0030	-0.0009	0.0	0.0038	-0.0020

Table D.41: Average importance for each node class for the fourth layer of the CNN (256-filter layer)

	1			8			32			256			1024		
	Zero Nodes	Worse Nodes	Important Nodes	Zero Nodes	Worse Nodes	Important Nodes	Zero Nodes	Worse Nodes	Important Nodes	Zero Nodes	Worse Nodes	Important Nodes	Zero Nodes	Worse Nodes	Important Nodes
mean	-0.0	0.0010	-0.0345	-0.0	0.0010	-0.0017	-0.0	0.0014	-0.0025	0.0	0.0016	-0.0061	-0.0	0.0011	-0.0129
std	0.0	0.0009	0.0190	0.0	0.0011	0.0007	0.0	0.0009	0.0013	0.0	0.0010	0.0023	0.0	0.0008	0.0046
min	-0.0	0.0000	-0.0675	-0.0	0.0003	-0.0034	-0.0	0.0004	-0.0058	-0.0	0.0000	-0.0109	-0.0	0.0000	-0.0253
25%	-0.0	0.0004	-0.0432	-0.0	0.0005	-0.0021	-0.0	0.0008	-0.0032	0.0	0.0007	-0.0066	0.0	0.0005	-0.0136
50%	-0.0	0.0007	-0.0356	-0.0	0.0007	-0.0016	-0.0	0.0012	-0.0020	0.0	0.0016	-0.0060	0.0	0.0011	-0.0124
75%	0.0	0.0016	-0.0211	0.0	0.0011	-0.0011	0.0	0.0015	-0.0016	0.0	0.0023	-0.0044	0.0	0.0014	-0.0107
max	0.0	0.0027	-0.0052	0.0	0.0048	-0.0010	0.0	0.0040	-0.0010	0.0	0.0031	-0.0034	0.0	0.0031	-0.0068

Table D.42: Average importance for each node class for the fifth layer of the CNN (32-node layer)

D.2.2 Fashion MNIST

Figures

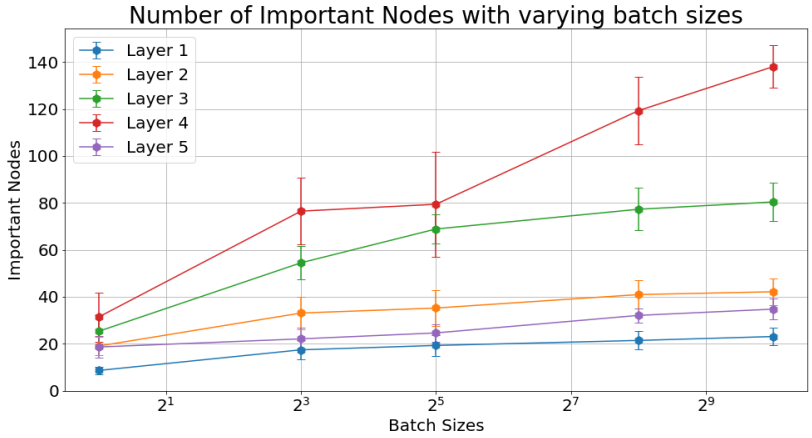

Figure D.6: Number of important nodes per batch size

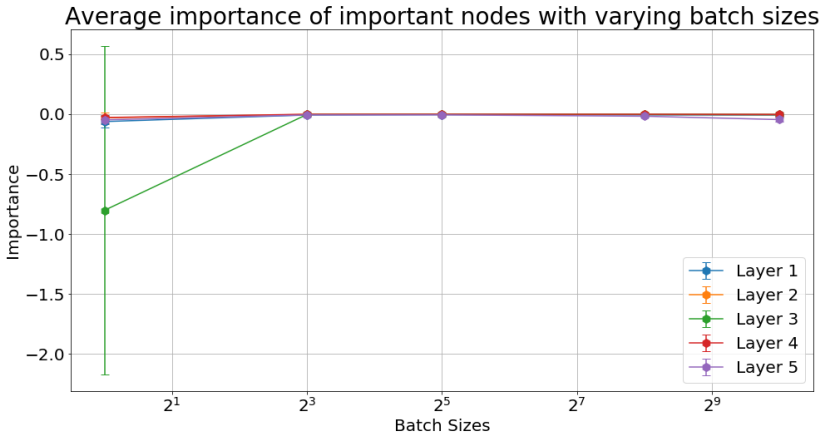

Figure D.7: Average node importance of important nodes per batch size

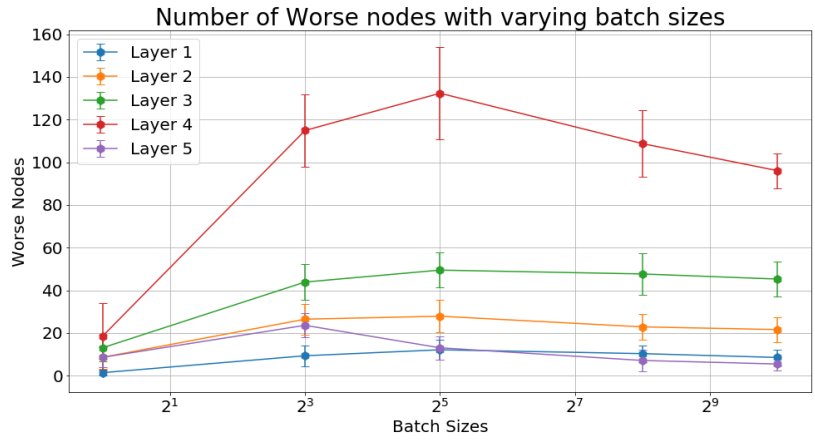

Figure D.8: Number of worse nodes per batch size

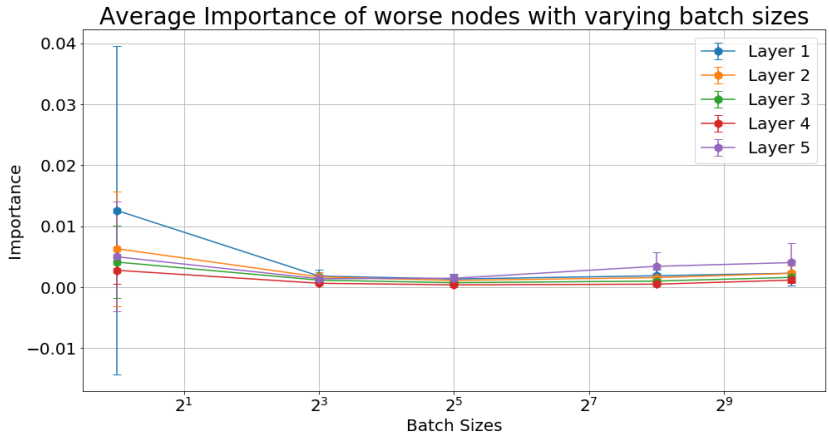

Figure D.9: Average node importance of worse nodes per batch size

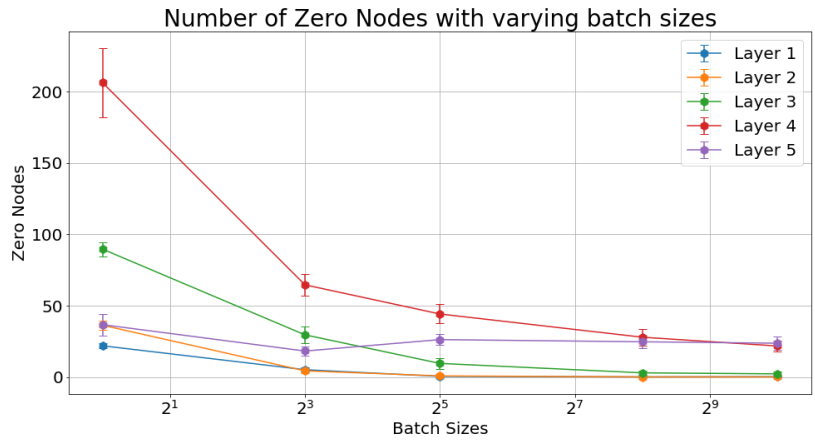

Figure D.10: Number of zero nodes per batch size

Tables

	1	8	32	256	1024
mean	0.8674	0.9065	0.9114	0.8976	0.8715
std	0.0072	0.0044	0.0037	0.0049	0.0049
min	0.8502	0.8975	0.9034	0.8888	0.8631
25%	0.8627	0.9042	0.9092	0.8946	0.8686
50%	0.8692	0.9079	0.9121	0.8970	0.8709
75%	0.8713	0.9090	0.9132	0.9007	0.8741
max	0.8796	0.9136	0.9172	0.9068	0.8821

Table D.43: Statistics of accuracy of CNNs trained on Fashion MNIST at each batch size

	1	8	32	256	1024
mean	0.4019	0.2807	0.2526	0.2816	0.3540
std	0.0357	0.0152	0.0088	0.0120	0.0127
min	0.3635	0.2565	0.2396	0.2652	0.3279
25%	0.3761	0.2713	0.2459	0.2697	0.3446
50%	0.3889	0.2788	0.2490	0.2823	0.3557
75%	0.4192	0.2855	0.2593	0.2909	0.3614
max	0.4878	0.3196	0.2708	0.3009	0.3728

Table D.44: Statistics of loss of CNNs trained on Fashion MNIST at each batch size

	1			8			32			256			1024		
	Zero Nodes	Worse Nodes	Important Nodes	Zero Nodes	Worse Nodes	Important Nodes	Zero Nodes	Worse Nodes	Important Nodes	Zero Nodes	Worse Nodes	Important Nodes	Zero Nodes	Worse Nodes	Important Nodes
mean	22.00	1.4	8.60	5.27	9.33	17.40	0.60	12.13	19.27	0.27	10.33	21.40	0.40	8.53	23.07
std	1.69	1.4	1.72	1.53	4.78	4.14	0.74	4.53	4.50	0.46	3.96	3.96	0.51	3.74	3.84
min	19.00	0.0	5.00	3.00	2.00	10.00	0.00	4.00	12.00	0.00	4.00	14.00	0.00	2.00	16.00
25%	21.00	0.0	8.00	4.00	7.00	15.00	0.00	10.00	16.00	0.00	7.50	20.00	0.00	6.00	20.50
50%	22.00	2.0	8.00	5.00	9.00	17.00	0.00	12.00	19.00	0.00	11.00	21.00	0.00	7.00	25.00
75%	23.00	2.0	9.50	6.00	11.00	21.00	1.00	15.50	21.00	0.50	12.00	24.00	1.00	11.50	25.50
max	25.00	4.0	12.00	8.00	18.00	24.00	2.00	20.00	28.00	1.00	18.00	28.00	1.00	15.00	30.00

Table D.45: Number of nodes in each class of nodes for the first layer of the CNN (32-filter layer)

	1			8			32			256			1024		
	Zero Nodes	Worse Nodes	Important Nodes	Zero Nodes	Worse Nodes	Important Nodes	Zero Nodes	Worse Nodes	Important Nodes	Zero Nodes	Worse Nodes	Important Nodes	Zero Nodes	Worse Nodes	Important Nodes
mean	36.40	8.60	19.00	4.47	26.47	33.07	0.93	27.87	35.20	0.20	22.87	40.93	0.27	21.60	42.13
std	3.33	5.21	3.82	1.64	7.20	6.78	0.96	7.56	7.63	0.41	6.13	6.09	0.46	5.79	5.80
min	30.00	1.00	13.00	2.00	12.00	20.00	0.00	14.00	23.00	0.00	10.00	32.00	0.00	9.00	33.00
25%	35.50	5.50	16.50	3.50	23.00	30.00	0.00	25.00	30.50	0.00	19.00	37.50	0.00	19.00	37.50
50%	37.00	8.00	20.00	5.00	27.00	33.00	1.00	28.00	33.00	0.00	24.00	40.00	0.00	20.00	43.00
75%	38.50	12.00	21.00	6.00	29.00	37.50	1.50	32.50	38.00	0.00	26.50	44.50	0.50	26.50	45.00
max	40.00	19.00	26.00	7.00	40.00	46.00	3.00	41.00	50.00	1.00	32.00	54.00	1.00	31.00	55.00

Table D.46: Number of nodes in each class of nodes for the second layer of the CNN (64-filter layer)

	1			8			32			256			1024	
	Zero Nodes	Worse Nodes	Important Nodes	Zero Nodes	Worse Nodes	Important Nodes	Zero Nodes	Worse Nodes	Important Nodes	Zero Nodes	Worse Nodes	Important Nodes	Zero Nodes	Worse Nodes	Important Nodes
mean	89.67	13.07	25.27	29.67	43.87	54.47	9.67	49.47	68.87	3.07	47.67	77.27	2.33	45.27	80.4
std	5.04	6.23	6.98	5.92	8.33	7.08	3.68	8.13	6.19	1.39	9.63	9.04	1.80	8.03	8.2
min	78.00	1.00	15.00	22.00	29.00	41.00	4.00	31.00	58.00	1.00	23.00	66.00	0.00	31.00	68.0
25%	86.50	9.00	18.50	25.50	38.00	51.00	7.50	46.50	65.00	2.00	43.50	70.50	1.00	38.50	74.5
50%	89.00	13.00	25.00	29.00	45.00	53.00	9.00	51.00	68.00	3.00	47.00	76.00	2.00	44.00	83.0
75%	93.50	18.00	30.50	34.00	49.00	57.50	10.50	54.00	72.50	4.00	56.00	82.00	3.00	51.50	85.5
max	96.00	25.00	37.00	42.00	59.00	69.00	17.00	63.00	83.00	5.00	59.00	100.00	6.00	57.00	96.0

Table D.47: Number of nodes in each class of nodes for the third layer of the CNN (128-filter layer)

	1			8			32			256			1024	
	Zero Nodes	Worse Nodes	Important Nodes	Zero Nodes	Worse Nodes	Important Nodes	Zero Nodes	Worse Nodes	Important Nodes	Zero Nodes	Worse Nodes	Important Nodes	Zero Nodes	Worse Nodes	Important Nodes
mean	206.20	18.47	31.33	64.67	114.87	76.47	44.27	132.33	79.4	28.00	108.73	119.27	21.87	96.07	138.07
std	24.21	15.55	10.53	7.39	17.10	14.12	6.70	21.71	22.4	5.99	15.55	14.43	3.68	8.17	9.14
min	123.00	5.00	21.00	41.00	78.00	59.00	35.00	82.00	41.0	20.00	73.00	102.00	16.00	84.00	120.00
25%	208.00	8.50	26.50	64.00	111.00	66.50	39.00	123.50	66.5	22.50	104.00	109.50	19.50	89.00	133.00
50%	211.00	14.00	28.00	67.00	117.00	74.00	43.00	134.00	78.0	29.00	112.00	115.00	22.00	97.00	138.00
75%	217.00	22.50	33.50	68.00	123.00	80.00	49.50	141.00	92.0	32.50	119.50	126.50	24.50	100.00	144.00
max	227.00	67.00	66.00	73.00	149.00	111.00	56.00	172.00	129.0	38.00	128.00	150.00	28.00	110.00	154.00

Table D.48: Number of nodes in each class of nodes for the fourth layer of the CNN (256-filter layer)

	1			8			32			256			1024	
	Zero Nodes	Worse Nodes	Important Nodes	Zero Nodes	Worse Nodes	Important Nodes	Zero Nodes	Worse Nodes	Important Nodes	Zero Nodes	Worse Nodes	Important Nodes	Zero Nodes	Worse Nodes	Important Nodes
mean	36.80	8.60	18.60	18.33	23.60	22.07	26.33	13.07	24.60	24.80	7.13	32.07	23.80	5.47	34.73
std	7.52	4.58	4.63	3.31	5.73	4.68	3.92	5.52	3.81	4.39	5.14	3.06	4.68	3.20	4.50
min	20.00	3.00	11.00	12.00	12.00	17.00	20.00	5.00	18.00	16.00	0.00	26.00	14.00	0.00	27.00
25%	36.00	6.00	16.50	16.00	20.00	19.00	25.00	10.00	22.00	21.50	3.50	31.50	21.00	3.00	32.00
50%	38.00	8.00	18.00	18.00	24.00	21.00	26.00	11.00	25.00	26.00	6.00	32.00	24.00	6.00	34.00
75%	39.00	9.50	20.00	21.50	28.00	24.00	28.00	14.50	27.50	28.00	10.50	34.00	25.50	7.50	37.00
max	47.00	20.00	31.00	23.00	32.00	36.00	37.00	25.00	30.00	31.00	16.00	37.00	33.00	11.00	44.00

