Exploring Redundancy in Neural Networks: Pruning by Genetic Algorithm & Filter Energy

by

Saša Janjić

A thesis submitted to
The Faculty of Graduate Studies of
The University of Manitoba
in partial fulfillment of the requirements
of the degree of

Master of Science

Department of Computer Science
The University of Manitoba
Winnipeg, Manitoba, Canada
August 2018

Thesis advisor Author

Parimala Thulasiraman

Neil Bruce **Saša Janjić**

**Exploring Redundancy in Neural Networks:
Pruning by Genetic Algorithm & Filter Energy**

Abstract

Much work has been done on making convolutional models larger and more robust, but recent works have shown there is significant redundancy, suggesting that they are vastly more complex than necessary. The goal of this thesis is to explore the degree to which the representational redundancy can be reduced. I contribute a weight pruning genetic algorithm, an energy-based filter pruning algorithm, and provide insights on model compression and structure. Evolved weight pruning of MNIST trained multilayer perceptrons and convolutional networks showed that in some cases 72.4% and 89.6% of layer parameters can be pruned without retraining, and yield improvements in test set accuracy. Energy-based filter pruning showed that ImageNet trained VGG and ResNet models also exhibit significant redundancy, with VGG layers incurring an average 3.2% loss in accuracy after 9.83% compression, and ResNet layers incurring an 3.30% loss after 87.94% compression for over one–third of the layers.

Contents

List of Figures

List of Tables

List of Publications

- Redundancy in Convolutional Neural Networks: Insights on Model Compression and Structure. 2018 International Joint Conference on Neural Networks (IJCNN), Rio de Janeiro, Brazil, July 8-13, 2018, pp. 3601–3608.

Acknowledgments

I would like to begin by expressing my gratitude to my patient advisors for enabling my pursuit of this degree and the self-improvement it necessitated. To my parents and my partner, a heart-felt thank you for keeping the pressure on, and supporting me along the way. Finally, a special thank you to my friend Joe Tang for taking the time to go through my raw writing with a fine-toothed comb and providing precise, constructive feedback and corrections.

This thesis is dedicated to those who stressed or were stressed by me.

Chapter 1

Introduction

There has been a clear trend towards networks of complexity and efficiency, across an ever expanding field of applications [LeCun et al., 2015]. The reason for the trend is that these larger, more dense networks have higher accuracy than prior, more shallow models. To understand why deep, dense networks are more successful at learning, it is important to understand how error can be measured.

The probability of error in a given network can be measured by Minimum Error Entropy [de Sá et al., 2013, fig 6.23]:

$$\hat{f}\left(e_k\right) = \frac{1}{nh}\sum_{i=1}^{n} K\left(\frac{e_k - e_i}{h}\right) \tag{1.1}$$

In Equation 1.1, $\{n,\ h,\ K,\ e_k,\ e_i\}$ represent the number of neurons in the network, bandwidth, kernel, output error, and input error respectively. From this, we can see that the *minimum error entropy* is inversely proportional to the number of neurons and the network bandwidth. As a result, the probability of error is inversely proportional to the number of neurons and bandwidth. Therefore, a greater number of highly connected neurons gives lower error as the error gradient has more dimensions to descend across, results in faster learning, if provided sufficient data. However, as the number of weighted connections in the model increases, so does the likelihood of overfitting the data, hurting generalizability of the model.

This trend has given rise to many very deep, dense networks, especially in areas of image processing and general classification tasks. Furthermore, the necessity of being able to properly utilize this ever increasing density has brought about more effective and sophisticated regularization techniques and initialization strategies.

A recent trend has been the deployment of neural networks on mobile devices for tasks such as image processing and speech recognition [Howard et al., 2017; Wu et al., 2016; Lei et al., 2013], where a balance between accuracy and portability is required. As such, while greater accuracy can be achieved by larger networks, they become less and less accessible due to device limitations in terms of space, computational load, energy usage, and latency [Han et al., 2015b; Howard et al., 2017]. There are currently a multitude of competing strategies available to help inform the selection of good starting parameters, initialization techniques, and regularization techniques to encourage optimal use of available neurons, increase the rate of convergence, and combat overfitting [Srivastava et al., 2014; Cohen et al., 2016; LeCun et al., 2015]; all working to achieve greater utilization of learned connections, commonly referred to as *parameters*.

Convolutional Neural Networks (CNN) have become immensely popular for image and pattern recognition tasks, catalyzed by several highly successful works leading to widespread adoption and interest [LeCun et al., 1998a; Krizhevsky et al., 2012;

LeCun et al., 2015]. These networks work by applying learned filters over their input plane rather than having fixed weights for each input, vastly reducing the number of necessary parameters. At the core of many CNN are Rectified Linear Units (ReLU), introduced by Hahnloser et al. [2000], which are currently the most popular activation function [LeCun et al., 2015]. This is due to ReLU's similarities to biological neurons, relatively low compute cost, and insusceptibility to the nefarious *vanishing gradients* problem, where the error gradient diminishes to the point where the parameters can no longer be updated, that other activation function are susceptible to [Glorot et al., 2011]. However, ReLUs too suffer from an issue where the units may die during training.

A dead unit occurs when a very large gradient cripples the weights on the connections to the point where the unit can never be activated and therefore can not be updated [Clevert et al., 2015]. As a result, some filters in convolutional layers may be regressed into oblivion over the course of training. It follows that any down-sampling layers would almost certainly ignore these highly atrophied filters, as any input signal convolved with a low-energy filter will result in a low energy output. As such, the low-energy filters become easily redundant in the face of higher-energy non-orthogonal filters.

In traditional training approaches, there exist no mechanisms to dynamically increase or decrease the number of neurons per layer. With fixed neuron count per layer there can be three cases for the representation captured in each layer:

(i) The layer contains too few neurons to fully model the input.

(ii) The layer contains the ideal number of neurons to fully model the input.

(iii) The model contains an excess of neurons, beyond what is necessary to model the input.

The ideal network for a given dataset presents itself as a technical implausibility due to the immense complexity of the networks unless every aspect is controlled for and randomness eliminated. For this reason, case (*ii*) is seldom experienced beyond handcrafted toy problems. Therefore, in this thesis I will instead focus on cases (*i*) and (*iii*).

In case (*i*) where the network contains too few neurons, the general strategy has been to simply add more neurons. There have been some works related to the transfer of learned features across networks [Yosinski et al., 2014] and task retraining [Mallya and Lazebnik, 2017]; however, the problem of adding neurons during or after training remains largely open. Such ex post facto approaches restrict the network in that they incorporate the new neurons into the existing representational hierarchy rather than engaging in a plasticity related process. In general, case (*i*) is challenging as it requires knowledge of what information is to be captured per level, which in itself requires intimate and tested knowledge of the data, network architecture, and chosen hyper-parameters. In practice, is it simpler to train another network with more parameters to model the problem. A naïve method to set the number of parameters is to add neurons or layers until the training becomes impractical or begins to negatively affect model performance. A consequence of the above solution presents us with case (*iii*).

While a large number of neurons may have been useful for learning, an excess of neurons will, in the best case do nothing. In the worst case, these extra neurons will be a source of noise or interfere with the activations of subsequent neurons. Where in all cases there exists the possibility of suboptimal representation and use of neurons, case (*iii*) is unique in that there are neurons within the model known to have a non-beneficial contribution. Removing such neurons yields a reduction in network size, realized as a reduction in compute and memory footprint when using the model.

The contribution of this work is to explore the degree of redundancy, potentially exhibited in any of the above cases, in commonly used convolutional networks, and the consequences of their removal. To this end, I propose two pruning algorithms: a genetic algorithm-based weight pruning algorithm to target the fully connected layers, and an energy-based pruning strategy for the convolutional layers. From the experiments performed, both algorithms show fascinating results, indicating that both fully connected layers and convolutional layers can be compressed substantially across a variety of model architectures.

The genetic algorithm was applied to models trained on the MNIST handwritten digit dataset, and showed compression of over 70% and up to nearly 90% of layer parameters can be pruned without retraining while also yielding a mild improvement in test set accuracy. The energy-based filter pruning algorithm was applied to models trained on the much more intensive ImageNet dataset, and showed that these models also exhibited significant redundancy, incurring as much as 90% compression in over 1/3 of layers or near 50% compression in 2/3 of layers with only a single digit loss in model accuracy per layer. In the analysis of the results, I present further insights on model structure and the redundancy found in these models.

The thesis is organized as follows: I present a comprehensive literature survey, establishing the conceptual foundation in chapter 2, and examine relevant candidate convolutional neural networks and key related works in chapter 3. Building off this contemporary literature, I introduce and explain my two proposed pruning strategies in chapter 4. This is followed by empirical results of the general applicability of the presented algorithms in chapter 5, with other approaches of limited success discussed in chapter 6. Finally, I reflect on insights gained from the trials and present concluding remarks in chapter 7, and discuss ongoing and future works in chapter 8.

Chapter 2

Background

In this chapter I summarize the core concepts on which the related works and my proposed techniques are built. I present an overview of genetic algorithms, multilayer perceptrons, and convolutional neural networks. Following this, I describe datasets employed for my experiments: MNIST and ImageNet.

2.1 Genetic Algorithm

Genetic Algorithms (GA) are an optimization strategy within the family of evolutionary algorithms, wherein solutions are generated and evolved with biologically inspired operators against a specific fitness criteria that determines the success of each potential solution [Mitchell, 1998]. They are a very powerful and flexible global optimization tools which can be used to traverse large, turbulent solution spaces in far reaching applications.

Populations can be initialized either randomly, or with domain specific information, distributions, or known solutions. The algorithm follows the template of operations below, where each phase is performed for each individual within the population: *Selection*, *Crossover*, *Mutation*, and *Evaluation*. The pseudocode is given in Algorithm 1.

Algorithm 1 Basic Genetic Algorithm Pseudocode

```
1: function GENETIC ALGORITHM
2:     $population = initialize\_population()$
3:     $fitnesses = evaluate(population)$
4:     while $not\ end\_condition$ do
5:         $partners = selection(population)$
6:         $new\_population = crossover(population, partners)$
7:         $new\_population = mutation(new\_population)$
8:         $new\_fitnesses = evaluate(new\_population)$
9:         $replacement\_indices = new\_fitnesses > fitnesses$
10:        $population[replacement\_indices] = new\_population[replacement\_indices]$
11:        $fitnesses[replacement\_indices] = new\_fitnesses[replacement\_indices]$
12:    $return\ population[best\_index(fitnesses)]$
```

After the population is *Initialized* and *Evaluated* based on the fitness function, the first phase is *Selection* where, for each individual, a partner is selected to perform Crossover with. This phase generally entails a favourable selection from a subset of individuals, based on their fitness scores. During *Crossover* each individual and their selected partner exchange a subset of their chromosomes. Following this, *Mutation* modifies the individual's chromosome in a random or targeted manner. The probability of Crossover and Mutation occurring is a tunable parameter, determined by the goal of the algorithm or the state of the population. For example, a more sophisticated genetic algorithm can tune the rate of Crossover and Mutation to escape local minima or breed out less successful genetic material. Lastly, the newly generated individuals are again *Evaluated* with the fitness function; the new individuals can unilaterally or conditionally replace existing individuals. The process is continued until a suitable termination condition is reached, such as reaching a particular fitness (or other metric) score, iterating for a set number of generations, iterating for a given period of time, among others.

The steps of the GA produce a powerful and flexible global optimization tools which can be used to traverse large, turbulent solution spaces in far reaching applications.

2.2 Multilayer Perceptron

Multilayer Perceptrons (MLP), a form of Artificial Neural Network (ANN), are models inspired by a simplified interpretation of biological neural networks, composed of layers of nodes (artificial neurons) connected in an all-to-all forward-feeding manner. These weighted connections between nodes act as synapses connecting the neurons, propagating information through the network based on the node's activation function. The weighted outputs of the nodes from the previous layer are summed or otherwise combined and passed through a transformation function as a way to map the inputs to a fixed range. Take for example a hyperbolic tangent activation function, which produces an output in the range of -1 to 1 for any input. During the learning process of the ANN, these weighted edges are optimized, and are often referred to as the *parameters* of the model. A multilayer perceptron is presented in Figure 2.1.

MLPs, while inspired by the brain and its general intelligence, only excel in narrow applications. One important limitation of MLPs is in their ability to scale when attempting one-to-one correspondence to data with a large number of inputs, for example image data [LeCun et al., 1998a]. MLPs are, however, used as part of many convolutional networks to map the convolutional layers to another, a new plane, or to the output plane. Within the context of convolutional neural networks, MLPs are often referred to as *Fully Connected Layers*.

Figure 2.1: Multilayer Perceptron Network

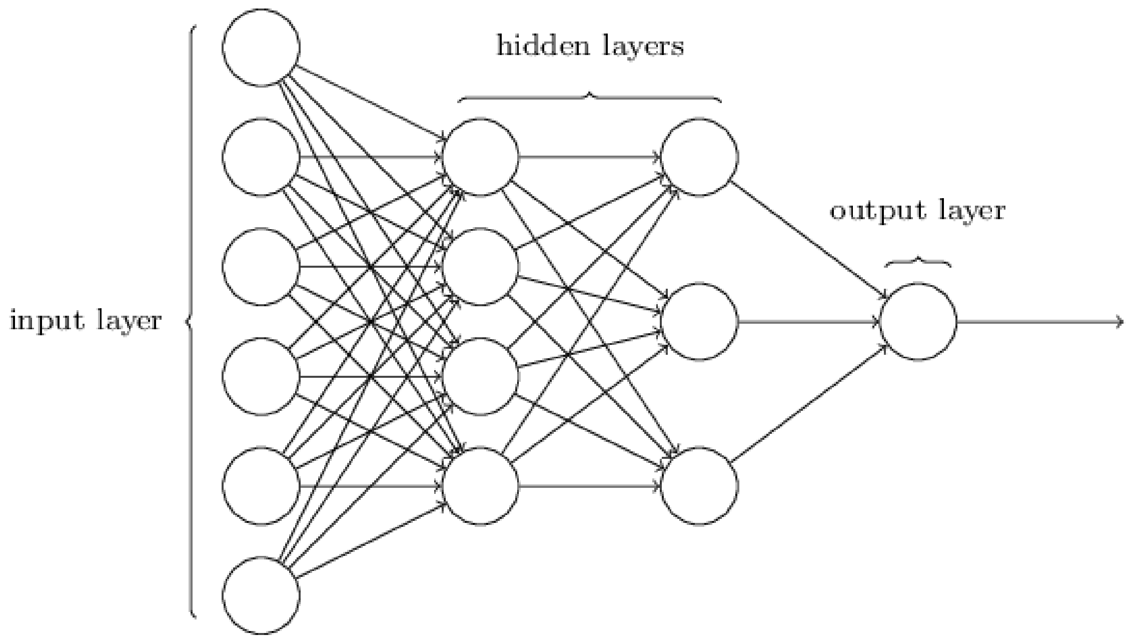


Figure sourced from Wiki [2018]

2.3 Convolutional Neural Networks

Convolutional Neural Networks (CNN) are a form of forward-feeding neural network, that operate similarly to MLPs – neurons receive an input from units they are connected to, sum their input, pass the input through an activation function, and provide output. Where MLPs are fully connected between layers, CNNs utilize a form of weight sharing through convolutional layers. Convolution layers consist of multiple filters which, rather than being statically connected to their prior layer, are convolved over the output of the prior layer. These layers allow CNNs to process data without the need for significant preprocessing or normalization, unlike the fixed-size, relatively invariant inputs expected by MLPs [LeCun et al., 1998a,b]. Generally after the filters perform their convolutions, subsampling is applied to reduce the size of the filter outputs; max or mean pooling is commonly used, where the respective value is selected from a cluster of neurons. Following the convolutional layers, the learned features are mapped to the output with a MLP consisting of a series of fully connected layers.

Figure 2.2: Convolutional Neural Network

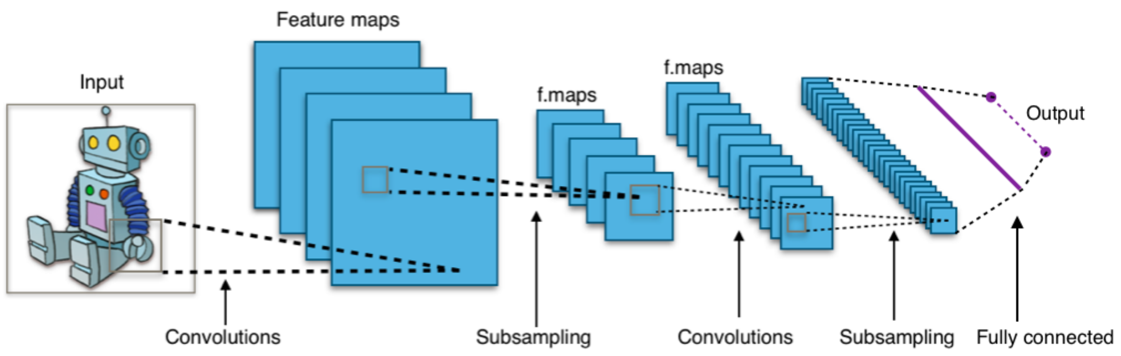


Figure sourced from Wiki [2017]

For the purposes of this work, I will focus on readily available pretrained networks of well established convolutional architectures. This was done to ensure a common, accepted, and well understood baseline for the experiments presented in chapter 5. Described below are two highly popular and generalizable convolutional network architectures: VGG[1] and ResNet[2].