Table D.49: Number of nodes in each class of nodes for the fifth layer of the CNN (32-node layer)

	1			8			32			256			1024	
	Zero Nodes	Worse Nodes	Important Nodes	Zero Nodes	Worse Nodes	Important Nodes	Zero Nodes	Worse Nodes	Important Nodes	Zero Nodes	Worse Nodes	Important Nodes	Zero Nodes	Worse Nodes	Important Nodes
mean	-0.0	0.0126	-0.0613	-0.0	0.0018	-0.0067	0.0	0.0013	-0.0038	-0.0	0.0019	-0.0062	0.0	0.0023	-0.0082
std	0.0	0.0270	0.0527	0.0	0.0010	0.0027	0.0	0.0007	0.0011	0.0	0.0010	0.0021	0.0	0.0020	0.0027
min	-0.0	0.0000	-0.1895	-0.0	0.0006	-0.0132	-0.0	0.0001	-0.0056	-0.0	0.0005	-0.0121	0.0	0.0002	-0.0142
25%	0.0	0.0000	-0.0737	-0.0	0.0011	-0.0077	-0.0	0.0008	-0.0048	0.0	0.0011	-0.0068	0.0	0.0009	-0.0096
50%	0.0	0.0027	-0.0326	-0.0	0.0017	-0.0061	0.0	0.0015	-0.0036	0.0	0.0017	-0.0062	0.0	0.0020	-0.0073
75%	0.0	0.0055	-0.0296	0.0	0.0021	-0.0046	0.0	0.0018	-0.0029	0.0	0.0026	-0.0050	0.0	0.0031	-0.0064
max	0.0	0.0817	-0.0191	0.0	0.0042	-0.0037	0.0	0.0026	-0.0026	0.0	0.0035	-0.0035	0.0	0.0081	-0.0048

Table D.50: Average importance for each node class for the first layer of the CNN (32-filter layer)

	1			8			32			256			1024	
	Zero Nodes	Worse Nodes	Important Nodes	Zero Nodes	Worse Nodes	Important Nodes	Zero Nodes	Worse Nodes	Important Nodes	Zero Nodes	Worse Nodes	Important Nodes	Zero Nodes	Worse Nodes	Important Nodes
mean	-0.0	0.0126	-0.0613	-0.0	0.0018	-0.0067	0.0	0.0013	-0.0038	-0.0	0.0019	-0.0062	0.0	0.0023	-0.0082
std	0.0	0.0270	0.0527	0.0	0.0010	0.0027	0.0	0.0007	0.0011	0.0	0.0010	0.0021	0.0	0.0020	0.0027
min	-0.0	0.0000	-0.1895	-0.0	0.0006	-0.0132	-0.0	0.0001	-0.0056	-0.0	0.0005	-0.0121	0.0	0.0002	-0.0142
25%	0.0	0.0000	-0.0737	-0.0	0.0011	-0.0077	-0.0	0.0008	-0.0048	0.0	0.0011	-0.0068	0.0	0.0009	-0.0096
50%	0.0	0.0027	-0.0326	-0.0	0.0017	-0.0061	0.0	0.0015	-0.0036	0.0	0.0017	-0.0062	0.0	0.0020	-0.0073
75%	0.0	0.0055	-0.0296	0.0	0.0021	-0.0046	0.0	0.0018	-0.0029	0.0	0.0026	-0.0050	0.0	0.0031	-0.0064
max	0.0	0.0817	-0.0191	0.0	0.0042	-0.0037	0.0	0.0026	-0.0026	0.0	0.0035	-0.0035	0.0	0.0081	-0.0048

Table D.51: Average importance for each node class for the second layer of the CNN (64-filter layer)

	1			8			32			256			1024		
	Zero Nodes	Worse Nodes	Important Nodes	Zero Nodes	Worse Nodes	Important Nodes	Zero Nodes	Worse Nodes	Important Nodes	Zero Nodes	Worse Nodes	Important Nodes	Zero Nodes	Worse Nodes	Important Nodes
mean	-0.0	0.0126	-0.0613	-0.0	0.0018	-0.0067	0.0	0.0013	-0.0038	-0.0	0.0019	-0.0062	0.0	0.0023	-0.0082
std	0.0	0.0270	0.0527	0.0	0.0010	0.0027	0.0	0.0007	0.0011	0.0	0.0010	0.0021	0.0	0.0020	0.0027
min	-0.0	0.0000	-0.1895	-0.0	0.0006	-0.0132	-0.0	0.0001	-0.0056	-0.0	0.0005	-0.0121	0.0	0.0002	-0.0142
25%	0.0	0.0000	-0.0737	-0.0	0.0011	-0.0077	-0.0	0.0008	-0.0048	0.0	0.0011	-0.0068	0.0	0.0009	-0.0096
50%	0.0	0.0027	-0.0326	-0.0	0.0017	-0.0061	0.0	0.0015	-0.0036	0.0	0.0017	-0.0062	0.0	0.0020	-0.0073
75%	0.0	0.0055	-0.0296	0.0	0.0021	-0.0046	0.0	0.0018	-0.0029	0.0	0.0026	-0.0050	0.0	0.0031	-0.0064
max	0.0	0.0817	-0.0191	0.0	0.0042	-0.0037	0.0	0.0026	-0.0026	0.0	0.0035	-0.0035	0.0	0.0081	-0.0048

Table D.52: Average importance for each node class for the third layer of the CNN (128-filter layer)

	1			8			32			256			1024		
	Zero Nodes	Worse Nodes	Important Nodes	Zero Nodes	Worse Nodes	Important Nodes	Zero Nodes	Worse Nodes	Important Nodes	Zero Nodes	Worse Nodes	Important Nodes	Zero Nodes	Worse Nodes	Important Nodes
mean	-0.0	0.0126	-0.0613	-0.0	0.0018	-0.0067	0.0	0.0013	-0.0038	-0.0	0.0019	-0.0062	0.0	0.0023	-0.0082
std	0.0	0.0270	0.0527	0.0	0.0010	0.0027	0.0	0.0007	0.0011	0.0	0.0010	0.0021	0.0	0.0020	0.0027
min	-0.0	0.0000	-0.1895	-0.0	0.0006	-0.0132	-0.0	0.0001	-0.0056	-0.0	0.0005	-0.0121	0.0	0.0002	-0.0142
25%	0.0	0.0000	-0.0737	-0.0	0.0011	-0.0077	-0.0	0.0008	-0.0048	0.0	0.0011	-0.0068	0.0	0.0009	-0.0096
50%	0.0	0.0027	-0.0326	-0.0	0.0017	-0.0061	0.0	0.0015	-0.0036	0.0	0.0017	-0.0062	0.0	0.0020	-0.0073
75%	0.0	0.0055	-0.0296	0.0	0.0021	-0.0046	0.0	0.0018	-0.0029	0.0	0.0026	-0.0050	0.0	0.0031	-0.0064
max	0.0	0.0817	-0.0191	0.0	0.0042	-0.0037	0.0	0.0026	-0.0026	0.0	0.0035	-0.0035	0.0	0.0081	-0.0048

Table D.53: Average importance for each node class for the fourth layer of the CNN (256-filter layer)

	1			8			32			256			1024		
	Zero Nodes	Worse Nodes	Important Nodes	Zero Nodes	Worse Nodes	Important Nodes	Zero Nodes	Worse Nodes	Important Nodes	Zero Nodes	Worse Nodes	Important Nodes	Zero Nodes	Worse Nodes	Important Nodes
mean	-0.0	0.0050	-0.0473	-0.0	0.0014	-0.0060	-0.0	0.0015	-0.0057	0.0	0.0034	-0.0168	-0.0	0.0040	-0.0450
std	0.0	0.0090	0.0347	0.0	0.0006	0.0027	0.0	0.0007	0.0018	0.0	0.0023	0.0048	0.0	0.0032	0.0182
min	-0.0	0.0002	-0.1225	-0.0	0.0007	-0.0135	-0.0	0.0003	-0.0088	-0.0	0.0000	-0.0283	-0.0	0.0000	-0.0834
25%	-0.0	0.0006	-0.0564	-0.0	0.0010	-0.0069	-0.0	0.0009	-0.0073	0.0	0.0018	-0.0189	0.0	0.0011	-0.0501
50%	-0.0	0.0010	-0.0332	0.0	0.0012	-0.0057	-0.0	0.0015	-0.0057	0.0	0.0028	-0.0149	0.0	0.0038	-0.0410
75%	0.0	0.0033	-0.0234	0.0	0.0015	-0.0043	0.0	0.0017	-0.0040	0.0	0.0056	-0.0134	0.0	0.0055	-0.0317
max	0.0	0.0298	-0.0154	0.0	0.0028	-0.0029	0.0	0.0029	-0.0038	0.0	0.0074	-0.0118	0.0	0.0099	-0.0241

Table D.54: Average importance for each node class for the fifth layer of the CNN (32-node layer)

D.2.3 CIFAR-10

Tables

	1			8			32			256			1024		
	Zero Nodes	Worse Nodes	Important Nodes	Zero Nodes	Worse Nodes	Important Nodes	Zero Nodes	Worse Nodes	Important Nodes	Zero Nodes	Worse Nodes	Important Nodes	Zero Nodes	Worse Nodes	Important Nodes
mean	31.13	0.07	0.80	10.40	3.0	18.60	1.87	6.07	24.07	0.40	9.87	21.73	0.20	4.13	27.67
std	2.29	0.26	2.11	2.77	2.2	2.67	1.06	2.02	2.09	0.74	4.67	4.32	0.41	3.91	3.92
min	25.00	0.00	0.00	6.00	0.0	15.00	0.00	2.00	20.00	0.00	2.00	14.00	0.00	0.00	21.00
25%	32.00	0.00	0.00	9.00	1.5	17.00	1.00	5.00	23.50	0.00	7.50	19.50	0.00	1.00	25.00
50%	32.00	0.00	0.00	10.00	3.0	18.00	2.00	6.00	24.00	0.00	10.00	22.00	0.00	4.00	28.00
75%	32.00	0.00	0.00	12.50	3.5	20.50	2.50	7.00	25.00	0.50	12.50	23.50	0.00	7.00	31.00
max	32.00	1.00	6.00	15.00	8.0	24.00	4.00	10.00	28.00	2.00	18.00	29.00	1.00	10.00	32.00

Table D.55: Number of nodes in each class of nodes for the first layer of the CNN (32-filter layer)

	1			8			32			256			1024		
	Zero Nodes	Worse Nodes	Important Nodes	Zero Nodes	Worse Nodes	Important Nodes	Zero Nodes	Worse Nodes	Important Nodes	Zero Nodes	Worse Nodes	Important Nodes	Zero Nodes	Worse Nodes	Important Nodes
mean	61.40	0.33	2.27	20.87	7.13	36.00	8.27	13.53	42.2	2.67	21.07	40.27	0.53	16.80	46.67
std	6.86	0.90	5.99	3.94	3.42	4.88	3.56	5.90	6.1	1.50	3.53	3.53	1.06	3.76	4.05
min	44.00	0.00	0.00	12.00	2.00	28.00	5.00	5.00	31.0	1.00	15.00	36.00	0.00	7.00	40.00
25%	64.00	0.00	0.00	20.00	4.00	31.00	5.50	10.00	38.0	1.50	18.00	37.00	0.00	14.50	44.00
50%	64.00	0.00	0.00	20.00	8.00	37.00	7.00	12.00	43.0	2.00	22.00	39.00	0.00	18.00	45.00
75%	64.00	0.00	0.00	23.00	8.00	39.50	9.50	15.50	47.0	4.00	23.50	43.00	0.50	19.50	49.50
max	64.00	3.00	18.00	26.00	14.00	44.00	16.00	28.00	51.0	5.00	26.00	46.00	3.00	21.00	56.00

Table D.56: Number of nodes in each class of nodes for the second layer of the CNN (64-filter layer)

	1			8			32			256			1024		
	Zero Nodes	Worse Nodes	Important Nodes	Zero Nodes	Worse Nodes	Important Nodes	Zero Nodes	Worse Nodes	Important Nodes	Zero Nodes	Worse Nodes	Important Nodes	Zero Nodes	Worse Nodes	Important Nodes
mean	123.27	1.07	3.67	58.53	13.73	55.73	25.33	29.87	72.80	6.13	47.87	74.00	3.87	40.20	83.93
std	13.23	3.39	9.98	13.99	6.54	12.62	7.66	7.50	7.92	2.59	4.82	4.23	1.85	8.19	7.59
min	81.00	0.00	0.00	29.00	5.00	38.00	16.00	15.00	58.00	2.00	41.00	68.00	1.00	27.00	70.00
25%	128.00	0.00	0.00	50.50	10.50	47.00	19.00	26.50	67.00	4.50	43.00	71.00	3.00	33.50	78.00
50%	128.00	0.00	0.00	62.00	12.00	56.00	26.00	30.00	74.00	6.00	50.00	74.00	4.00	41.00	84.00
75%	128.00	0.00	0.00	69.00	15.50	61.00	28.50	32.00	79.50	7.00	52.00	76.50	4.50	46.00	90.50
max	128.00	13.00	34.00	77.00	29.00	86.00	43.00	46.00	86.00	12.00	54.00	82.00	8.00	55.00	97.00

Table D.57: Number of nodes in each class of nodes for the third layer of the CNN (128-filter layer)

	1			8			32			256			1024		
	Zero Nodes	Worse Nodes	Important Nodes	Zero Nodes	Worse Nodes	Important Nodes	Zero Nodes	Worse Nodes	Important Nodes	Zero Nodes	Worse Nodes	Important Nodes	Zero Nodes	Worse Nodes	Important Nodes
mean	248.07	2.67	5.27	103.33	60.53	92.13	81.73	80.87	93.40	46.07	99.47	110.47	32.60	80.73	142.67
std	20.59	7.02	13.66	18.78	13.32	12.81	13.54	13.03	11.35	6.82	10.40	8.45	7.91	9.84	13.59
min	187.00	0.00	0.00	71.00	44.00	72.00	56.00	59.00	71.00	37.00	83.00	95.00	18.00	67.00	117.00
25%	255.00	0.00	0.00	89.00	47.50	87.00	74.50	70.00	84.00	41.50	92.00	105.50	28.50	74.00	132.50
50%	256.00	0.00	0.00	104.00	62.00	90.00	80.00	83.00	95.00	45.00	99.00	109.00	34.00	80.00	144.00
75%	256.00	0.50	0.50	117.50	70.50	92.50	88.00	89.50	101.00	48.00	109.50	117.50	38.50	89.00	153.00
max	256.00	25.00	44.00	140.00	82.00	123.00	112.00	106.00	114.00	64.00	113.00	123.00	45.00	99.00	164.00

Table D.58: Number of nodes in each class of nodes for the fourth layer of the CNN (256-filter layer)

	1			8			32			256			1024		
	Zero Nodes	Worse Nodes	Important Nodes	Zero Nodes	Worse Nodes	Important Nodes	Zero Nodes	Worse Nodes	Important Nodes	Zero Nodes	Worse Nodes	Important Nodes	Zero Nodes	Worse Nodes	Important Nodes
mean	60.73	0.73	2.53	36.4	7.07	20.53	43.13	2.80	18.07	37.00	5.20	21.80	35.73	1.13	27.13
std	8.64	2.15	6.70	5.5	4.06	3.70	4.37	2.21	2.89	4.83	3.03	3.91	4.65	1.46	3.93
min	38.00	0.00	0.00	22.0	3.00	16.00	36.00	0.00	12.00	31.00	1.00	15.00	26.00	0.00	22.00
25%	64.00	0.00	0.00	33.0	5.00	17.00	41.00	1.00	16.00	33.00	3.50	19.00	33.00	0.00	24.50
50%	64.00	0.00	0.00	38.0	6.00	21.00	43.00	2.00	18.00	37.00	4.00	22.00	38.00	1.00	26.00
75%	64.00	0.00	0.00	40.0	8.00	23.50	45.50	4.00	20.00	38.50	7.00	23.50	39.00	1.50	30.00
max	64.00	8.00	20.00	43.0	20.00	26.00	52.00	7.00	22.00	46.00	11.00	29.00	42.00	4.00	34.00

Table D.59: Number of nodes in each class of nodes for the fifth layer of the CNN (32-node layer)

	1			8			32			256			1024		
	Zero Nodes	Worse Nodes	Important Nodes	Zero Nodes	Worse Nodes	Important Nodes	Zero Nodes	Worse Nodes	Important Nodes	Zero Nodes	Worse Nodes	Important Nodes	Zero Nodes	Worse Nodes	Important Nodes
mean	-0.0	0.0004	-0.0467	0.0	0.0013	-0.0711	-0.0	0.0048	-0.0341	0.0	0.0856	-0.1385	0.0	0.0061	-0.1007
std	0.0	0.0016	0.1235	0.0	0.0018	0.0205	0.0	0.0036	0.0071	0.0	0.1352	0.1498	0.0	0.0048	0.0299
min	-0.0	0.0000	-0.3654	-0.0	0.0000	-0.1089	-0.0	0.0010	-0.0492	0.0	0.0079	-0.6552	0.0	0.0000	-0.1643
25%	0.0	0.0000	0.0000	-0.0	0.0003	-0.0867	-0.0	0.0023	-0.0387	0.0	0.0191	-0.1429	0.0	0.0025	-0.1185
50%	0.0	0.0000	0.0000	0.0	0.0010	-0.0700	0.0	0.0042	-0.0322	0.0	0.0389	-0.0867	0.0	0.0050	-0.0956
75%	0.0	0.0000	0.0000	0.0	0.0013	-0.0575	0.0	0.0055	-0.0298	0.0	0.0626	-0.0683	0.0	0.0092	-0.0793
max	0.0	0.0060	0.0000	0.0	0.0071	-0.0328	0.0	0.0147	-0.0238	0.0	0.4927	-0.0608	0.0	0.0150	-0.0582

Table D.60: Average importance for each node class for the first layer of the CNN (32-filter layer)

	1			8			32			256			1024		
	Zero Nodes	Worse Nodes	Important Nodes	Zero Nodes	Worse Nodes	Important Nodes	Zero Nodes	Worse Nodes	Important Nodes	Zero Nodes	Worse Nodes	Important Nodes	Zero Nodes	Worse Nodes	Important Nodes
mean	-0.0	0.0004	-0.0467	0.0	0.0013	-0.0711	-0.0	0.0048	-0.0341	0.0	0.0856	-0.1385	0.0	0.0061	-0.1007
std	0.0	0.0016	0.1235	0.0	0.0018	0.0205	0.0	0.0036	0.0071	0.0	0.1352	0.1498	0.0	0.0048	0.0299
min	-0.0	0.0000	-0.3654	-0.0	0.0000	-0.1089	-0.0	0.0010	-0.0492	0.0	0.0079	-0.6552	0.0	0.0000	-0.1643
25%	0.0	0.0000	0.0000	-0.0	0.0003	-0.0867	-0.0	0.0023	-0.0387	0.0	0.0191	-0.1429	0.0	0.0025	-0.1185
50%	0.0	0.0000	0.0000	0.0	0.0010	-0.0700	0.0	0.0042	-0.0322	0.0	0.0389	-0.0867	0.0	0.0050	-0.0956
75%	0.0	0.0000	0.0000	0.0	0.0013	-0.0575	0.0	0.0055	-0.0298	0.0	0.0626	-0.0683	0.0	0.0092	-0.0793
max	0.0	0.0060	0.0000	0.0	0.0071	-0.0328	0.0	0.0147	-0.0238	0.0	0.4927	-0.0608	0.0	0.0150	-0.0582

Table D.61: Average importance for each node class for the second layer of the CNN (64-filter layer)

	1			8			32			256			1024		
	Zero Nodes	Worse Nodes	Important Nodes	Zero Nodes	Worse Nodes	Important Nodes	Zero Nodes	Worse Nodes	Important Nodes	Zero Nodes	Worse Nodes	Important Nodes	Zero Nodes	Worse Nodes	Important Nodes
mean	-0.0	0.0004	-0.0467	0.0	0.0013	-0.0711	-0.0	0.0048	-0.0341	0.0	0.0856	-0.1385	0.0	0.0061	-0.1007
std	0.0	0.0016	0.1235	0.0	0.0018	0.0205	0.0	0.0036	0.0071	0.0	0.1352	0.1498	0.0	0.0048	0.0299
min	-0.0	0.0000	-0.3654	-0.0	0.0000	-0.1089	-0.0	0.0010	-0.0492	0.0	0.0079	-0.6552	0.0	0.0000	-0.1643
25%	0.0	0.0000	0.0000	-0.0	0.0003	-0.0867	-0.0	0.0023	-0.0387	0.0	0.0191	-0.1429	0.0	0.0025	-0.1185
50%	0.0	0.0000	0.0000	0.0	0.0010	-0.0700	0.0	0.0042	-0.0322	0.0	0.0389	-0.0867	0.0	0.0050	-0.0956
75%	0.0	0.0000	0.0000	0.0	0.0013	-0.0575	0.0	0.0055	-0.0298	0.0	0.0626	-0.0683	0.0	0.0092	-0.0793
max	0.0	0.0060	0.0000	0.0	0.0071	-0.0328	0.0	0.0147	-0.0238	0.0	0.4927	-0.0608	0.0	0.0150	-0.0582

Table D.62: Average importance for each node class for the third layer of the CNN (128-filter layer)

	1			8			32			256			1024		
	Zero Nodes	Worse Nodes	Important Nodes	Zero Nodes	Worse Nodes	Important Nodes	Zero Nodes	Worse Nodes	Important Nodes	Zero Nodes	Worse Nodes	Important Nodes	Zero Nodes	Worse Nodes	Important Nodes
mean	-0.0	0.0004	-0.0467	0.0	0.0013	-0.0711	-0.0	0.0048	-0.0341	0.0	0.0856	-0.1385	0.0	0.0061	-0.1007
std	0.0	0.0016	0.1235	0.0	0.0018	0.0205	0.0	0.0036	0.0071	0.0	0.1352	0.1498	0.0	0.0048	0.0299
min	-0.0	0.0000	-0.3654	-0.0	0.0000	-0.1089	-0.0	0.0010	-0.0492	0.0	0.0079	-0.6552	0.0	0.0000	-0.1643
25%	0.0	0.0000	0.0000	-0.0	0.0003	-0.0867	-0.0	0.0023	-0.0387	0.0	0.0191	-0.1429	0.0	0.0025	-0.1185
50%	0.0	0.0000	0.0000	0.0	0.0010	-0.0700	0.0	0.0042	-0.0322	0.0	0.0389	-0.0867	0.0	0.0050	-0.0956
75%	0.0	0.0000	0.0000	0.0	0.0013	-0.0575	0.0	0.0055	-0.0298	0.0	0.0626	-0.0683	0.0	0.0092	-0.0793
max	0.0	0.0060	0.0000	0.0	0.0071	-0.0328	0.0	0.0147	-0.0238	0.0	0.4927	-0.0608	0.0	0.0150	-0.0582

Table D.63: Average importance for each node class for the fourth layer of the CNN (256-filter layer)

		1			8			32			256			1024	
	Zero Nodes	Worse Nodes	Important Nodes	Zero Nodes	Worse Nodes	Important Nodes	Zero Nodes	Worse Nodes	Important Nodes	Zero Nodes	Worse Nodes	Important Nodes	Zero Nodes	Worse Nodes	Important Nodes
mean	0.0	0.0003	-0.0045	-0.0	0.0013	-0.0651	-0.0	0.0026	-0.1168	-0.0	0.0721	-0.1321	0.0	0.0040	-0.1026
std	0.0	0.0010	0.0120	0.0	0.0010	0.0378	0.0	0.0042	0.0601	0.0	0.1153	0.0741	0.0	0.0057	0.0349
min	-0.0	0.0000	-0.0377	-0.0	0.0001	-0.1547	-0.0	0.0000	-0.2707	-0.0	0.0072	-0.2734	-0.0	0.0000	-0.1467
25%	0.0	0.0000	0.0000	-0.0	0.0005	-0.0887	-0.0	0.0002	-0.1348	0.0	0.0182	-0.1723	0.0	0.0000	-0.1345
50%	0.0	0.0000	0.0000	0.0	0.0007	-0.0507	-0.0	0.0007	-0.1063	0.0	0.0331	-0.1011	0.0	0.0010	-0.1156
75%	0.0	0.0000	0.0000	0.0	0.0018	-0.0391	0.0	0.0029	-0.0723	0.0	0.0452	-0.0790	0.0	0.0061	-0.0682
max	0.0	0.0041	0.0000	0.0	0.0031	-0.0240	0.0	0.0144	-0.0520	0.0	0.3705	-0.0520	0.0	0.0213	-0.0570

Table D.64: Average importance for each node class for the fifth layer of the CNN (32-node layer)

D.3 Pre-calculated Pruning

D.3.1 MNIST

Figures - Training set

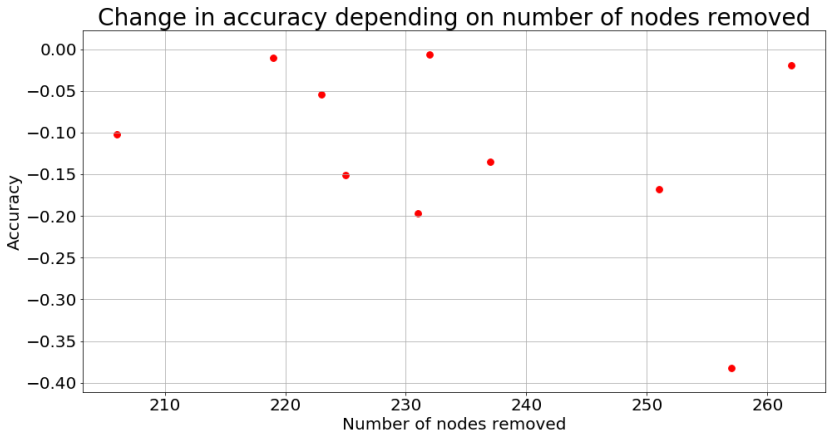

Figure D.11: Change in accuracy after pruning based on pre-calculated node importance

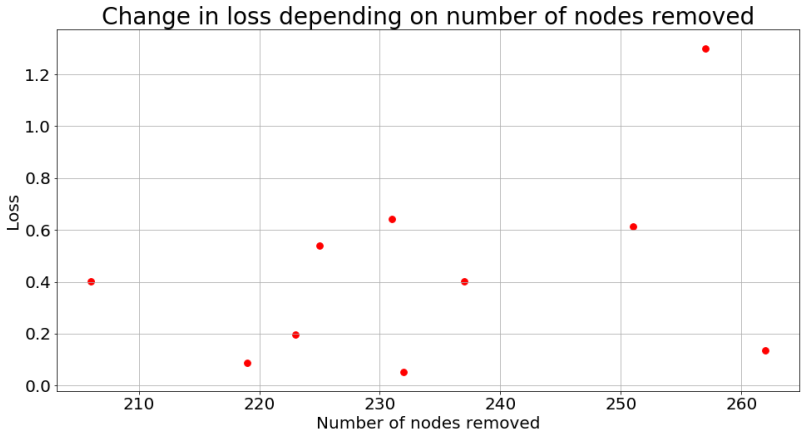

Figure D.12: Change in loss after pruning based on pre-calculated node importance

Figures - Validation set

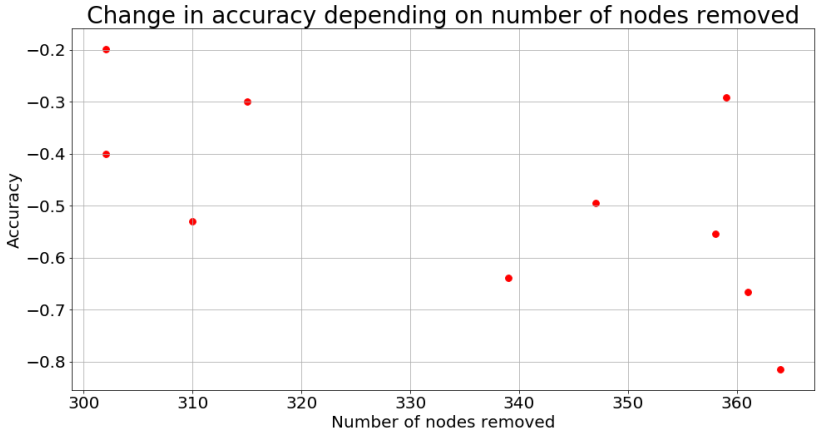

Figure D.13: Change in accuracy after pruning based on pre-calculated node importance

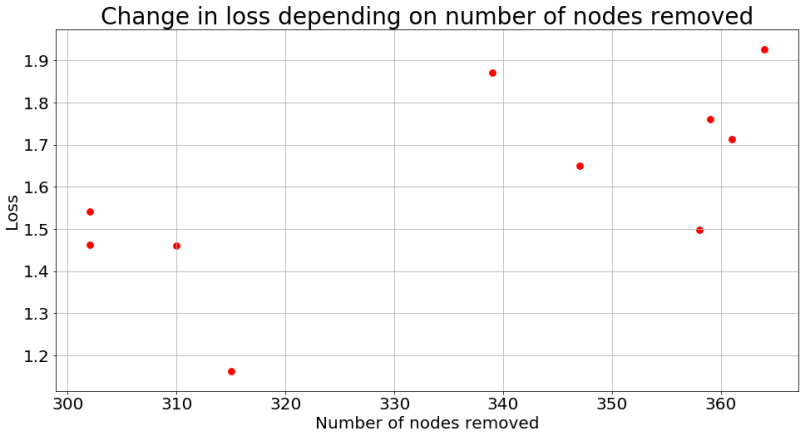

Figure D.14: Change in loss after pruning based on pre-calculated node importance

Tables - Training set

	Layer 5	Layer 4	Layer 3	Layer 2	Layer 1
mean	33.90	129.10	45.80	16.40	9.10
std	4.91	14.62	6.51	6.69	3.35
min	26.00	110.00	37.00	10.00	4.00
25%	31.25	119.50	39.50	11.25	7.50
50%	34.00	125.50	47.00	16.00	9.50
75%	37.25	132.75	50.00	18.00	10.00
max	41.00	157.00	56.00	32.00	16.00

Table D.65: Statistics of number of nodes pruned from CNNs trained on MNIST

	Change in Accuracy
mean	-0.1227
std	0.1146
min	-0.3826
25%	-0.1638
50%	-0.1187
75%	-0.0282
max	-0.0063

Table D.66: Statistics of change in accuracy of CNNs trained on MNIST

	Change in Loss
mean	0.4370
std	0.3731
min	0.0509
25%	0.1505
50%	0.4016
75%	0.5955
max	1.3008

Table D.67: Statistics of change loss of CNNs trained on MNIST

Tables - Validation set

	Layer 5	Layer 4	Layer 3	Layer 2	Layer 1
mean	41.40	179.80	67.20	29.30	18.00
std	4.17	14.25	6.89	3.02	2.26
min	35.00	158.00	59.00	24.00	16.00
25%	38.50	166.75	60.75	27.25	17.00
50%	41.00	182.50	66.50	30.00	18.00
75%	44.50	192.75	73.75	31.00	18.00
max	48.00	195.00	76.00	33.00	24.00

Table D.68: Statistics of number of nodes pruned from CNNs trained on MNIST

	Change in Accuracy
mean	-0.4888
std	0.1919
min	-0.8140
25%	-0.6169
50%	-0.5121
75%	-0.3252
max	-0.1997

Table D.69: Statistics of change in accuracy of CNNs trained on MNIST

	Change in Loss
mean	1.6048
std	0.2270
min	1.1634
25%	1.4711
50%	1.5953
75%	1.7499
max	1.9258