2.3.1 VGG

A key work that showcased the importance of depth in contemporary convolutional network design was Very Deep Convolutional Networks for Large-Scale Image Recognition (VGG) [Simonyan and Zisserman, 2014]. Breaking from the top network designs of the time, VGG opted for many layers of smaller 3x3 filters and showed the utility and success of such configurations. VGG achieved state of the art performance in both classification and localization in ImageNet Challenge 2014 [Russakovsky et al., 2015], showcasing convolutional networks from 11 to 19 layers in depth [Simonyan and Zisserman, 2014].

VGG's success lies not only with its accuracy, but more importantly its high degree of generalizability; this makes VGG a fitting architecture for compression experiments. The 16-layer model was selected for testing as it was as performant as the deeper 19-layer model, but at a lower computational cost.

[1] Pretrained VGG16 model [Simonyan and Zisserman, 2014] acquired from `https://gist.github.com/ksimonyan/211839e770f7b538e2d8`

[2] Pretrained ResNet-152 model [He et al., 2016] acquired from `https://github.com/facebook/fb.resnet.torch/tree/master/pretrained`

2.3.2 ResNet

Deep networks are superior for their ability to integrate features of varying degrees of abstraction [LeCun et al., 2015], but there are several difficulties associated with their training. These difficulties can be handled by normalized initialization and intermediate normalization layers, but inescapably an increase in depth causes an increase in the training error. Deep Residual Networks (ResNet) were introduced by He et al. [2016], and used *shortcut connections* to efficiently pass layer outputs to downstream layers, allowing for very deep models [He et al., 2016]. These connections were added as a means of skipping layers without accumulating additional parameters nor increasing the computational complexity. The original publication introduced ResNet on ImageNet and CIFAR-10 datasets, showing easier optimization and lower training error than other plainly stacked networks.

As a result of its high degree of performance, generalizabilty, and immense depth, ResNet is another excellent candidate for filter pruning. At the time of its publication, the 152-layer ResNet model was the deepest network presented on ImageNet, but due to its simplified structure, had a lower complexity than VGG [He et al., 2016].

2.4 Datasets

In this section I provide an overview of two highly prominent datasets used in the field of computer vision: MNIST and ImageNet. Both datasets are used for image classification, where models are evaluated based on their performance in outputting the correct label for each sample image.

The MNIST handwritten digit dataset is a staple of modern machine learning. At its introduction, the dataset provided a task that was easy for humans to classify, but difficult for machines to classify. The dataset remains relevant through providing a compact and well studied model performance benchmark to test and compare contemporary vision models.

ImageNet is a more recent and far larger dataset, featuring full colour images for many thousand object categories. ImageNet presents itself as a general purpose computer vision dataset, and has inspired the design of new architectures and learning methods to meet the challenge.

2.4.1 MNIST

The MNIST dataset [LeCun et al., 1998a] is a collection of handwritten digits, created for the purposes of image processing and machine learning. Based on the National Institute of Standards and Technology's (NIST) handwritten character dataset, the dataset provides 10 classes of labeled data of the digits from 0 to 9 collected from high school students and employees across the United States of America. The dataset has 60 000 samples in the training set with 1 000 samples for the test set. The samples are presented as 28 by 28 pixel greyscale images, based on the original 20 by 20 pixel black and white NIST images. As the number of classes is small, the error is generally tracked as total classification error rather than per class; state of the art models currently achieve error rates as low as 0.23% [LeCun and Cortes].

2.4.2 ImageNet

The ImageNet dataset [Russakovsky et al., 2015] is a collection of labeled object images, created to use for visual object recognition software research. This dataset was first introduced by Princeton University's Department of Computer Science in 2009 at the Conference on Computer Vision and Pattern Recognition (CVPR) by Russakovsky et al. [2015]. The dataset has over 14 million third-party images, labelled in over 20 000 categories. For each category, 90% of the images are randomly allocated to the training set and the rest to the testing set. During use of the dataset, the samples are often resized to a resolution of 256 by 256 pixel. Commonly, model performance is evaluated via two metrics of accuracy: Top-1 and Top-5. In the case of Top-1, the predicted class with the strongest response is compared with the sample's label, while for Top-5 a prediction is considered correct if the sample's label is in the top 5 predicted classes.

Chapter 3

Related Works

In this chapter I present and contrast works related to pruning or, more generally, works pertaining to the elimination or optimization of model parameters.

There have been many influential works which used a Genetic Algorithm to optimize the structure of neural networks and improve their success. One particular work of interest was Whitley et al. [1990], where a form of GA structure optimization was performed, not to apply compression or eliminate redundancy, but to aid in learning. Through their experiments the authors showed that training could be vastly improved though evolved connectivity. For a set of basic problems, it was demonstrated that randomly initialized networks were unable to reliably converge to a passable solution, but with a Genetic Algorithm reconnecting weights, the modified networks converged rapidly. This was accomplished through a modified GA coined *GENITOR*, which, rather than the offspring replacing the parents in the population as would occur in a standard GA, the offspring replace the lowest fitness population members. *GENITOR II* further improved upon the performance of its predecessor by employing several smaller populations and swapping individuals to improve the diversity of the populations. This multi-population method allowed each population to function independently and converge quickly within their own search space. This has the further benefit of guarding the populations from getting trapped in a local minima, as only high fitness individuals from each population are exchanged. The key advantage of the evolved approach was that the GA was less likely to become trapped in local minima, as it does not need to follow the error gradient as gradient descent does [Whitley et al., 1995]. While the results of the *GENITOR* algorithms were stronger than other approaches, including backpropagation, the algorithms were excessively slow due to the search and sorting of the population involved. To compound the issue, there are a multitude of possible solutions for any given network so crossover can disrupt these solutions when mating individuals [Whitley et al., 1995].

An early work on weight pruning was *Optimal Brain Damage* (OBD) [LeCun et al., 1990], which correctly anticipated the trend in neural networks towards larger networks of greater size and structure. As established earlier, more parameters makes the problem easier to learn, however, there is a greater chance of overfitting and, consequently, reduced generalizability [LeCun et al., 1990; Reed, 1993]. The optimal solution, therefore, is one that favourably balances error and complexity [LeCun et al., 1990]. Unfortunately, it is not obvious which parameters should be deleted, nor what the impact will be to the network. Furthermore, whatever the pruning criteria, it must be able to scale effectively to service large networks. The solution provided by LeCun et al. was to prune by saliency, defined as the change in the objective function (loss) incurred by that parameter's deletion. To keep the solution scalable, saliency was estimated by analytically predicting the perturbation of the parameters with a Taylor series expansion; the goal was to find the set of parameters whose deletion would incur the lowest increase in error. The process for a given model was to train until the network reached a minima, compute and sort parameters by salience, and prune the parameters by setting them to 0. The results presented in the work showed that between 30–60% network wide pruning can be expected with minimal loss in accuracy.

Shortly after the publication of *Optimal Brain Damage*, Hassibi and Stork published a similar approach titled *Optimal Brain Surgeon* (OBS). The key difference being that OBS uses a more general, and computationally intensive, definition of saliency to more accurately estimate the error incurred by each parameter's deletion. In a follow up work, Hassibi et al. performed additional comparisons between OBD and OBS, and found in both methods that retraining reintroduces training set noise in highly pruned networks, hurting generalizability [Hassibi et al., 1994].

Following the introduction of convolutional neural networks [LeCun et al., 1998a], and their implementation [Simard et al., 2005] and popularization on Graphics Processing Units (GPU) [Krizhevsky et al., 2012] there were a number of works whose focus was around the pruning and effective use of parameters in convolutional layers.

One such work was *Learning both Weights and Connections for Efficient Neural Networks* by Han et al.. As the authors explain, while modern neural networks are very powerful, they consume considerable storage, bandwidth, and compute resources – well beyond the resources available to mobile devices. Furthermore, training cannot improve network architecture as it is fixed before training starts. To address this, the authors introduced a pruning method based on weight threshold, removing neurons when their input or output connections summed to 0. The method followed an iterative process of pruning and retraining, to compensate for error introduced by pruning. Notably, the rate of dropout was adjusted based on the degree of pruning in the layers. Experiments run on MNIST-trained convolutional networks were reduced to $^1/_9$ their initial size and reduced to $^1/_3$ their initial compute cost. Unfortunately, the retraining took 173 hours, far outstripping the initial network training time of 75 hours.

Similarly, work has been done to explore and reduce redundancy in networks used for transfer learning – a popular approach to achieving state-of-the-art accuracy by leveraging existing, large pretrained networks by retraining them for other, often smaller, datasets or tasks. Though the accuracy is excellent in the specialized domains the networks have been repurposed towards, they are often excessively resource intensive and far more complex than necessary [Molchanov et al., 2016]. To remove the excess parameters from the model during transfer learning, Molchanov et al. implemented a greedy pruning strategy with retraining. The process is to first train a model until it converges, then iterate pruning with retraining, and repeat until the target trade-off between accuracy and inference performance is met.

A variety of pruning criteria were evaluated including minimum weight, activation, mutual information, and Taylor series from OBD, modified Taylor series, average percentage of zeros, random, and reinitialization. Across all the trials, the most successful pruning criteria were the Taylor expansion-based methods. The results show that the process can be very beneficial to various network architectures used for transfer learning, including AlexNet, VGG-16, and R3DCNN, on a variety of datasets and hardware; the degree of pruning and accuracy loss varied per dataset.

Building conceptually upon prior transfer learning works, PackNet [Mallya and Lazebnik, 2017] presented a very interesting twist on network pruning. Rather than removing weights to optimize a network for a given task, the authors presented a way to reuse the pruned weights to learn new tasks. First, 50–75% of the lowest absolute magnitude weights of all convolutional and fully connected layers are set to 0; the weights are frozen and the model retrained to compensate for their removal. Next, the pruned weights are randomly initialized and used to learn a new task or dataset, while freezing the weights of the previously learned tasks. This process allows multiple datasets to be learned in a single network with performance near to the accuracy achievable with separate networks. This method does not remove or in any way reconnect or modify the structure of the network; rather, layers are sparsified and the pruned connections are set aside and used to learn for new tasks. A follow-up work introduced *Piggyback* networks, which showed it possible to learn new tasks by learning binary masks for the weights of the host network, without the need for modifying the weights through retraining [Mallya and Lazebnik, 2018].

While these pruning techniques can help networks, even state-of-the-art architectures, a key limitation is that while the redundant parameters are removed, retraining, especially when iterative, allows the network to further optimize the learned features. However, this comes at a heavy storage and compute cost, where the pruning methods presented above often take several times longer to prune their networks than was required to train them. Additionally, to perform retraining, the full dataset is required.

Datasets, like the networks used to learn them, are ever-growing in resolution, number of samples and categories, and underlying complexity.

For the purpose of this thesis, I focus on the degree to which networks can be pruned *without* retraining. This allows the methods proposed in chapter 4 to be applicable to a wide range of models with low compute cost, and requires only the test-set of each dataset. However, while the redundant representations within the model may be removed, the willful omission of retraining does not permit any further optimization of the network structure.

An alternative, perhaps complimentary, approach to post-training pruning is Neuroevolution. There are many works which attempt to generate the network architecture during training, such as *Evolving Neural Networks through Augmenting Topologies* [Stanley and Miikkulainen, 2002], and more recently *Neural networks with differentiable structure* [Miconi, 2016]. These works, though highly interesting, are outside the scope of this thesis.

Chapter 4

Eliminating Redundancy

For the task of general weight pruning in artificial neural networks, I propose two techniques: a Weight Pruning Genetic Algorithm algorithm for Multilayer Perceptrons, and an Energy-based Filter Pruning strategy.

4.1 Weight Pruning Genetic Algorithm

Weight pruning in artificial neural networks has been well studied for decades [LeCun et al., 1990; Hassibi and Stork, 1993; Mallya and Lazebnik, 2017], including the use of genetic algorithms for pruning [Whitley et al., 1990] and lower bit precision [Han et al., 2015a]; many of these works are focused around reducing the memory footprint of the models. Existing regularization techniques can ensure the network better utilizes available neurons during training, but the exact number of neurons and network topology for a given dataset are unknown. As a result, there will inevitably be some degree of redundant or vestigial structure incurred during training. The goal is to explore the nature of redundancy in neural networks, and contribute insights for network compression and model structure.

I propose a Genetic Algorithm (GA) with custom pruning mutators to serve on Multilayer Perceptron networks, outlined in Algorithm 2. GA was chosen as the basis of this proposed algorithm as it is a strong global heuristic search optimization with a long history of robust solution generation in a staggeringly wide range of applications.

4.1.1 Fitness function

It is necessary to define a clear fitness function to describe the quality of a solution. To measure solution quality for a given network, I considered two aspects to base the fitness function around: how well the model can classify the input, and how many weights are needed to achieve its degree of accuracy. Put simply, the desired outcome is to compress a given network as much as possible, without significant expense of classification accuracy. To facilitate this trade-off, the fitness function can simply be defined as the weighted sum of the overall model *Accuracy* across the test data set, and the ratio of pruned weights, expressed as the *Compression Ratio*. The equation of this notion is presented in Equation 4.1, with α and β representing the weights of the components of the fitness function.

$$fitness = \alpha \cdot \Delta_{Accuracy} + \beta \cdot Q_{Compression} \tag{4.1}$$

Change in accuracy is defined as the change in the correct classification of the sample, the True Positive rate (TP), with respect to the total number of samples in the test dataset. Each candidate pruned solution (TP_{pruned}) is compared against the accuracy of the initial starting network ($TP_{starting}$). For each layer weight's (w), the compression ratio is defined as the number of weights equal to zero ($\delta_{w_i,0}$) divided by the total number of weights in the layer. The fitness function is presented in full in Equation 4.2.

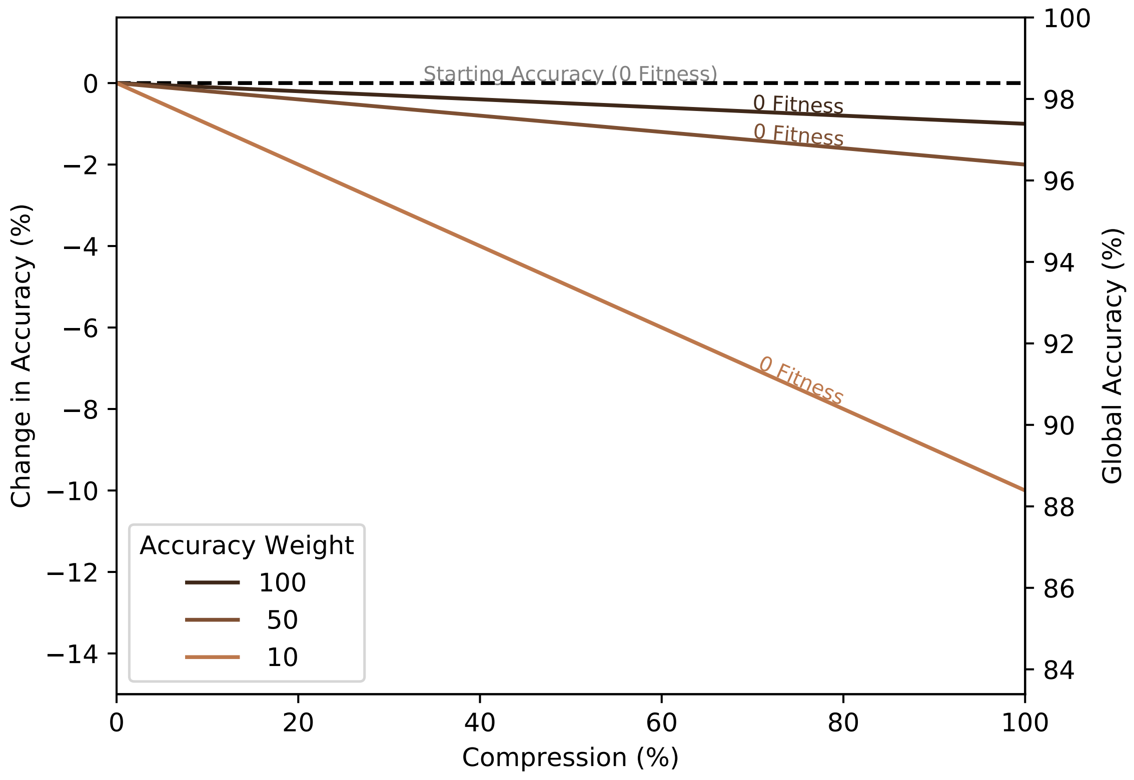
$$fitness = \alpha \cdot \frac{TP_{pruned} - TP_{starting}}{|Samples|} + \beta \cdot \frac{1}{|w|} \sum_{i=0}^{|w|} \delta_{w_i,0} \tag{4.2}$$

The initial layer starts at 0 fitness, having 0% change in accuracy and 0% compression. The fitness function component weights α and β can be used to set the desired trade-off between accuracy and compression. For example, with α and β both set to 1.0, a one percent gain in compression at the expense of a one percent loss in accuracy would have a fitness of 0, equal to that of the initial layer weights. In this example, a 1:1 trade-off is highly undesirable as it is easier to add more weights than it is to use them efficiently; therefore, a high weight on accuracy is suggested such that the GA does not trade accuracy for compression easily.

Figure 4.1 illustrates the 0-fitness lines for varying accuracy weights α with a fixed compression weight β of 1.0. Compression ratio is on the X-axis, with change in accuracy from the initial layer and global accuracy shown on the left and right sides of the Y-axis, respectively.

The objective function of the GA is to maximize the score of the model classification accuracy and ratio of eliminated weights. This approach is applicable to linear operators found in all common network architectures, and allows the balance between compression and accuracy to be tuned to meet the needs of the application.