Table D.70: Statistics of change loss of CNNs trained on MNIST

D.3.2 Fashion MNIST

Figures - Training set

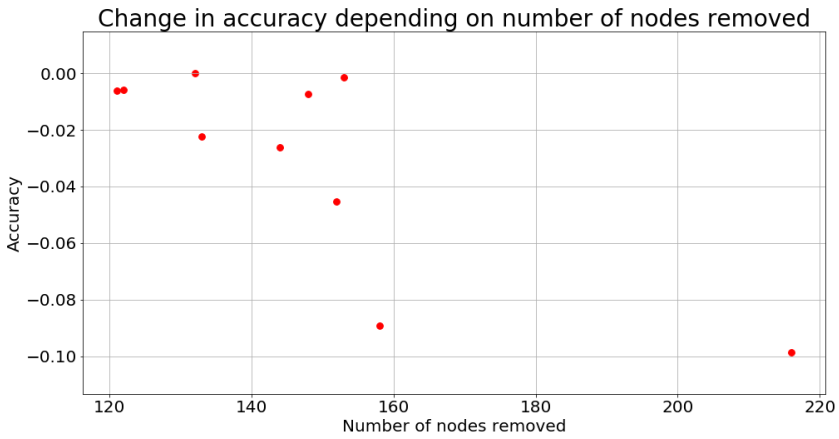

Figure D.15: Change in accuracy after pruning based on pre-calculated node importance

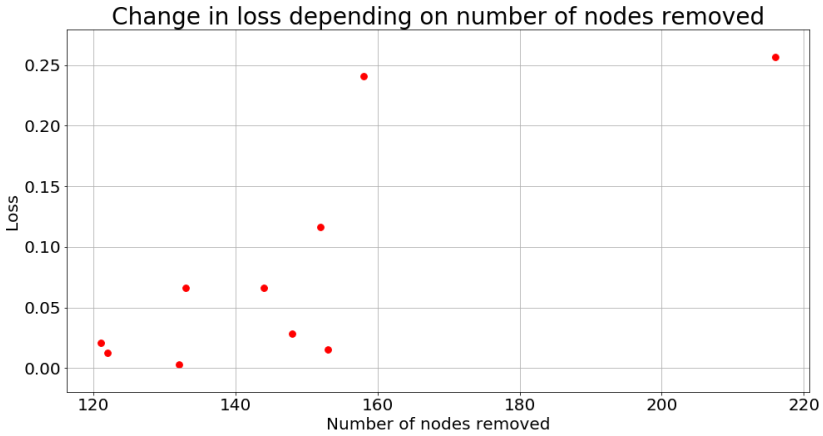

Figure D.16: Change in loss after pruning based on pre-calculated node importance

Figures - Validation set

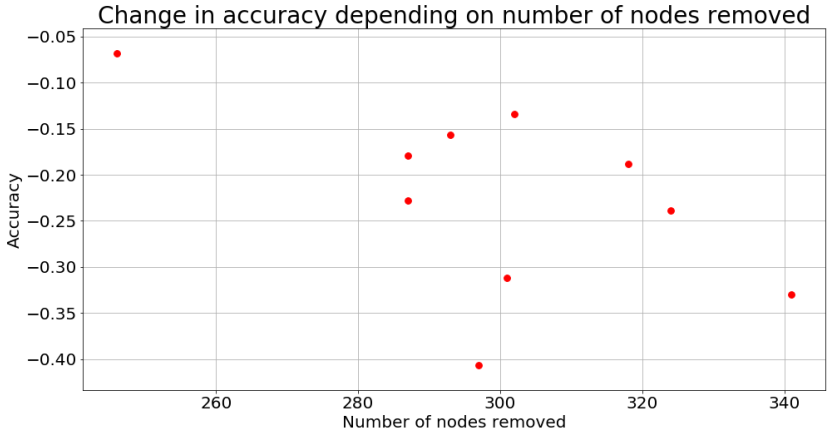

Figure D.17: Change in accuracy after pruning based on pre-calculated node importance

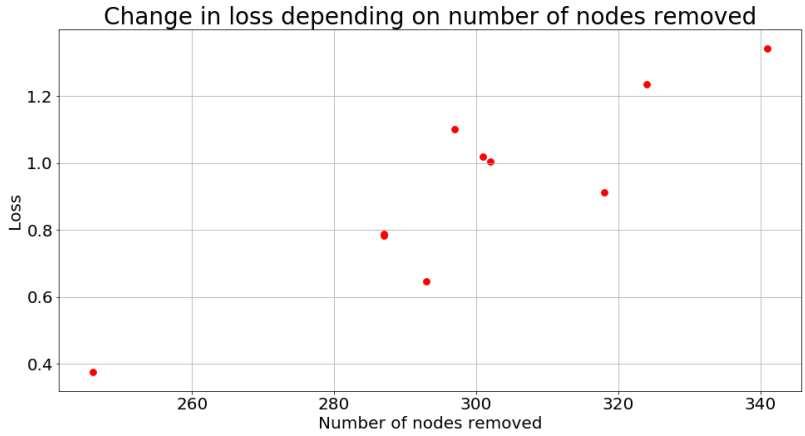

Figure D.18: Change in loss after pruning based on pre-calculated node importance

Tables - Training set

	Layer 5	Layer 4	Layer 3	Layer 2	Layer 1
mean	28.00	78.40	30.60	7.60	3.30
std	4.37	15.15	4.97	5.80	2.50
min	21.00	61.00	25.00	2.00	0.00
25%	25.00	72.25	27.50	4.25	1.50
50%	27.00	75.00	30.00	6.00	3.50
75%	31.50	80.50	31.00	7.00	4.75
max	35.00	114.00	39.00	20.00	8.00

Table D.71: Statistics of number of nodes pruned from CNNs trained on Fashion MNIST

	Change in Accuracy
mean	-0.0302
std	0.0364
min	-0.0986
25%	-0.0404
50%	-0.0148
75%	-0.0059
max	0.0000

Table D.72: Statistics of change in accuracy of CNNs trained on Fashion MNIST

	Change in Loss
mean	0.0826
std	0.0941
min	0.0027
25%	0.0168
50%	0.0470
75%	0.1040
max	0.2568

Table D.73: Statistics of change loss of CNNs trained on Fashion MNIST

Tables - Validation set

	Layer 5	Layer 4	Layer 3	Layer 2	Layer 1
mean	39.30	168.00	56.30	24.20	11.80
std	3.09	15.14	6.31	4.78	3.77
min	34.00	136.00	43.00	17.00	4.00
25%	39.25	162.00	53.75	21.50	11.00
50%	40.00	168.00	57.00	23.50	11.50
75%	40.75	176.00	60.75	26.00	14.75
max	44.00	193.00	64.00	32.00	16.00

Table D.74: Statistics of number of nodes pruned from CNNs trained on Fashion MNIST

	Change in Accuracy
mean	-0.2241
std	0.1016
min	-0.4070
25%	-0.2936
50%	-0.2081
75%	-0.1624
max	-0.0679

Table D.75: Statistics of change in accuracy of CNNs trained on Fashion MNIST

	Change in Loss
mean	0.9204
std	0.2854
min	0.3761
25%	0.7847
50%	0.9581
75%	1.0797
max	1.3406

Table D.76: Statistics of change loss of CNNs trained on Fashion MNIST

D.3.3 CIFAR-10

Figures - Training set

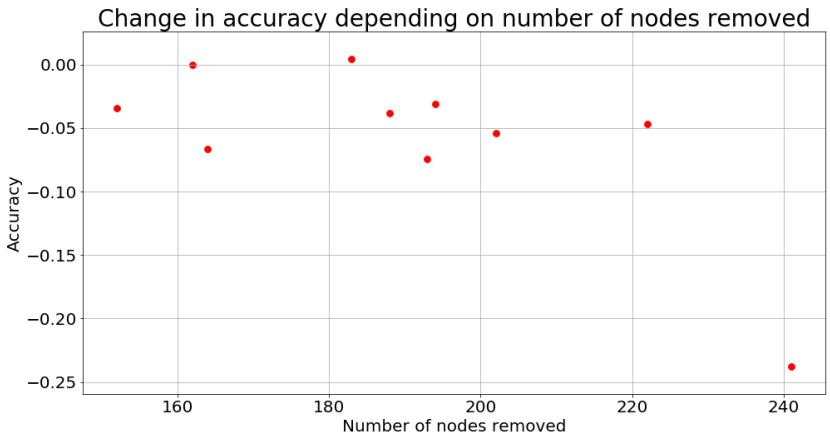

Figure D.19: Change in accuracy after pruning based on pre-calculated node importance

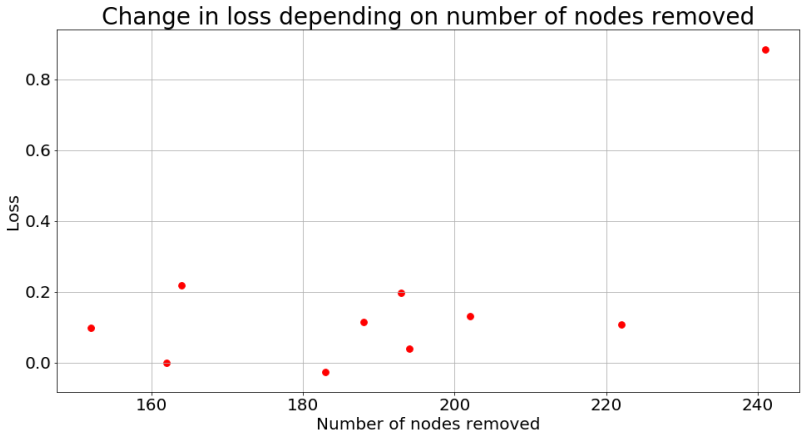

Figure D.20: Change in loss after pruning based on pre-calculated node importance

Figures - Validation set

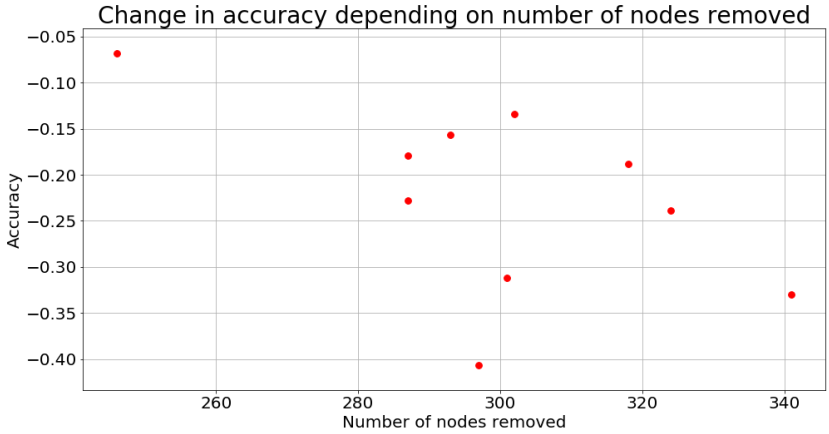

Figure D.21: Change in accuracy after pruning based on pre-calculated node importance

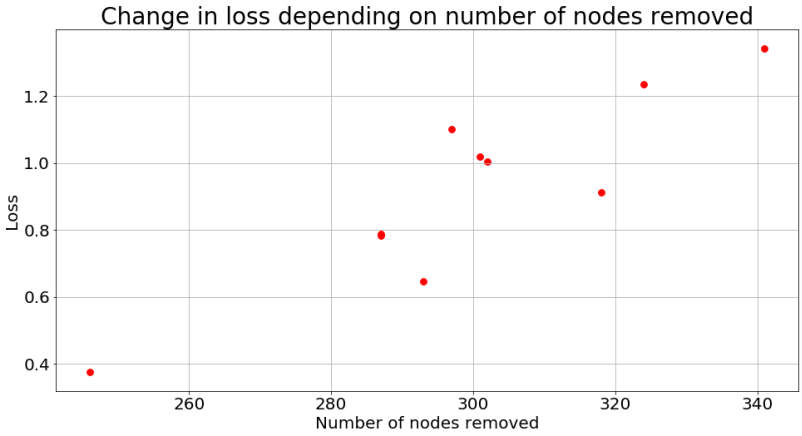

Figure D.22: Change in loss after pruning based on pre-calculated node importance

Tables - Training set

	Layer 5	Layer 4	Layer 3	Layer 2	Layer 1
mean	38.90	99.40	36.30	11.60	3.90
std	3.75	15.89	7.73	3.37	1.37
min	31.00	77.00	29.00	7.00	2.00
25%	37.25	87.25	30.25	10.00	3.25
50%	39.00	99.50	35.50	11.00	4.00
75%	41.75	103.00	37.75	12.50	4.00
max	44.00	127.00	53.00	19.00	6.00

Table D.77: Statistics of number of nodes pruned from CNNs trained on CIFAR-10

	Change in Accuracy
mean	-0.0579
std	0.0680
min	-0.2376
25%	-0.0634
50%	-0.0423
75%	-0.0321
max	0.0041

Table D.78: Statistics of change in accuracy of CNNs trained on CIFAR-10

	Change in Loss
mean	0.1762
std	0.2611
min	-0.0275
25%	0.0534
50%	0.1106
75%	0.1812
max	0.8854

Table D.79: Statistics of change loss of CNNs trained on CIFAR-10

Tables - Validation set

	Layer 5	Layer 4	Layer 3	Layer 2	Layer 1
mean	45.30	155.90	53.00	21.90	5.70
std	4.14	8.36	6.04	4.31	1.83
min	39.00	142.00	46.00	12.00	1.00
25%	42.00	150.25	49.00	20.50	5.25
50%	46.00	154.50	50.50	22.50	6.00
75%	47.00	162.50	55.50	24.00	7.00
max	53.00	168.00	65.00	28.00	7.00

Table D.80: Statistics of number of nodes pruned from CNNs trained on CIFAR-10

	Change in Accuracy
mean	-0.2505
std	0.1266
min	-0.4209
25%	-0.3670
50%	-0.2328
75%	-0.1368
max	-0.1059

Table D.81: Statistics of change in accuracy of CNNs trained on CIFAR-10

	Change in Loss
mean	0.8137
std	0.4598
min	0.2839
25%	0.4240
50%	0.7126
75%	1.1927
max	1.5416