Figure 4.1: Fitness Function



4.1.2 Algorithm

In this work, the chromosomes represent the binary usage of weights in the target layer of the network. This is to say, if there are n weights connecting two layers of the model, each chromosome is a binary vector of length n where each element represents the use or non-use of its corresponding weight. Algorithm 2 shows the pseudocode of the algorithm, and detailed explanation of each step is presented as follows:

Algorithm 2 Weight Pruning Genetic Algorithm

```
1: function GetFitness($population$)
2:   $fitnesses = array(|population|)$
3:   for $index = 1, |fitnesses|$ do
4:     $model' = copy(model)$
5:     $model'[target\_layer].weight = population[index]$
6:     $total\_valid = test(model', test\_data)$
7:     $p = population[index]$
8:     $prune\_ratio = sum(p == 0)/|p|$
9:     $fitnesses[index] = \alpha \cdot total\_valid + \beta \cdot prune\_ratio$
10:  $return\ fitnesses$
11: function EvolvePruning($starting\_weights$)
12:   $population = initialize\_population(starting\_weights)$
13:   $fitnesses = GetFitness(population)$
14:   for $generation = 1, max\_generations$ do
15:     $partners = tournament\_selection(population)$
16:     $population' = crossover(population, partners)$
17:     $population' = prune\_lowest\_weights(population')$
18:     $population' = prune\_random\_weights(population')$
19:     $fitnesses' = GetFitness(population')$
20:     for $x$ where $fitnesses' > fitnesses$ do
21:       $population[x] = population'[x]$
22:       $fitnesses[x] = fitnesses'[x]$
23:   $return\ population[argmax(fitnesses)]$
```

Initialization

To initialize the population of the genetic algorithm, each individual (excluding the first) is set to the starting weights of the layer with a desired percentage of the weights zeroed. The undamaged starting weights are retained in the first individual of the population to allow the solutions to rebuild critical weights that may have been lost in the initialization. This state of the population initialized with damaged weights and one undamaged individual is considered to be Generation 0. The call to initialize the population can be seen in the *EvolvePruning* function on line 12.

Evaluation

After the initialization of the population, the fitness scores of each individual are calculated per the fitness function described in subsection 4.1.1. The fitness function is presented through the *GetFitness* function on lines 1–10, and the call to score the initialized population occurs on line 13 in the *EvolvePruning* function of Algorithm 2.

Following this, the algorithm begins the iterative cycle of generating new individuals through selection, crossover, mutation, and evaluation; these steps can be seen in order in the *EvolvePruning* function on lines 15 to 19. Additionally, crossover and mutation occur per user-defined probabilities, tuned to the network the algorithm is set to prune.

Tournament Selection

For each chromosome in the population, m other chromosomes are randomly selected, and the one with the highest fitness chosen.

Two Point Crossover

Per selected chromosome pair, two indices are selected and the information of the chromosomes is swapped to produce offspring for the next generation. For example, given two individuals $\{i_1, i_2\}$ of length l, and two sorted unique indices $\{j, k\}$ between 0 and l, the offspring of the crossover would have chromosomal information 0–j from i_1, j–k from i_2, and k–l from i_1.

Mutation

Two pruning specific mutator functions are employed in this algorithm, the first to specifically target low magnitude weights, and the second to randomly prune weights. The mutators are unbounded and always target the remaining non-zero weights. These functions are employed to modify chromosomes to help avoid or escape local minima, and introduce diversity to the population. The mutators are presented on lines 17 and 18, respectively.

- Weakest weight termination

  Per chromosome, the weight of the lowest magnitude is set to zero. This mutator can be used to aggressively prune by a user-defined percent of the lowest magnitude; for example, to prune all weights which are less than the lowest magnitude weight plus 10%.

- Random weight termination

  Per chromosome, the values of random non-zero weights are set to zero. This mutator can also be used to prune by a user-defined rate; for example, to prune 10% of the remaining non-zero weights.

Iteration

After selection, crossover, and mutation have occurred for all individuals in the population for the current generation of solutions, each new individual is evaluated to determine their fitness. Additionally, I have used elitism, a strategy where the best solutions are carried over to the next generation [Mitchell, 1998]. This is done to ensure the solutions generated do not degrade in fitness through detrimental crossover or mutation, and to guard against loss of the best solution. This can be seen on line 20, where new individuals of greater fitness replace their respective indices in the population. This process is repeated until reaching the user-desired number of generations, and the highest-fitness solution is returned. The chromosomes of this solution are the best found replacement weights for the targeted layer.

4.2 Energy-based Filter Pruning Algorithm

Filter pruning of convolutional neural networks has garnered a fair deal of attention recently, though many of these works are focused on pruning for the sake of reduced compute and memory footprints. From the related works, we know that redundancy is either pruned away [Han et al., 2015a,b; Molchanov et al., 2016] or trained to perform a new task [Mallya and Lazebnik, 2017]. Common to all of these works is that they employ a significant amount of retraining, often greatly exceeding the amount of time spent training the model on the dataset initially.

The purpose of this algorithm, as with the Weight Pruning Genetic Algorithm, is to provide a pruning solution which can be applied to any modern network with convolutional layers, without the need for any specialized knowledge of the model or data. As contemporary convolutional models, such as VGG and ResNet, are very deep, the pruning criteria needs to be fast to allow for rapid assessment of the model's redundancy. A further encumbrance of larger models is, especially when trained on very large datasets, that the time to test the accuracy can be quite long on its own, spanning several minutes. For this reason, a genetic algorithm approach was not attempted as the fitness function would necessitate performing one such test set evaluation per member of the population. In light of these constraints, a straightforward energy-based filter pruning criteria is presented below.

4.2.1 Energy-based Filter Pruning Criteria

To assess the energy of each filter in each convolutional layer, a two dimensional linear frequency sweep, or chirp, is used to test each filter. The chirp is generated for each layer based on the expected input size set corresponding to the input image dimensions the network was trained against, and therefore contains all possible frequencies the filter of that layer could respond to: from 1 to $1/n$, where n is the diagonal resolution. As the chirp tests all frequencies a filter can respond to, the energy of the filter output signal can be used as a measure of filter utility. Specifically, filters with low energy are the prime targets for pruning, due to their minimal contribution.

Using energy as an indicator of contribution allows for the evaluation of filters in a way that does not require any knowledge of the filters, their arrangement in the model, or of the dataset the model was trained on. The equation of discrete-time signal energy [House, 2004], is used as the filter pruning criteria, and is presented in Equation 4.3.

$$E_\infty = \sum_{n=-\infty}^{\infty} |x(n)|^2 \tag{4.3}$$

4.2.2 Algorithm

The pruning algorithm relies principally on two function, the pseudocode of which is provided in Algorithm 3.

The first function, $PruneModel$, iterates over the convolutional layers of the model, and over the list of pruning energy thresholds that define the trials. This function is presented on line 14 of Algorithm 3. To evaluate the degree of redundancy in model's convolutions layers and the effects of their pruning, each layer was pruned and evaluated independently. This can be seen on line 22 where the model is reloaded after pruning and evaluating each convolutional layer.

The second function, $PruneFilters$, performs the filter pruning of the desired convolutional layer at the desired energy threshold. This function is presented on line 1 of Algorithm 3. Rather than specifying the number of filters to eliminate or target ratio, the pruning is based on the filter's contribution to the layer's output.

First, the layer to be pruned and the subsequent convolutional layer are selected, as seen on lines 2,3. Next, a chirp of appropriate size, based on the dimensions of the previous layer's output, is generated (line 4). This chirp is passed through the target layer and the energy of each filter's output is measured and normalized by the size of the layer filters (5–7). After sorting the filter energies, the contribution of each filter is given by its part of the cumulative sum energy (9). Pruning is then performed by eliminating all filters that fall below the desired energy threshold (10,11). The weights in the subsequent convolutional layers that connected to the eliminated filters of the pruned layer are also removed (12). The model is returned and the iteration of the pruning procedure continues.

Algorithm 3 Energy-based Filter Pruning Algorithm

```
1: function PruneFilters($model, layer\_idx, next\_layer\_idx, energy\_threshold$)
2:   $layer_i = model[layer\_idx]$
3:   $layer_{i+1} = model[next\_layer\_idx]$
4:   $chirp = GetChirp(layer_i.width, layer_i.height)$
5:   $filter\_responses = layer_i.forward(chirp)$
6:   $filter\_energies = GetEnergy(filter\_responses)/(layer.width \cdot layer.height)$
7:   $filter\_energies = filter\_energies/Sum(filter\_energies)$
8:   $sorted\_filter\_energies, sorted\_filter\_indices = Sort(filter\_energies)$
9:   $cumsum = CumSum(sorted\_filter\_energies)$
10:  $valid\_filters = sorted\_filter\_indices[cumsum > energy\_threshold]$
11:  $layer_i = layer_i[valid\_filters]$
12:  $layer_{i+1}.output\_plane = layer_{i+1}.output\_plane[valid\_filters]$
13:  $return\ model$
14: function PruneModel($model\_filename$)
15:  $model = load(model\_filename)$
16:  $conv\_indices = model.get\_convolutional\_layers()$
17:  $energy\_thresholds = [0.01, 0.02, 0.05, 0.10, 0.20, 0.50]$
18:  for $t\ in\ energy\_thresholds$ do
19:    for $i = 1, |conv\_indices|$ do
20:      $model = PruneFilters(model, conv\_indices[i], conv\_indices[i+1], t)$
21:      $loss, accuracy = EvaluateModel(model)$
22:      $model = load(model\_filename)$
23:  $return\ model$
```

Chapter 5

Results

Results for the Genetic Algorithm Weight Pruning and Energy-based Filter Pruning algorithms introduced in chapter 4 are presented in this chapter. The algorithms were implemented in the Lua programming language and used the Torch scientific computing framework [Collobert et al., 2011]. All experiments were run on a Nvidia GeForce GTX 1080 Ti graphics cards through the framework. The datasets were stored on a solid state disk to ensure short access times.

5.1 Genetic Algorithm Weight Pruning

To evaluate the proposed weight pruning algorithm, described in section 4.1, the algorithm was applied to two sample networks trained on the MNIST handwritten digit data. As described in subsection 2.4.1, the goal of the models is to best classify the presented 32x32 pixel handwritten digits. The first model prepared for the evaluation of the algorithm was a simple Multilayer Perceptron. The second model is a simple Convolutional Neural Network where the weight pruning algorithm was applied to the Fully Connected layer following the convolutional layers.

The following sections present and provide analysis of the results obtained from the proposed weight pruning algorithm. Details of the model structure and training parameters are provided for each model type in their respective sections.

5.1.1 Multilayer Perceptron

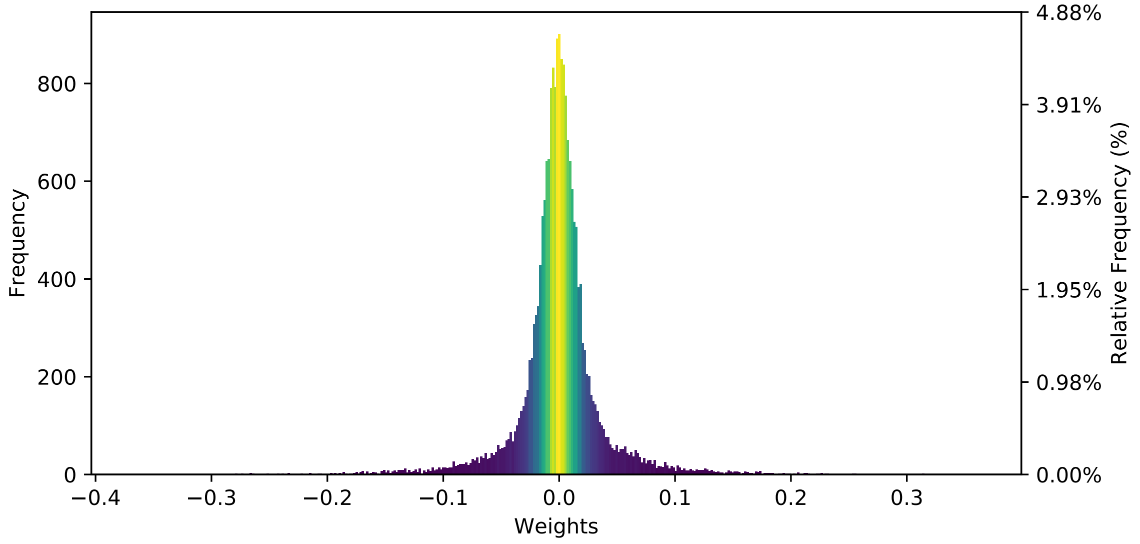
A simple Multilayer Perceptron (MLP) will be used as a basic example with which the weight pruning algorithm can be evaluated. As described in section 2.2, MLPs are layers of nodes fully connected in an all-to-all manner with nodes of the subsequent layer via weighted edges. For this experiment, a three layer neural network was employed, where the first layer had 1024 nodes, each corresponding to a pixel of the input image. The second layer contained 2048 nodes with a Hyperbolic Tangent (TanH) activation function, and the final layer contained 10 nodes, one node per output class. The MLP model was trained with 60 000 images from the dataset with stochastic gradient descent and a learning rate of 0.05 per sample; 10 000 images were reserved for testing. The trained network had an overall accuracy of 95.4% on the training data set, and 94.6% on the test data.

The Weight Pruning Genetic Algorithm was applied to the weights of the second layer of the MLP. In the sections below, I describe the initialization of the GA and present analysis of the algorithm performance through population fitness, accuracy, and compression ratio. Following this, the final solutions of each trial are analyzed and related to the fitness function defined in subsection 4.1.1.

Initial Conditions

To begin the analysis I first investigated the histogram of the second layer's weights.

Figure 5.1: Histogram of Initial Weights



In Figure 5.1 there is a clear concentration of low magnitude weights, centred around 0-magnitude with a high kurtosis of 12.03. The weights of this layer exist in the range of -0.3490 to 0.3137, with a staggering 83.45% of the weight magnitudes in the bottom 10% of the range. With this in mind, the weight pruning genetic algorithm was run to see the relationship between these low magnitude weights in the final layer and the global classification accuracy of the network.

Five trials were done to evaluate the performance of the final solution of the GA with the following initial damage rates: 10%, 30%, 50%, 70%, and 90%. All trials were performed with a population of 50 individuals and run for 20 generations, as these values were sufficient to determine the applicability of the algorithm.

Please note that all discussion and figures pertaining to some specified rate of damage refer to the same trial; all subsections below address different aspects of the five conducted MLP trials.

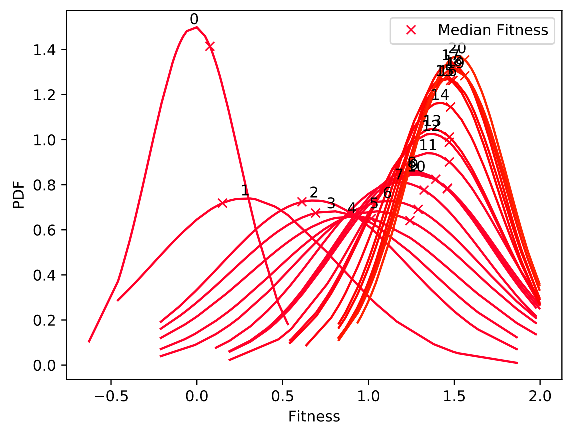
Fitness

Before performing analysis on the outcome of the compression, I first looked at the distribution of the population fitnesses per generation. This was done first to judge the degree of success of the genetic algorithm at evolving solutions and converging.

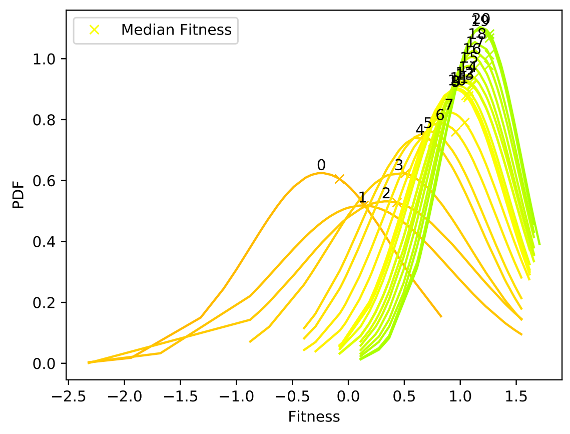
The fitnesses across generations for the trials are presented in Figure 5.2; several indicators for the success of the algorithm can be seen here. The most evident indicator is the consistent trend towards higher fitness per generation across all trials. The distributions can be observed to generally trend towards lower standard deviation in the population fitness per generation; this shows excellent convergence. Also, the skew towards higher fitness in the distributions, visible through the median per generation, is an expression of the GA's preference of higher fitness solutions during selection.

To better view the progression of the weight pruning algorithm, the fitnesses are presented as a timeseries in Figure 5.3, with generation on the X-axis and fitness on the Y-axis. In these figures, the lineage of individuals is tracked, including the top fitness and top accuracy solutions, as well as the average population fitness, through the use of plotted lines. The fitness range is presented as a colour shaded region to illustrate the extent of variation within the population.