Table D.82: Statistics of change loss of CNNs trained on CIFAR-10

D.4 Exhaustive Pruning

D.4.1 MNIST

Figures

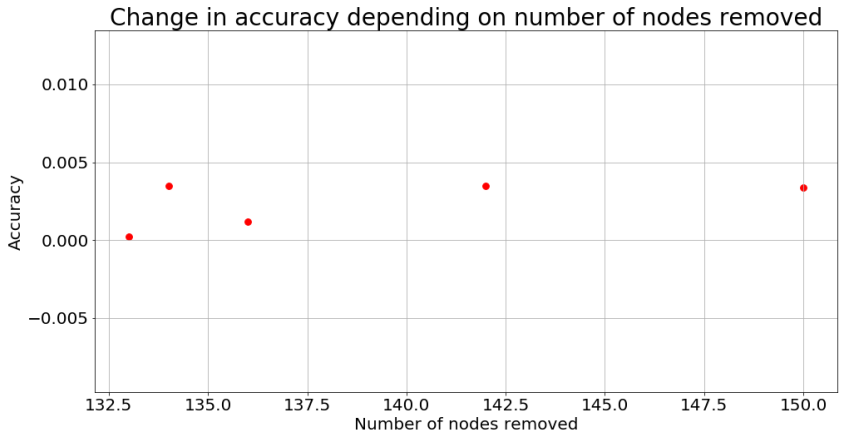

Figure D.23: Change in accuracy after pruning with exhaustive method

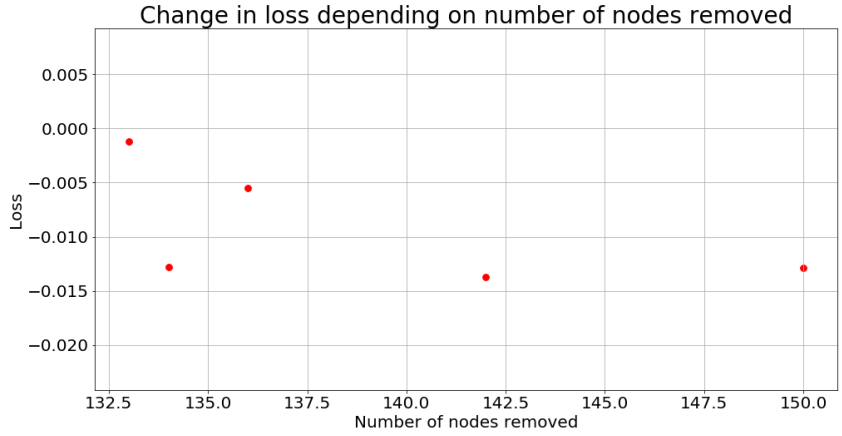

Figure D.24: Change in loss after pruning with exhaustive method

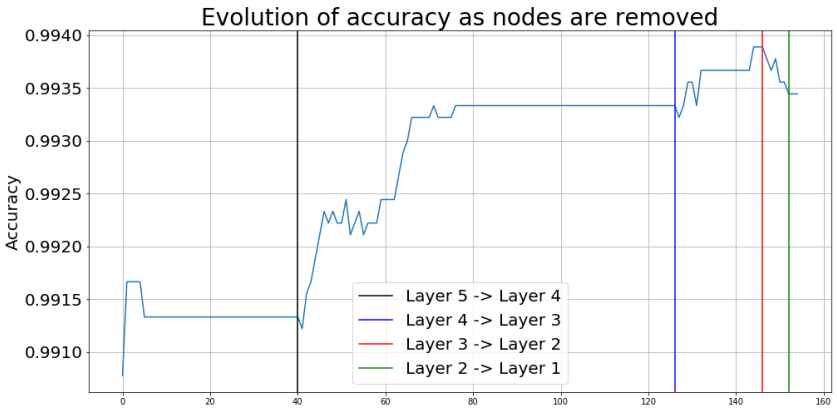

Figure D.25: Evolution of validation accuracy as a CNN is pruned

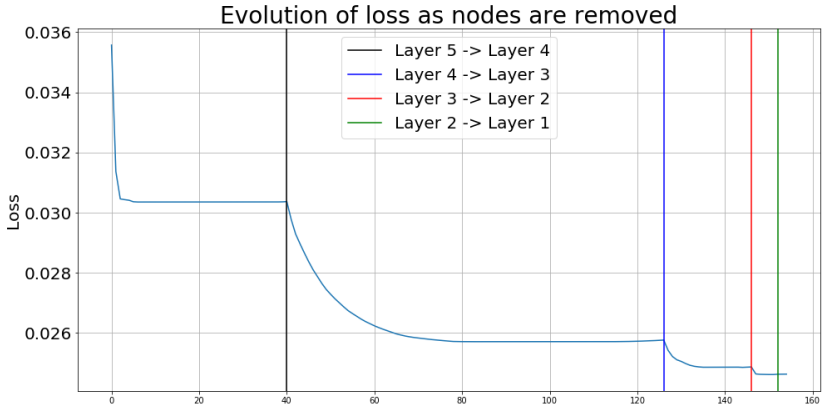

Figure D.26: Evolution of validation loss as a CNN is pruned

Tables

	Layer 5	Layer 4	Layer 3	Layer 2	Layer 1
mean	34.20	79.60	18.80	2.80	3.60
std	4.44	10.21	2.28	0.84	2.19
min	29.00	68.00	16.00	2.00	0.00
25%	30.00	70.00	17.00	2.00	4.00
50%	36.00	82.00	19.00	3.00	4.00
75%	37.00	87.00	21.00	3.00	4.00
max	39.00	91.00	21.00	4.00	6.00

Table D.83: Statistics of number of nodes pruned from CNNs trained on MNIST

	Accuracy
mean	0.0024
std	0.0016
min	0.0002
25%	0.0012
50%	0.0034
75%	0.0035
max	0.0035

Table D.84: Statistics of change accuracy of CNNs trained on MNIST

	Loss
mean	-0.0092
std	0.0056
min	-0.0137
25%	-0.0129
50%	-0.0128
75%	-0.0055
max	-0.0012

Table D.85: Statistics of change loss of CNNs trained on MNIST

D.4.2 Fashion MNIST

Figures

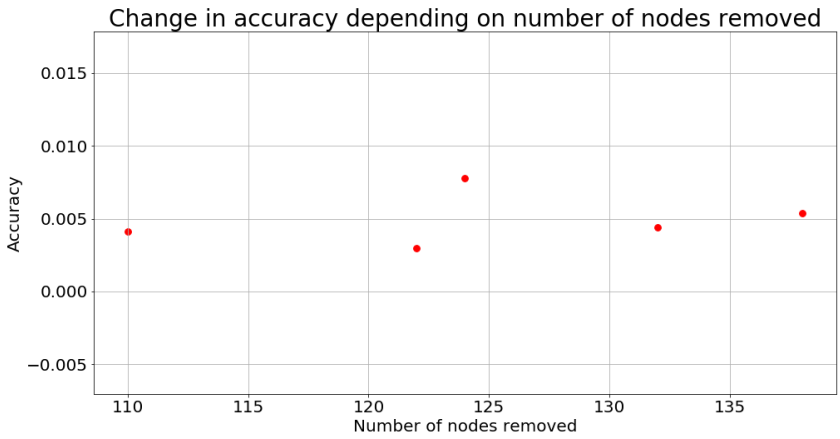

Figure D.27: Change in accuracy after pruning with exhaustive method

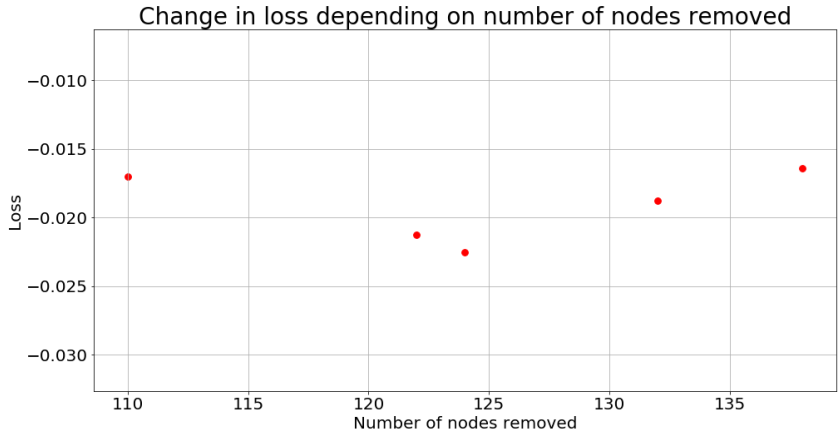

Figure D.28: Change in loss after pruning with exhaustive method

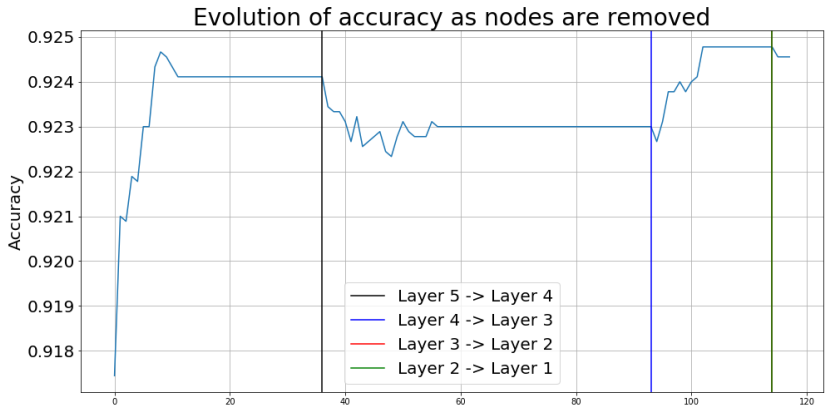

Figure D.29: Evolution of validation accuracy as a CNN is pruned

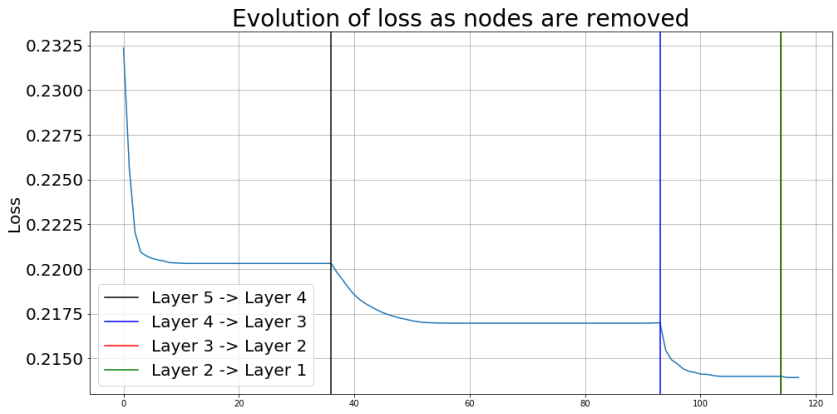

Figure D.30: Evolution of validation loss as a CNN is pruned

Tables

	Layer 5	Layer 4	Layer 3	Layer 2	Layer 1
mean	32.60	64.00	23.00	3.00	2.60
std	8.99	7.18	4.12	1.87	1.67
min	22.00	55.00	20.00	0.00	1.00
25%	24.00	60.00	20.00	3.00	1.00
50%	37.00	64.00	20.00	3.00	3.00
75%	38.00	67.00	27.00	4.00	3.00
max	42.00	74.00	28.00	5.00	5.00

Table D.86: Statistics of number of nodes pruned from CNNs trained on Fashion MNIST

	Accuracy
mean	0.0049
std	0.0018
min	0.0030
25%	0.0041
50%	0.0044
75%	0.0054
max	0.0078

Table D.87: Statistics of change accuracy of CNNs trained on Fashion MNIST

	Loss
mean	-0.0192
std	0.0027
min	-0.0225
25%	-0.0213
50%	-0.0188
75%	-0.0170
max	-0.0164

Table D.88: Statistics of change loss of CNNs trained on Fashion MNIST

D.4.3 CIFAR-10

Figures

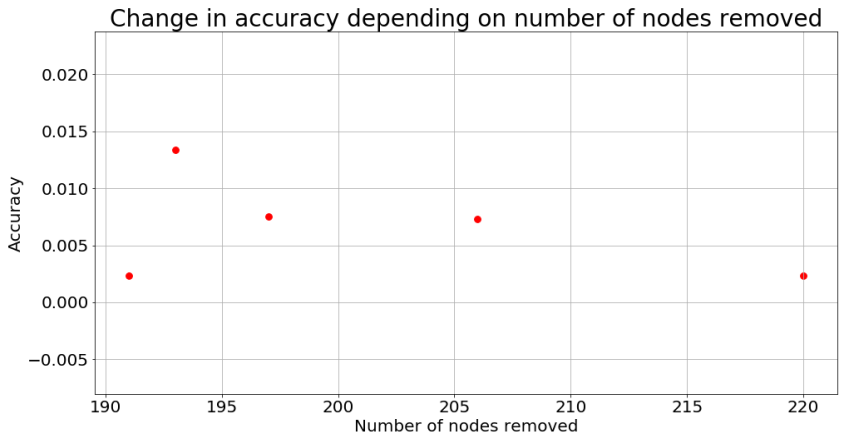

Figure D.31: Change in accuracy after pruning with exhaustive method

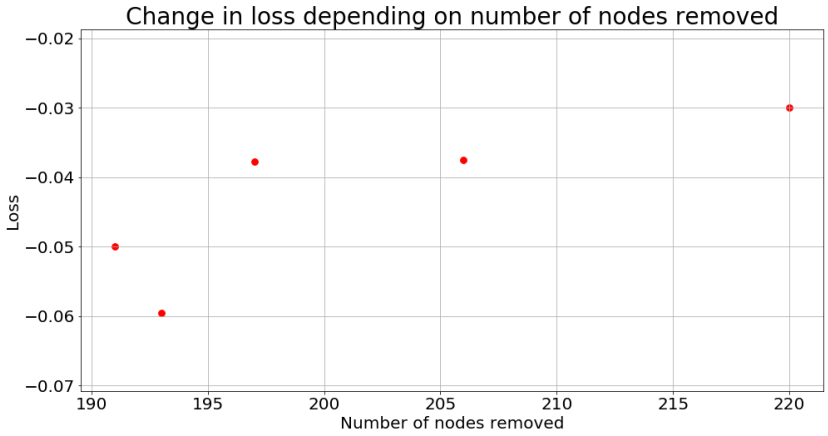

Figure D.32: Change in loss after pruning with exhaustive method

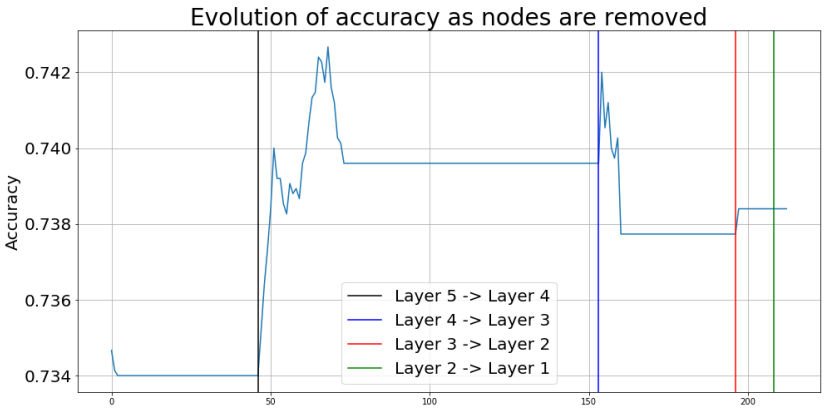

Figure D.33: Evolution of validation accuracy as a CNN is pruned

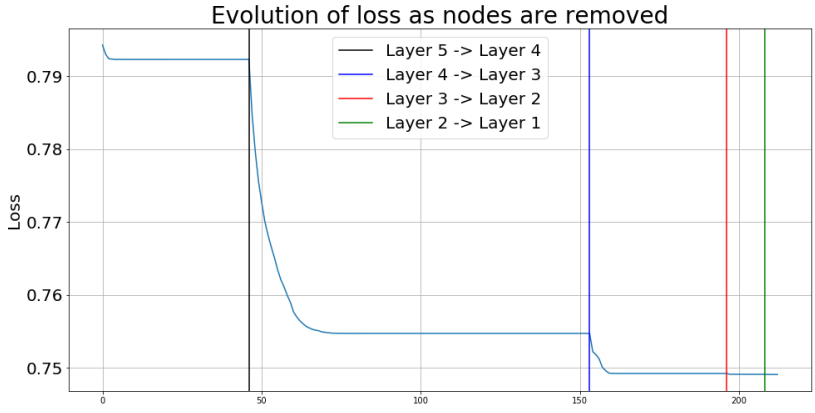

Figure D.34: Evolution of validation loss as a CNN is pruned

Tables

	Layer 5	Layer 4	Layer 3	Layer 2	Layer 1
mean	44.00	110.00	33.20	10.80	3.4
std	3.46	3.16	5.97	3.96	2.3
min	42.00	106.00	28.00	7.00	0.0
25%	42.00	108.00	29.00	8.00	3.0
50%	42.00	110.00	32.00	9.00	3.0
75%	44.00	112.00	34.00	14.00	5.0
max	50.00	114.00	43.00	16.00	6.0

Table D.89: Statistics of number of nodes pruned from CNNs trained on CIFAR-10

	Accuracy
mean	0.0066
std	0.0046
min	0.0023
25%	0.0023
50%	0.0073
75%	0.0075
max	0.0134

Table D.90: Statistics of change accuracy of CNNs trained on CIFAR-10

	Loss
mean	-0.0429
std	0.0117
min	-0.0595
25%	-0.0500
50%	-0.0377
75%	-0.0375
max	-0.0300