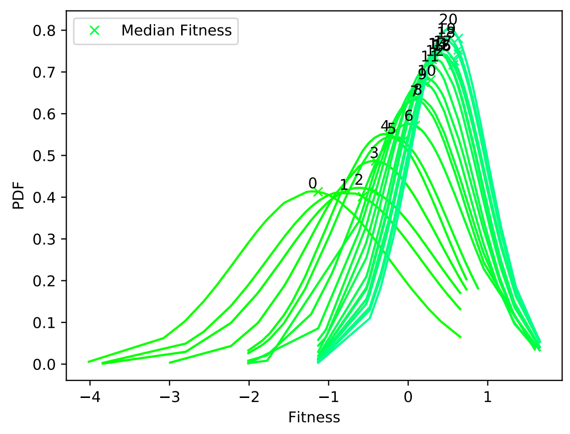
Figure 5.2: Population Fitness Distribution per Generation



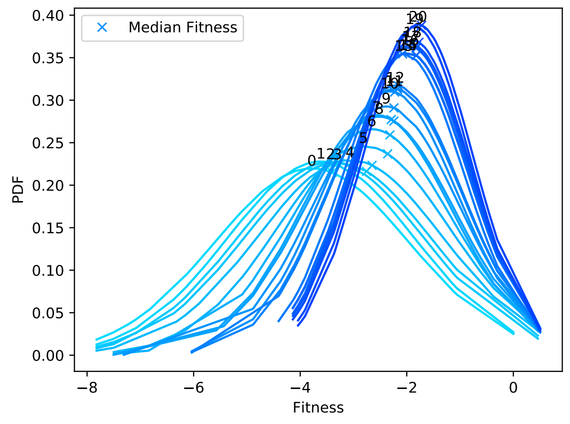
(a) 10%



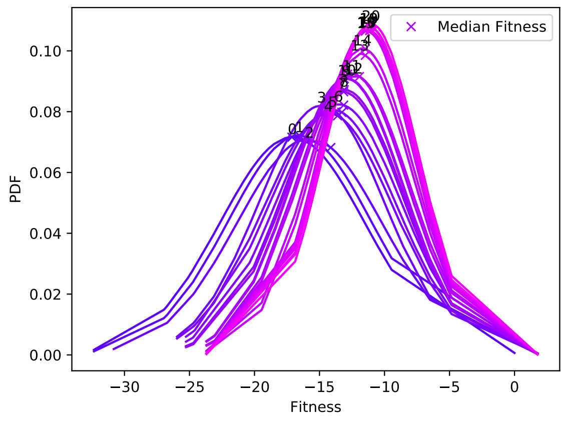
(b) 30%



(c) 50%



(d) 70%



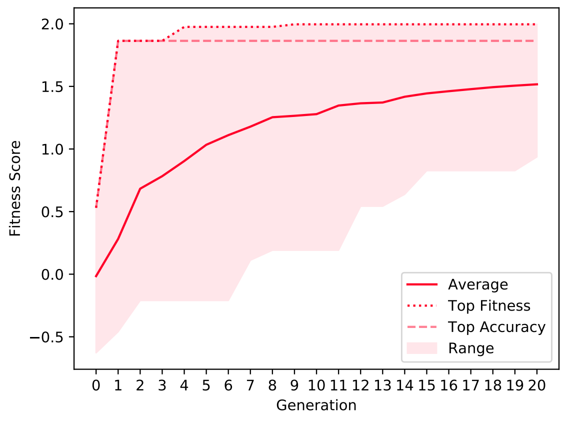
(e) 90%

While each trial contains its own individual variations, some common patterns are visible. Due to the initial random damage, the starting population has a wide fitness range as some solutions inevitably had vital weights removed and incurred catastrophic damage. The initial population, generation 0, always contains the worst fitness for all trials; this can be seen in Figures 5.2 and 5.3 where the initial population has the left-most and bottom-most fitness value per trial, respectively. Elitism ensures that the lowest fitness in generation 0 is the lowest fitness for the trial, and guarantees a non-negative trend in the minimum fitness per generation. Across the presented trials, there is a steady upward trend towards greater average fitness, and a clear trend towards tighter distribution. This is principally due to elitism causing weaker individuals to be replaced by more successful offspring, and persisting highly successful individuals in the face of less successful offspring. Once again, the median is almost always significantly past the mean, with the effect more pronounced in later generations.

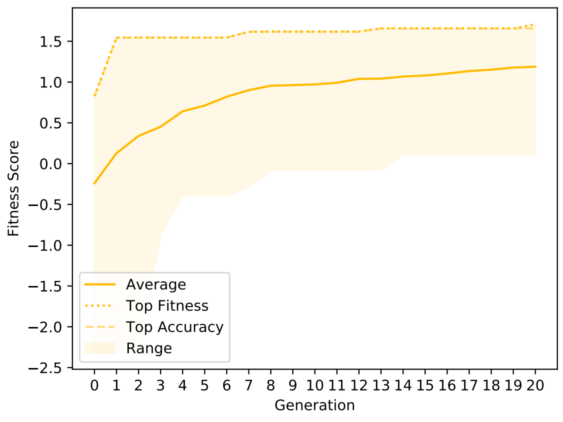
Unsurprisingly, due to the large accuracy weight in the fitness function, the individual of top accuracy per trial tends to be at the top of the fitness range. However, it is worth mentioning that in some of the trials, the solution with the top accuracy is not always the solution with the top fitness. This is most evident in Figure 5.3a, where the top fitness and top accuracy diverge after the third generation.

From these observations, the algorithm can be seen to work very well in this domain, reliably improving fitness distributions across all trials.

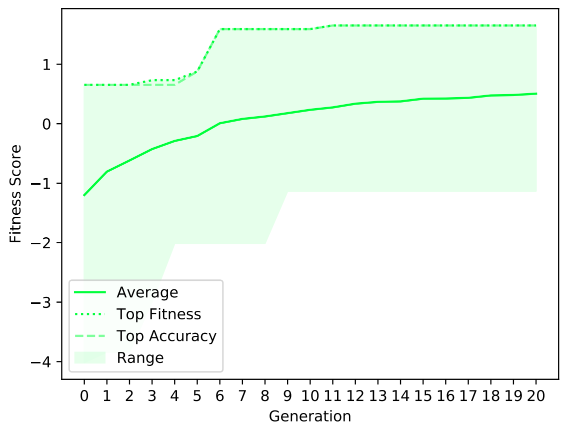
Figure 5.3: Population Fitnesses per Generation



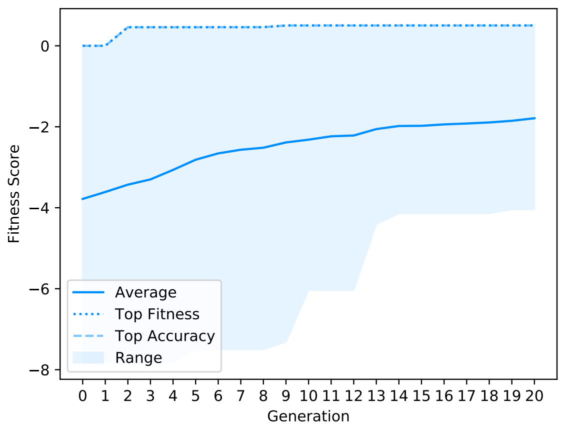
(a) 10%



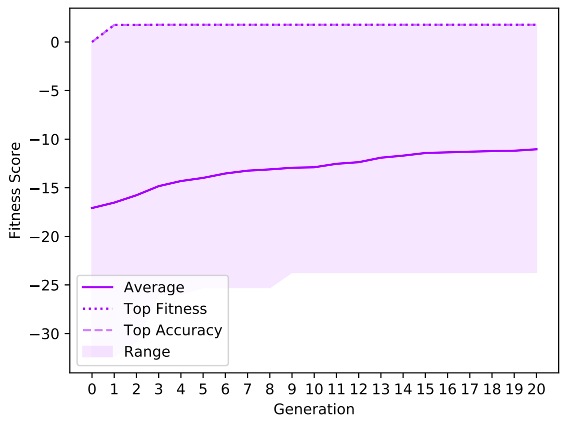
(b) 30%



(c) 50%



(d) 70%



(e) 90%

Accuracy and Compression

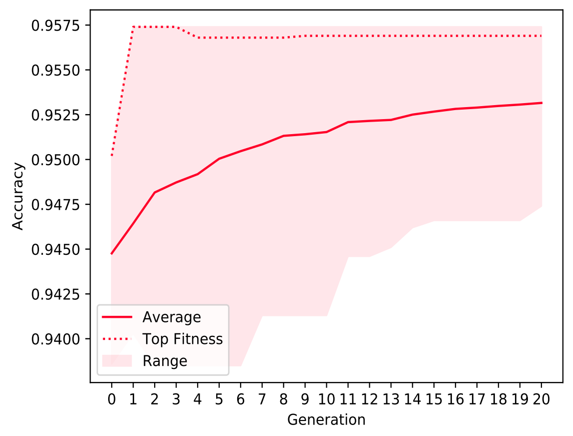
Next, I present analysis of the progression of accuracy and compression in the populations per generation, presented in Figure 5.4. For each trial, the figures for accuracy and compression are displayed side-by-side, their respective components presented on the Y-axis, and generation presented on the X-axis. The accuracy and, more-so, the rate of compression shows some very interesting results. As mentioned, the lineage of the top accuracy and top fitness individuals is tracked to show the progress of the population across the generations.