Table D.91: Statistics of change loss of CNNs trained on CIFAR-10

D.5 Greedy Pruning

D.5.1 MNIST

Figures - ignore cutoff: $-1e-2$

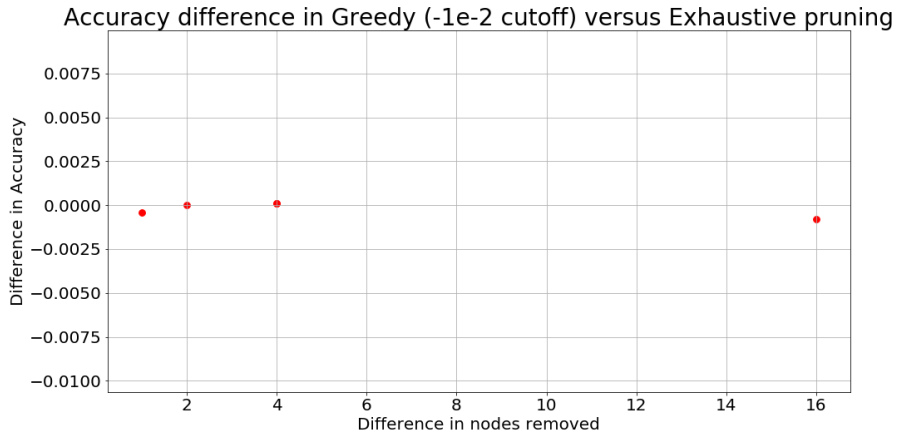

Figure D.35: Change in accuracy after pruning with greedy method and an ignore cutoff of $-1e-2$

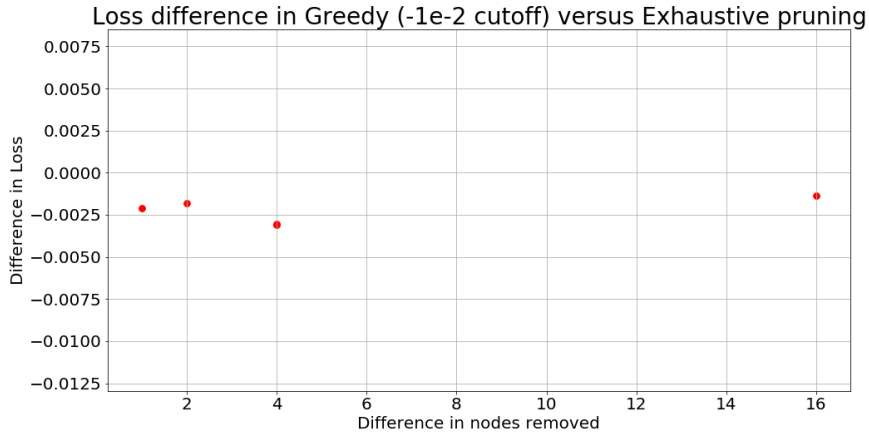

Figure D.36: Change in loss after pruning with greedy method and an ignore cutoff of $-1e-2$

Figures - ignore cutoff: $-1e-3$

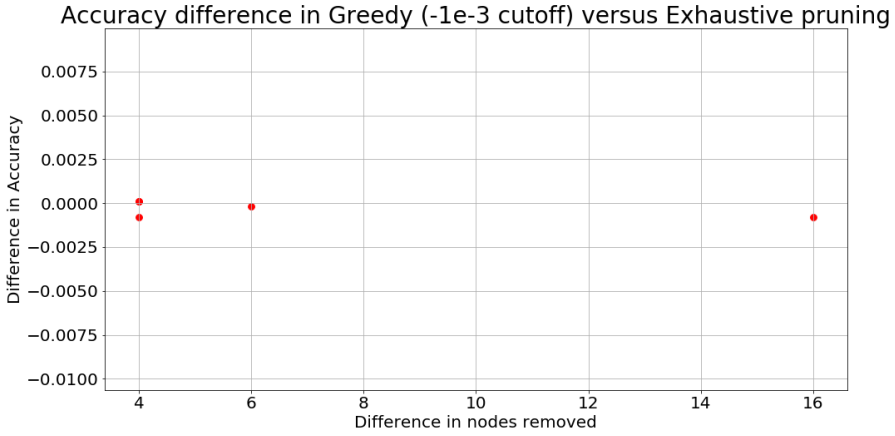

Figure D.37: Change in accuracy after pruning with greedy method and an ignore cutoff of $-1e-3$

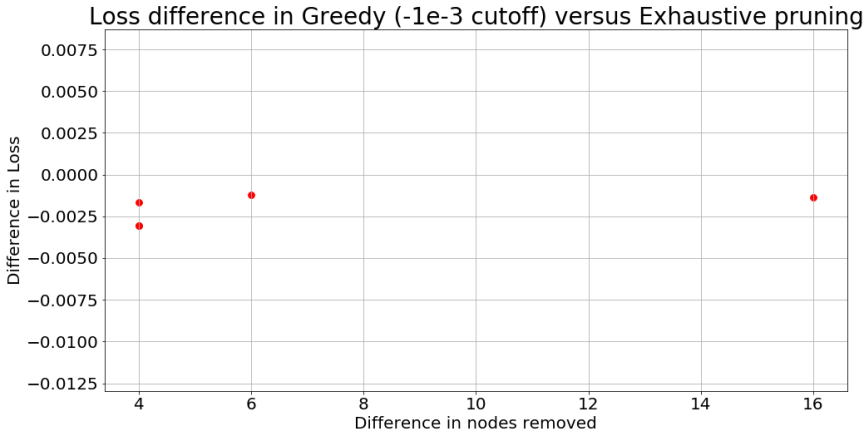

Figure D.38: Change in loss after pruning with greedy method and an ignore cutoff of $-1e-3$

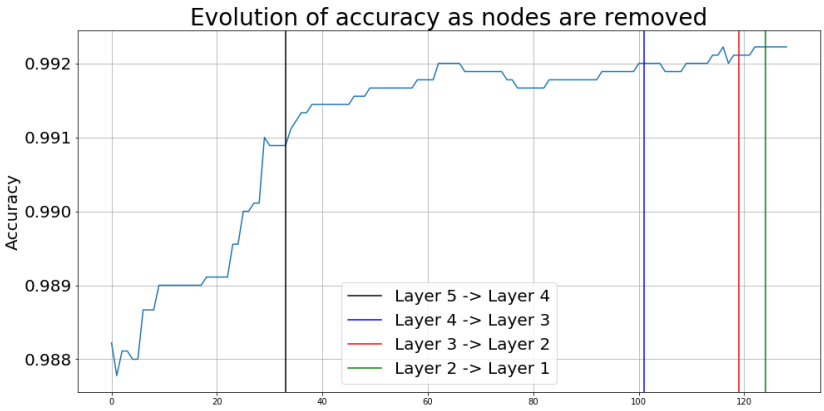

Figure D.39: Evolution of validation accuracy as a CNN is pruned

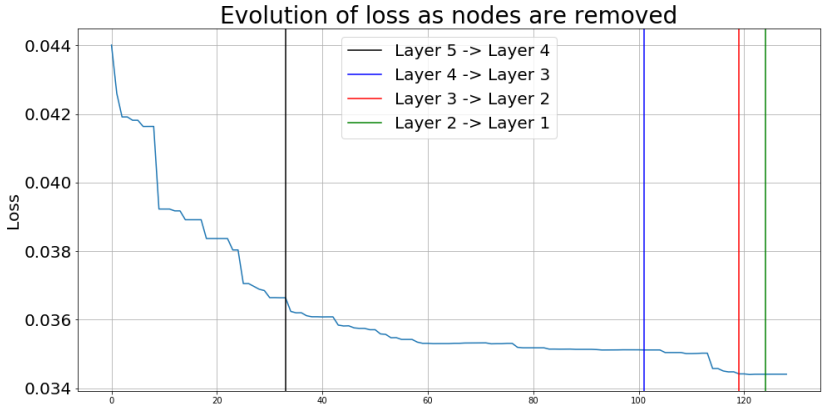

Figure D.40: Evolution of validation loss as a CNN is pruned

Tables - ignore cutoff: $-1e-2$

	Layer 5	Layer 4	Layer 3	Layer 2	Layer 1
mean	1.40	7.20	-2.20	-0.60	-0.40
std	2.07	7.33	5.81	1.14	0.55
min	-2.00	-3.00	-11.00	-2.00	-1.00
25%	1.00	3.00	-3.00	-1.00	-1.00
50%	2.00	8.00	-2.00	-1.00	0.00
75%	3.00	14.00	0.00	0.00	0.00
max	3.00	14.00	5.00	1.00	0.00

Table D.92: Statistics of the difference in the number of nodes pruned from CNNs trained on MNIST

Tables - ignore cutoff: $-1e-3$

	Layer 5	Layer 4	Layer 3	Layer 2	Layer 1
mean	1.60	8.40	-2.60	-0.40	-0.20
std	0.89	7.02	5.86	0.89	0.45
min	1.00	-3.00	-11.00	-1.00	-1.00
25%	1.00	7.00	-4.00	-1.00	0.00
50%	1.00	10.00	-3.00	-1.00	0.00
75%	2.00	14.00	0.00	0.00	0.00
max	3.00	14.00	5.00	1.00	0.00

Table D.93: Statistics of the difference in the number of nodes pruned from CNNs trained on MNIST

Tables - general

	Exhaustive	Greedy (-1e-2 cutoff)	Greedy (-1e-3 cutoff)
mean	0.0010	0.0012	0.0013
std	0.0022	0.0021	0.0025
min	-0.0007	-0.0008	-0.0008
25%	-0.0003	0.0001	-0.0001
50%	0.0004	0.0008	0.0008
75%	0.0009	0.0012	0.0012
max	0.0047	0.0047	0.0055

Table D.94: Statistics of change accuracy of CNNs trained on MNIST

	Exhaustive	Greedy (-1e-2 cutoff)	Greedy (-1e-3 cutoff)
mean	-0.0061	-0.0038	-0.0040
std	0.0074	0.0078	0.0077
min	-0.0187	-0.0169	-0.0171
25%	-0.0063	-0.0050	-0.0050
50%	-0.0021	0.0005	-0.0004
75%	-0.0016	0.0010	0.0010
max	-0.0016	0.0015	0.0015

Table D.95: Statistics of change loss of CNNs trained on MNIST

	Exhaustive	Greedy (-1e-2 cutoff)	Greedy (-1e-3 cutoff)
mean	2615.01	941.65	906.90
std	122.20	85.53	85.75
min	2440.88	838.73	805.07
25%	2589.20	883.37	836.69
50%	2618.20	931.63	911.02
75%	2644.88	1011.24	989.73
max	2781.88	1043.29	991.97

Table D.96: Time taken by different pruning algorithms to prune CNNs trained on MNIST

D.5.2 Fashion MNIST

Figures - ignore cutoff: $-1e-2$

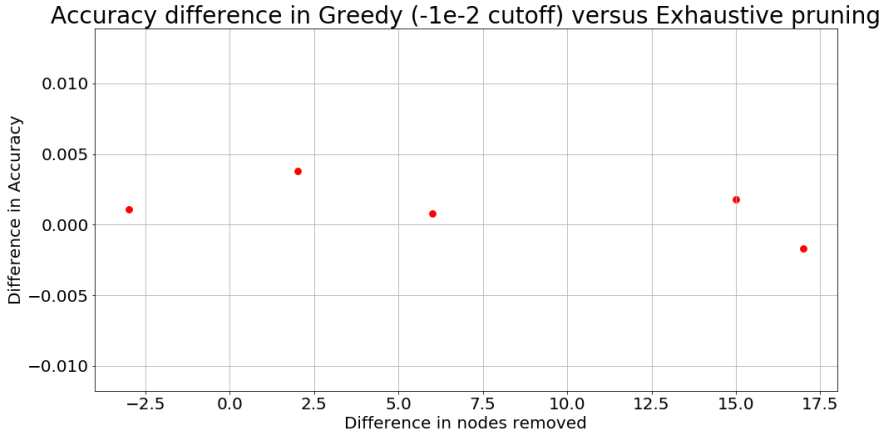

Figure D.41: Change in accuracy after pruning with greedy method and an ignore cutoff of $-1e-2$

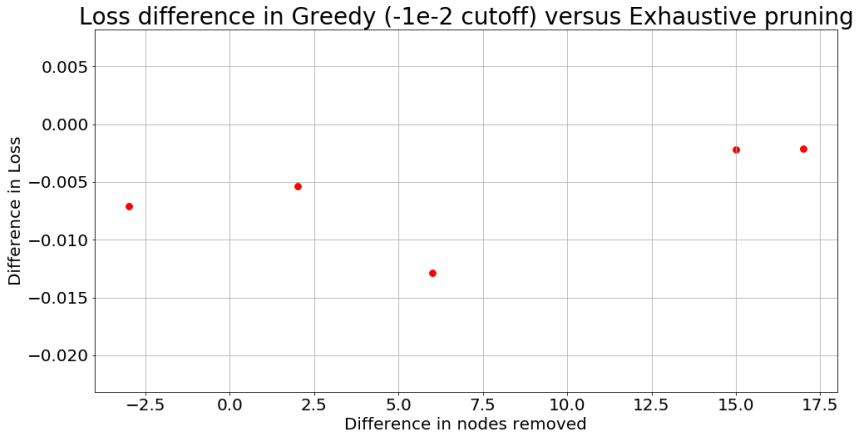

Figure D.42: Change in loss after pruning with greedy method and an ignore cutoff of $-1e-2$

Figures - ignore cutoff: $-1e-3$

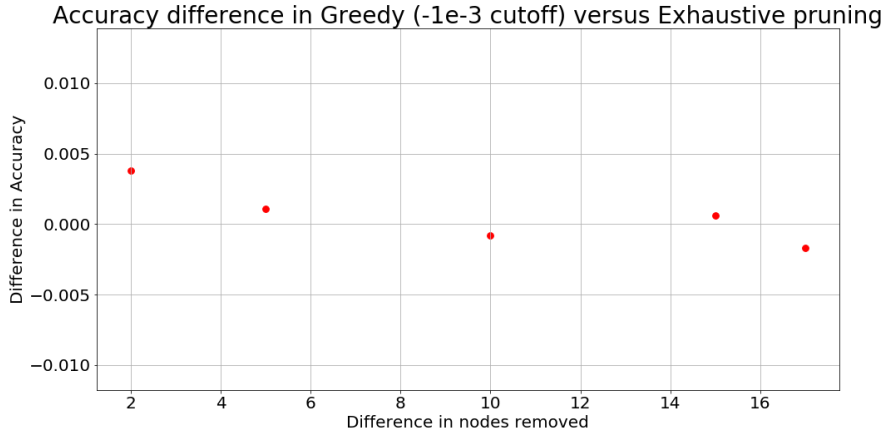

Figure D.43: Change in accuracy after pruning with greedy method and an ignore cutoff of $-1e-3$

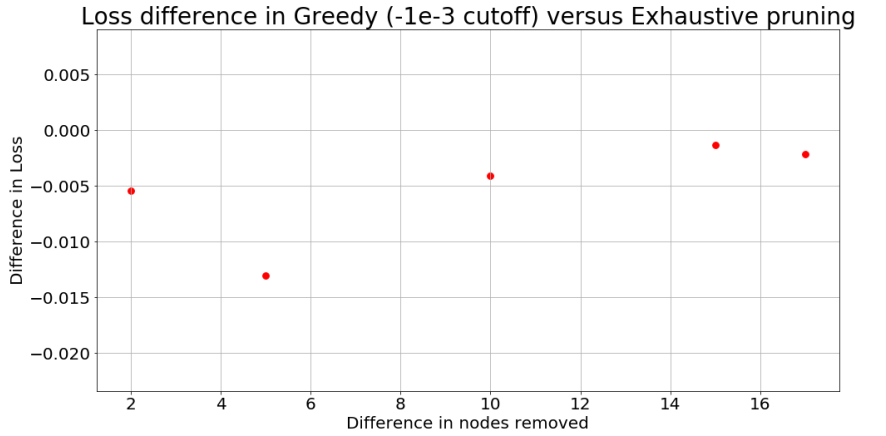

Figure D.44: Change in loss after pruning with greedy method and an ignore cutoff of $-1e-3$

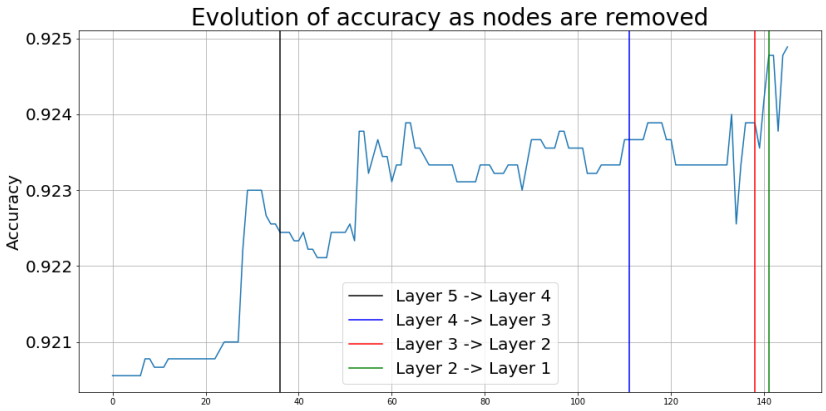

Figure D.45: Evolution of validation accuracy as a CNN is pruned

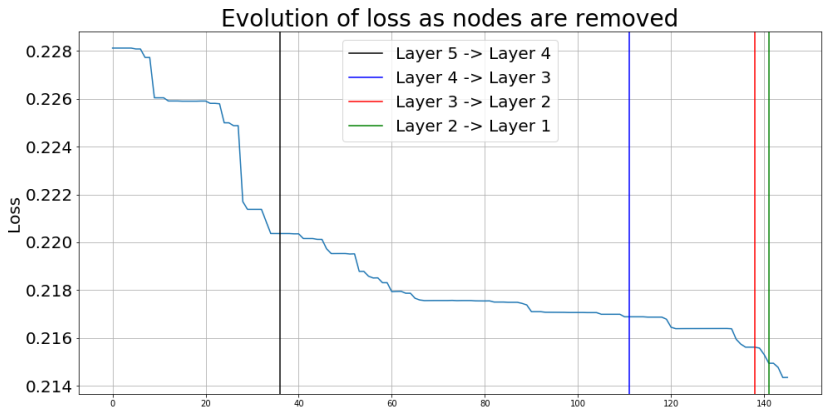

Figure D.46: Evolution of validation loss as a CNN is pruned

Tables - ignore cutoff: $-1e-2$

	Layer 5	Layer 4	Layer 3	Layer 2	Layer 1
mean	2.00	5.20	1.20	-0.60	-0.40
std	2.55	7.26	4.15	1.52	0.55
min	0.00	-2.00	-5.00	-3.00	-1.00
25%	0.00	0.00	-1.00	-1.00	-1.00
50%	1.00	2.00	3.00	0.00	0.00
75%	3.00	13.00	4.00	0.00	0.00
max	6.00	13.00	5.00	1.00	0.00

Table D.97: Statistics of the difference in the number of nodes pruned from CNNs trained on Fashion MNIST

Tables - ignore cutoff: $-1e-3$

	Layer 5	Layer 4	Layer 3	Layer 2	Layer 1
mean	1.60	7.40	1.00	-0.20	0.00
std	1.14	6.66	2.12	2.05	0.71
min	0.00	-1.00	-1.00	-2.00	-1.00
25%	1.00	2.00	-1.00	-2.00	0.00
50%	2.00	9.00	1.00	0.00	0.00
75%	2.00	13.00	2.00	0.00	0.00
max	3.00	14.00	4.00	3.00	1.00

Table D.98: Statistics of number of the difference in the nodes pruned from CNNs trained on Fashion MNIST

Tables - general

	Exhaustive	Greedy (-1e-2 cutoff)	Greedy (-1e-3 cutoff)
mean	0.0068	0.0056	0.0062
std	0.0050	0.0041	0.0037
min	0.0009	0.0026	0.0026
25%	0.0044	0.0026	0.0038
50%	0.0048	0.0037	0.0056
75%	0.0108	0.0070	0.0070
max	0.0131	0.0123	0.0120

Table D.99: Statistics of change accuracy of CNNs trained on Fashion MNIST

	Exhaustive	Greedy (-1e-2 cutoff)	Greedy (-1e-3 cutoff)
mean	-0.0265	-0.0205	-0.0213
std	0.0141	0.0098	0.0102
min	-0.0474	-0.0346	-0.0344
25%	-0.0338	-0.0267	-0.0297
50%	-0.0218	-0.0164	-0.0164
75%	-0.0158	-0.0136	-0.0145
max	-0.0134	-0.0113	-0.0113

Table D.100: Statistics of change loss of CNNs trained on Fashion MNIST

https://www.overleaf.com/project/5ed6731572d67d000198b89b

	Exhaustive	Greedy (-1e-2 cutoff)	Greedy (-1e-3 cutoff)
mean	2395.55	933.53	774.59
std	271.36	64.82	40.88
min	1963.57	847.93	736.49
25%	2385.10	900.61	743.52
50%	2412.23	935.46	759.17
75%	2516.48	963.62	801.89
max	2700.38	1020.05	831.90

Table D.101: Time taken by different pruning algorithms to prune CNNs trained on Fashion MNIST

D.5.3 CIFAR-10

Figures - ignore cutoff: $-1e-2$

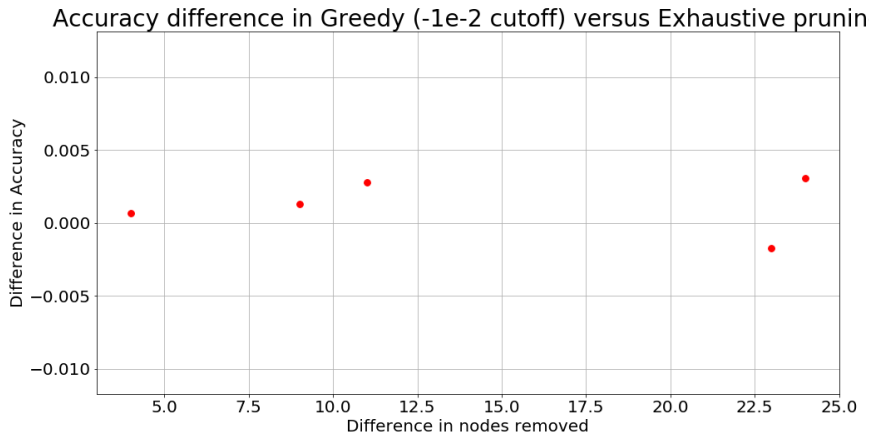

Figure D.47: Change in accuracy after pruning with greedy method and an ignore cutoff of $-1e-2$

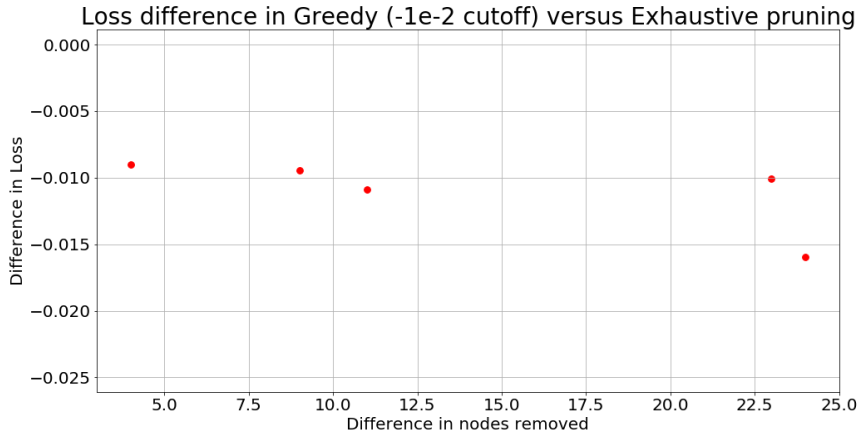

Figure D.48: Change in accuracy after pruning with greedy method and an ignore cutoff of $-1e-2$

Figures - ignore cutoff: $-1e-3$

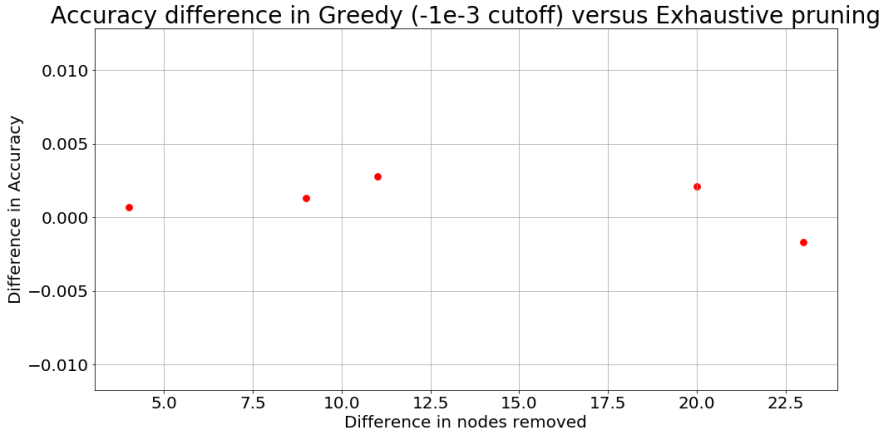

Figure D.49: Change in accuracy after pruning with greedy method and an ignore cutoff of $-1e-3$

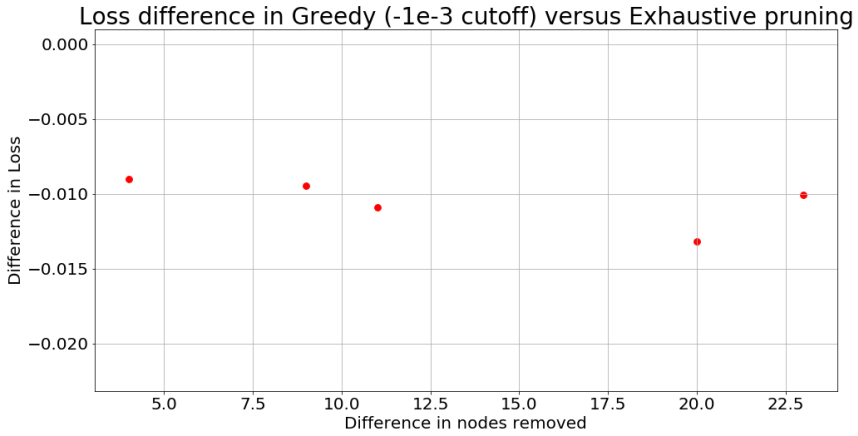

Figure D.50: Change in loss after pruning with greedy method and an ignore cutoff of $-1e-2$

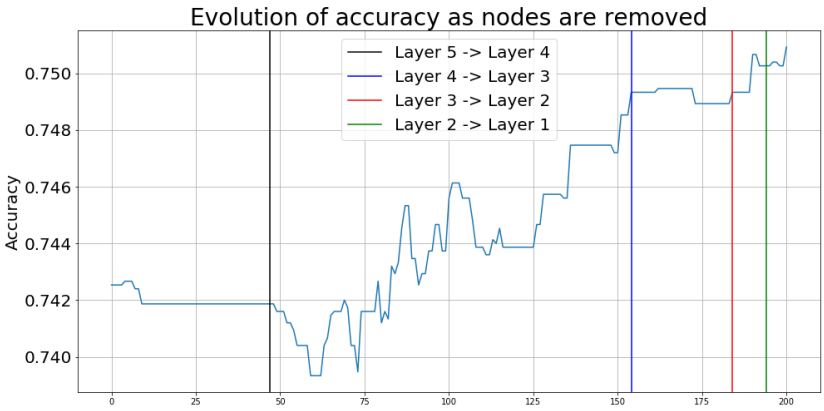

Figure D.51: Evolution of validation accuracy as a CNN is pruned

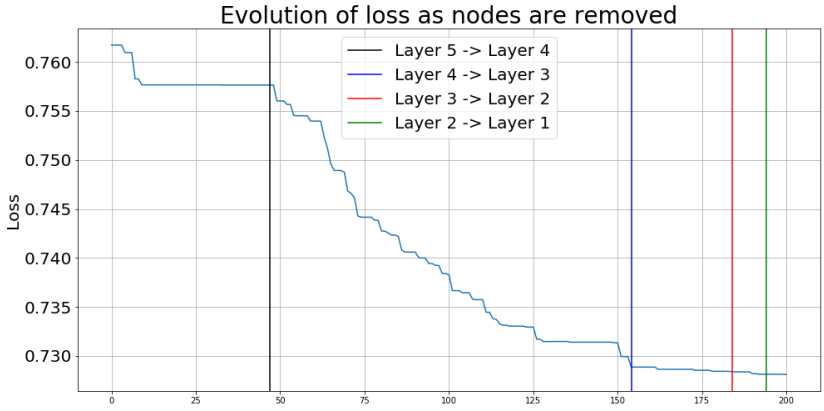

Figure D.52: Evolution of validation loss as a CNN is pruned

Tables - ignore cutoff: $-1e-2$

	Layer 5	Layer 4	Layer 3	Layer 2	Layer 1
mean	0.0	11.00	3.20	-0.20	0.20
std	0.0	5.43	3.03	0.45	0.84
min	0.0	5.00	0.00	-1.00	-1.00
25%	0.0	7.00	2.00	0.00	0.00
50%	0.0	10.00	2.00	0.00	0.00
75%	0.0	15.00	4.00	0.00	1.00
max	0.0	18.00	8.00	0.00	1.00

Table D.102: Statistics of the difference in the number of nodes pruned from CNNs trained on CIFAR-10

Tables - ignore cutoff: $-1e-3$

	Layer 5	Layer 4	Layer 3	Layer 2	Layer 1
mean	0.0	11.00	2.40	-0.20	0.20
std	0.0	5.43	1.67	0.45	0.84
min	0.0	5.00	0.00	-1.00	-1.00
25%	0.0	7.00	2.00	0.00	0.00
50%	0.0	10.00	2.00	0.00	0.00
75%	0.0	15.00	4.00	0.00	1.00
max	0.0	18.00	4.00	0.00	1.00

Table D.103: Statistics of the difference in the number of nodes pruned from CNNs trained on CIFAR-10

Tables - general

	Exhaustive	Greedy (-1e-2 cutoff)	Greedy (-1e-3 cutoff)
mean	0.0030	0.0018	0.0020
std	0.0052	0.0038	0.0038
min	-0.0041	-0.0024	-0.0024
25%	-0.0006	-0.0019	-0.0019
50%	0.0057	0.0028	0.0038
75%	0.0059	0.0050	0.0050
max	0.0083	0.0055	0.0055

Table D.104: Statistics of change accuracy of CNNs trained on CIFAR-10

	Exhaustive	Greedy (-1e-2 cutoff)	Greedy (-1e-3 cutoff)
mean	-0.0383	-0.0272	-0.0278
std	0.0115	0.0103	0.0110
min	-0.0527	-0.0396	-0.0396
25%	-0.0487	-0.0368	-0.0396
50%	-0.0317	-0.0216	-0.0216
75%	-0.0307	-0.0213	-0.0213
max	-0.0275	-0.0166	-0.0166

Table D.105: Statistics of change loss of CNNs trained on CIFAR-10

	Exhaustive	Greedy (-1e-2 cutoff)	Greedy (-1e-3 cutoff)
mean	5415.37	1443.77	473.41
std	1330.57	95.43	57.16
min	4695.06	1333.10	437.31
25%	4710.69	1403.83	439.18
50%	4932.61	1413.23	442.63
75%	4952.64	1483.17	476.33
max	7785.82	1585.55	571.58

Table D.106: Time taken by different pruning algorithms to prune CNNs trained on CIFAR-10

D.6 Effects of pruning on class accuracy

D.6.1 MNIST

Figures

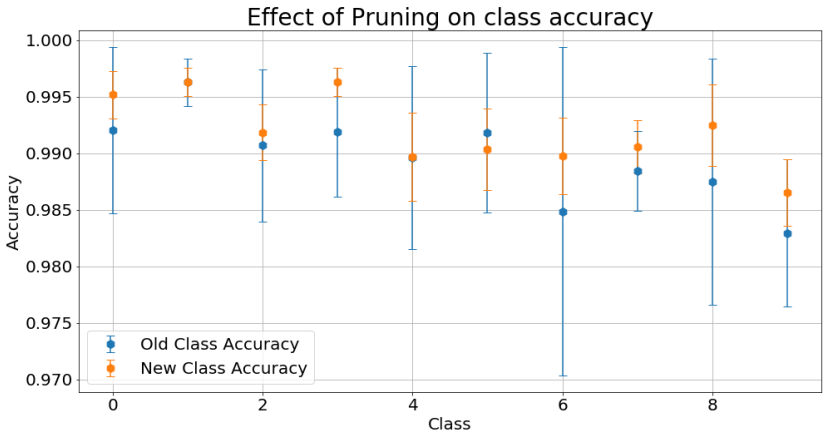

Figure D.53: Class accuracy before and after pruning CNNs trained on MNIST

Tables

	Before Pruning	After Pruning
mean	0.9897	0.9920
std	0.0020	0.0006
min	0.9869	0.9911
25%	0.9879	0.9914
50%	0.9903	0.9922
75%	0.9914	0.9925
max	0.9920	0.9926

Table D.107: Statistics on accuracy of CNNs trained on MNIST before and after pruning

	Before Pruning	After Pruning
mean	0.0377	0.0302
std	0.0085	0.0020
min	0.0269	0.0275
25%	0.0316	0.0284
50%	0.0348	0.0302
75%	0.0430	0.0316
max	0.0510	0.0332

Table D.108: Statistics on loss of CNNs trained on MNIST before and after pruning

	0	1	2	3	4	5	6	7	8	9
mean	0.9920	0.9963	0.9907	0.9919	0.9896	0.9918	0.9849	0.9884	0.9875	0.9830
std	0.0074	0.0021	0.0067	0.0057	0.0081	0.0071	0.0145	0.0035	0.0109	0.0065
min	0.9735	0.9921	0.9758	0.9762	0.9725	0.9787	0.9499	0.9815	0.9702	0.9722
25%	0.9918	0.9956	0.9884	0.9916	0.9870	0.9933	0.9872	0.9876	0.9802	0.9797
50%	0.9929	0.9969	0.9927	0.9941	0.9898	0.9944	0.9916	0.9893	0.9918	0.9841
75%	0.9967	0.9974	0.9952	0.9941	0.9967	0.9955	0.9927	0.9908	0.9946	0.9871
max	0.9990	0.9991	0.9981	0.9960	0.9980	0.9978	0.9937	0.9922	1.0000	0.9911

Table D.109: Statistics on class accuracy of CNNs trained on MNIST before pruning

	0	1	2	3	4	5	6	7	8	9
mean	0.9952	0.9963	0.9919	0.9963	0.9897	0.9904	0.9898	0.9906	0.9925	0.9865
std	0.0021	0.0012	0.0025	0.0012	0.0039	0.0036	0.0034	0.0023	0.0036	0.0029
min	0.9918	0.9947	0.9874	0.9941	0.9817	0.9832	0.9823	0.9883	0.9867	0.9822
25%	0.9939	0.9952	0.9903	0.9953	0.9880	0.9882	0.9880	0.9886	0.9908	0.9851
50%	0.9944	0.9965	0.9918	0.9970	0.9898	0.9905	0.9906	0.9898	0.9923	0.9866
75%	0.9969	0.9971	0.9937	0.9970	0.9916	0.9927	0.9916	0.9920	0.9946	0.9889
max	0.9980	0.9982	0.9952	0.9980	0.9969	0.9955	0.9937	0.9951	0.9990	0.9901

Table D.110: Statistics on class accuracy of CNNs trained on MNIST after pruning

D.6.2 Fashion MNIST

Figures

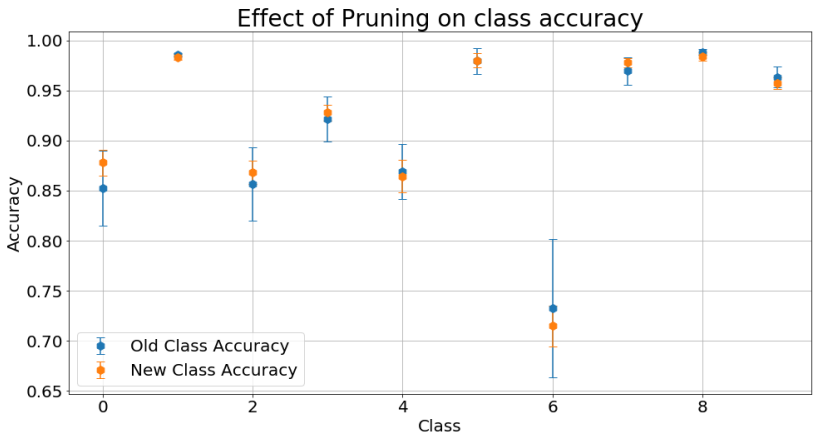

Figure D.54: Class accuracy before and after pruning CNNs trained on Fashion MNIST

Tables

	Before Pruning	After Pruning
mean	0.9118	0.9135
std	0.0027	0.0015
min	0.9072	0.9110
25%	0.9098	0.9133
50%	0.9116	0.9137
75%	0.9140	0.9141
max	0.9161	0.9161

Table D.111: Statistics on accuracy of CNNs trained on Fashion MNIST before and after pruning

	Before Pruning	After Pruning
mean	0.2509	0.2378
std	0.0135	0.0051
min	0.2340	0.2293
25%	0.2433	0.2343
50%	0.2505	0.2379
75%	0.2529	0.2398
max	0.2809	0.2456

Table D.112: Statistics on loss of CNNs trained on Fashion MNIST before and after pruning

	0	1	2	3	4	5	6	7	8	9
mean	0.8525	0.9856	0.8566	0.9213	0.8691	0.9794	0.7325	0.9695	0.9882	0.9633
std	0.0371	0.0019	0.0368	0.0227	0.0277	0.0128	0.0691	0.0136	0.0030	0.0106
min	0.8150	0.9830	0.8000	0.8800	0.8040	0.9510	0.6080	0.9420	0.9840	0.9490
25%	0.8280	0.9842	0.8292	0.9113	0.8608	0.9755	0.7005	0.9640	0.9865	0.9555
50%	0.8385	0.9850	0.8510	0.9265	0.8710	0.9825	0.7655	0.9670	0.9880	0.9635
75%	0.8742	0.9868	0.8878	0.9380	0.8865	0.9875	0.7852	0.9788	0.9888	0.9728
max	0.9200	0.9890	0.9080	0.9500	0.9010	0.9940	0.7940	0.9900	0.9940	0.9780

Table D.113: Statistics on class accuracy of CNNs trained on Fashion MNIST before pruning

	0	1	2	3	4	5	6	7	8	9
mean	0.8780	0.9834	0.8683	0.9279	0.8642	0.9800	0.7149	0.9777	0.9838	0.9571
std	0.0130	0.0033	0.0119	0.0080	0.0162	0.0070	0.0204	0.0047	0.0039	0.0058
min	0.8560	0.9780	0.8450	0.9110	0.8420	0.9670	0.6930	0.9700	0.9750	0.9440
25%	0.8675	0.9820	0.8622	0.9235	0.8522	0.9775	0.7010	0.9745	0.9822	0.9555
50%	0.8785	0.9835	0.8695	0.9285	0.8640	0.9815	0.7085	0.9780	0.9845	0.9570
75%	0.8882	0.9858	0.8760	0.9332	0.8712	0.9855	0.7315	0.9788	0.9860	0.9590
max	0.8930	0.9880	0.8840	0.9380	0.8980	0.9870	0.7510	0.9850	0.9890	0.9660

Table D.114: Statistics on class accuracy of CNNs trained on Fashion MNIST after pruning

D.6.3 CIFAR-10

Tables

	Before Pruning	After Pruning
mean	0.7323	0.7328
std	0.0064	0.0058
min	0.7250	0.7256
25%	0.7274	0.7279
50%	0.7314	0.7326
75%	0.7344	0.7363
max	0.7455	0.7430

Table D.115: Statistics on accuracy of CNNs trained on CIFAR-10 before and after pruning

	Before Pruning	After Pruning
mean	0.8011	0.7746
std	0.0238	0.0192
min	0.7598	0.7379
25%	0.7953	0.7645
50%	0.8076	0.7835
75%	0.8159	0.7855
max	0.8300	0.7946

Table D.116: Statistics on loss of CNNs trained on CIFAR-10 before and after pruning

	0	1	2	3	4	5	6	7	8	9
mean	0.7869	0.8486	0.6394	0.5000	0.6889	0.6336	0.8092	0.7735	0.8242	0.8190
std	0.0503	0.0305	0.0597	0.0779	0.0625	0.0848	0.0510	0.0414	0.0374	0.0458
min	0.7100	0.7910	0.5490	0.3640	0.5670	0.4340	0.7210	0.7160	0.7510	0.7150
25%	0.7510	0.8322	0.6132	0.4655	0.6532	0.5975	0.7788	0.7442	0.8148	0.8062
50%	0.7905	0.8485	0.6430	0.4950	0.7050	0.6655	0.8000	0.7660	0.8340	0.8215
75%	0.8080	0.8715	0.6772	0.5490	0.7238	0.6800	0.8530	0.8008	0.8460	0.8350
max	0.8690	0.8940	0.7430	0.6100	0.7920	0.7330	0.8780	0.8450	0.8640	0.8870

Table D.117: Statistics on class accuracy of CNNs trained on CIFAR-10 before pruning

	0	1	2	3	4	5	6	7	8	9
mean	0.7719	0.8554	0.5907	0.5261	0.7076	0.6425	0.8199	0.7739	0.8361	0.8041
std	0.0196	0.0185	0.0388	0.0182	0.0200	0.0353	0.0198	0.0200	0.0108	0.0136
min	0.7380	0.8300	0.5130	0.4990	0.6700	0.5650	0.7870	0.7490	0.8150	0.7830
25%	0.7662	0.8382	0.5705	0.5155	0.6975	0.6250	0.8055	0.7557	0.8320	0.7972
50%	0.7715	0.8605	0.5945	0.5225	0.7135	0.6530	0.8215	0.7730	0.8385	0.8030
75%	0.7858	0.8660	0.6165	0.5357	0.7193	0.6607	0.8372	0.7877	0.8440	0.8128
max	0.7990	0.8810	0.6450	0.5640	0.7390	0.6910	0.8450	0.8070	0.8500	0.8280

Table D.118: Statistics on class accuracy of CNNs trained on CIFAR-10 after pruning

D.7 Iterative Weight Initialization

D.7.1 MNIST

Tables

	Unoptimized Weights	Optimized Weights
mean	0.9906	0.9899
std	0.0012	0.0014
min	0.9889	0.9881
25%	0.9896	0.9885
50%	0.9909	0.9901
75%	0.9909	0.9907
max	0.9927	0.9918

Table D.119: Statistics of accuracy of CNNs trained on MNIST

	Unoptimized Weights	Optimized Weights
mean	0.0317	0.0372
std	0.0031	0.0055
min	0.0263	0.0279
25%	0.0297	0.0333
50%	0.0321	0.0372
75%	0.0336	0.0412
max	0.0371	0.0459

Table D.120: Statistics of loss of CNNs trained on MNIST

D.7.2 Fashion MNIST

Tables

	Unoptimized Weights	Optimized Weights
mean	0.9110	0.9110
std	0.0030	0.0028
min	0.9050	0.9069
25%	0.9089	0.9092
50%	0.9112	0.9107
75%	0.9136	0.9125
max	0.9143	0.9168

Table D.121: Statistics of accuracy of CNNs trained on Fashion MNIST

	Unoptimized Weights	Optimized Weights
mean	0.2559	0.2556
std	0.0079	0.0064
min	0.2438	0.2466
25%	0.2498	0.2505
50%	0.2552	0.2567
75%	0.2635	0.2586
max	0.2663	0.2672

Table D.122: Statistics of loss of CNNs trained on Fashion MNIST

Bibliography

[1] Richard Bellman. 'Dynamic programming'. In: *Science* 153.3731 (1966), pp. 34–37.

[2] Douglas E Comer et al. 'Computing as a discipline'. In: *Communications of the ACM* 32.1 (1989), pp. 9–23.

[3] George Cybenko. 'Approximation by superpositions of a sigmoidal function'. In: *Mathematics of control, signals and systems* 2.4 (1989), pp. 303–314.

[4] John Duchi, Elad Hazan and Yoram Singer. 'Adaptive subgradient methods for online learning and stochastic optimization.' In: *Journal of machine learning research* 12.7 (2011).

[5] Xavier Glorot and Yoshua Bengio. 'Understanding the difficulty of training deep feedforward neural networks'. In: ed. by Yee Whye Teh and Mike Titterington. Vol. 9. Proceedings of Machine Learning Research. Chia Laguna Resort, Sardinia, Italy: JMLR Workshop and Conference Proceedings, 13–15 May 2010, pp. 249–256. URL: http://proceedings.mlr.press/v9/glorot10a.html.

[6] Xavier Glorot, Antoine Bordes and Yoshua Bengio. 'Deep sparse rectifier neural networks'. In: *Proceedings of the fourteenth international conference on artificial intelligence and statistics*. 2011, pp. 315–323.

[7] Richard HR Hahnloser et al. 'Digital selection and analogue amplification coexist in a cortex-inspired silicon circuit'. In: *Nature* 405.6789 (2000), pp. 947–951.

[8] Song Han, Huizi Mao and William J Dally. 'Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding'. In: *arXiv preprint arXiv:1510.00149* (2015).

[9] Kaiming He et al. 'Deep residual learning for image recognition'. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 770–778.

[10] Geoffrey E Hinton et al. 'Improving neural networks by preventing co-adaptation of feature detectors'. In: *arXiv preprint arXiv:1207.0580* (2012).

[11] Diederik P. Kingma and Jimmy Ba. *Adam: A Method for Stochastic Optimization*. 2014. arXiv: 1412.6980 [cs.LG].

[12] Alex Krizhevsky, Geoffrey Hinton et al. 'Learning multiple layers of features from tiny images'. In: (2009).

[13] Y. Lecun et al. 'Gradient-based learning applied to document recognition'. In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324. DOI: 10.1109/5.726791.

[14] Yann A LeCun et al. 'Efficient backprop'. In: *Neural networks: Tricks of the trade*. Springer, 2012, pp. 9–48.

[15] Martín Abadi et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. 2015. URL: https://www.tensorflow.org/.

[16] Warren S McClulloch and Walter Pitts. 'A logical calculus of the ideas immanent in neurons activity'. In: *Bulletin of mathematical biophysics* 5.115-133 (1943), p. 10.

[17] Marvin Minsky and Seymour Papert. 'An introduction to computational geometry'. In: *Cambridge tiass., HIT* (1969).

[18] Pavlo Molchanov et al. 'Pruning convolutional neural networks for resource efficient inference'. In: *arXiv preprint arXiv:1611.06440* (2016).

[19] Konstantin Pogorelov et al. 'KVASIR: A Multi-Class Image Dataset for Computer Aided Gastrointestinal Disease Detection'. In: *Proceedings of the 8th ACM on Multimedia Systems Conference*. MMSys'17. Taipei, Taiwan: ACM, 2017, pp. 164–169. ISBN: 978-1-4503-5002-0. DOI: 10.1145/3083187.3083212. URL: http://doi.acm.org/10.1145/3083187.3083212.

[20] Herbert Robbins and Sutton Monro. 'A stochastic approximation method'. In: *The annals of mathematical statistics* (1951), pp. 400–407.

[21] Frank Rosenblatt. 'The perceptron: a probabilistic model for information storage and organization in the brain.' In: *Psychological review* 65.6 (1958), p. 386.

[22] David E Rumelhart, Geoffrey E Hinton and Ronald J Williams. 'Learning representations by back-propagating errors'. In: *nature* 323.6088 (1986), pp. 533–536.

[23] Olga Russakovsky et al. 'ImageNet Large Scale Visual Recognition Challenge'. In: *International Journal of Computer Vision (IJCV)* 115.3 (2015), pp. 211–252. DOI: 10.1007/s11263-015-0816-y.

[24] Karen Simonyan and Andrew Zisserman. 'Very deep convolutional networks for large-scale image recognition'. In: *arXiv preprint arXiv:1409.1556* (2014).

[25] Irwin Sobel and Gary Feldman. 'A 3x3 isotropic gradient operator for image processing'. In: *a talk at the Stanford Artificial Project in* (1968), pp. 271–272.

[26] Nitish Srivastava et al. 'Dropout: a simple way to prevent neural networks from overfitting'. In: *The journal of machine learning research* 15.1 (2014), pp. 1929–1958.

[27] Christian Szegedy et al. 'Going deeper with convolutions'. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2015, pp. 1–9.

[28] Han Xiao, Kashif Rasul and Roland Vollgraf. *Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms*. 2017. arXiv: 1708.07747 [cs.LG].