Based on the fitness function weights, α and β for accuracy and compression, there will be some point at which the trade-off between accuracy and compression among the solutions converge. As the arbitrary removal of weights can cause catastrophic failure of the network, the lineage of the top solutions will likely trace back to the initial undamaged individual. As the initial rate of damage increases, so does the likelihood that the top fitness individual will trace back to the undamaged individual.

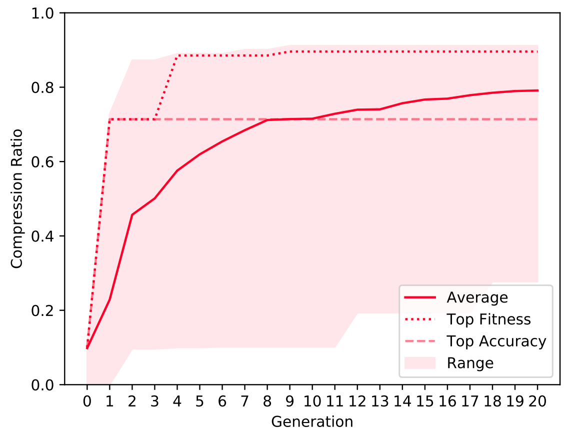
Cases where the initial damage rate is below this point of convergence will evolve solutions with a progressively greater amount of pruned weights. Conversely, cases where the initial damage rate is above this point of convergence will reduce their rate of compression and repair the critical weights.

Finally, it is worth mentioning that the undamaged solution is not strictly persisted. If a pruning of the initial undamaged individual were to take place and result in a child of higher fitness, that individual would overwrite the initial undamaged solution. This could potentially result in the permanent loss of the information of those weights from the population.

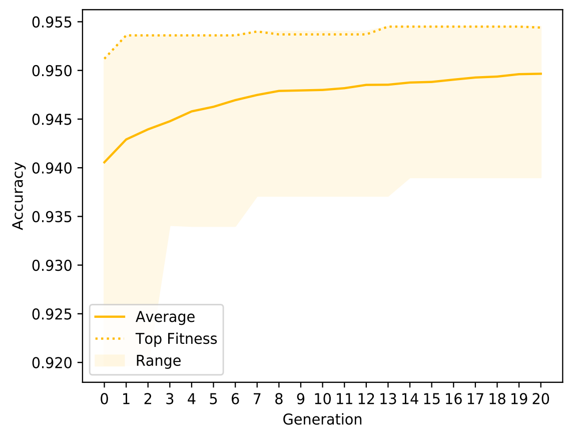
Figure 5.4: Population Accuracy & Compression per Generation



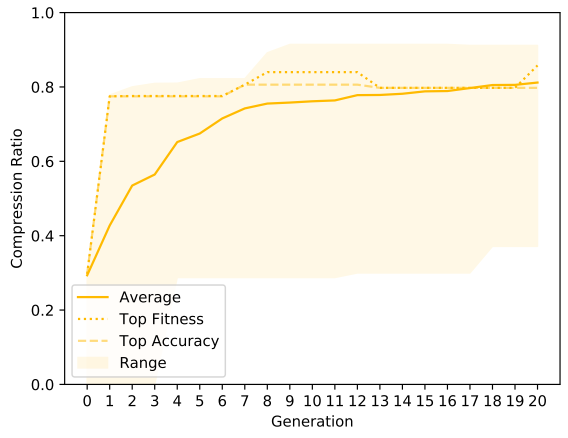
(a) 10% Accuracy



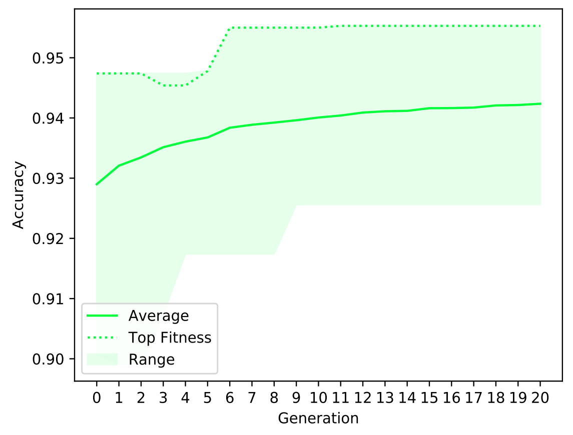
(b) 10% Compression



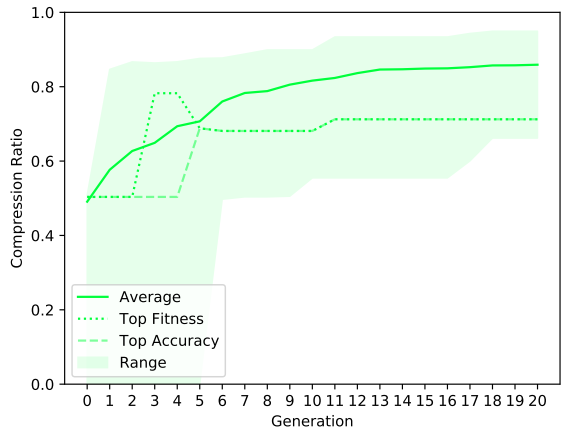
(c) 30% Accuracy



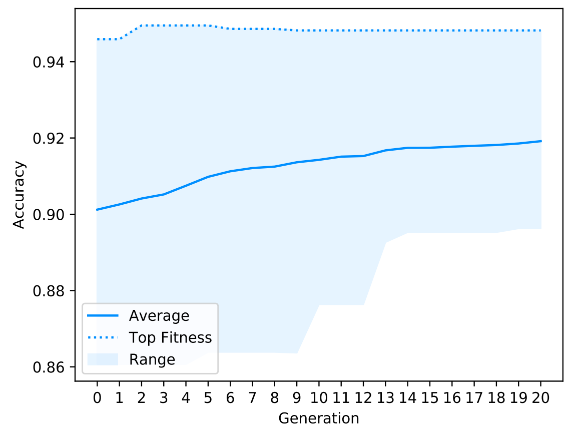
(d) 30% Compression



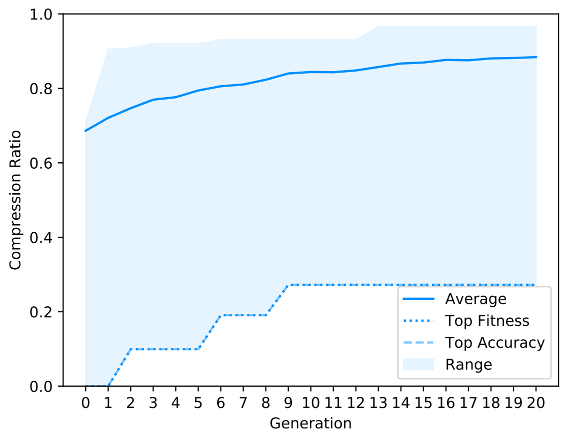
(e) 50% Accuracy



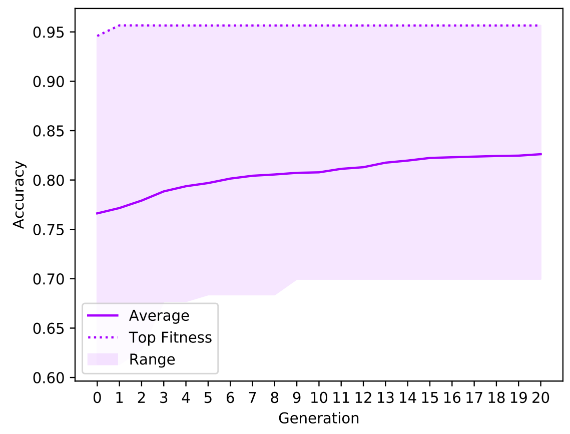
(f) 50% Compression



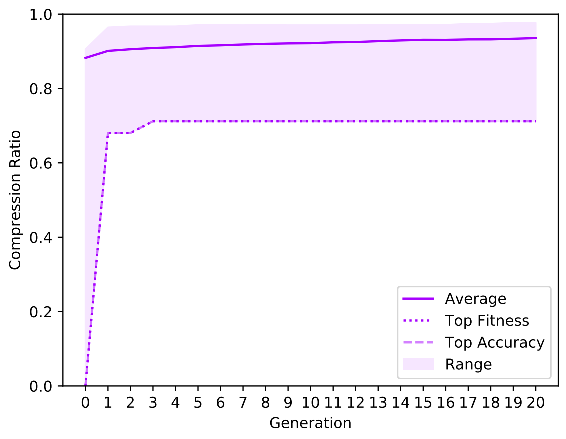
(g) 70% Accuracy



(h) 70% Compression



(i) 90% Accuracy



(j) 90% Compression

Across the trials, several generalizations can be drawn for the interaction of fitness with accuracy and compression. The large accuracy weight α forces the final top fitness solution to be of high accuracy, though substantial compression is seen across all trials, with an average compression of 85.64% across the solutions of all final generations. When considering only positive fitness final generation solutions for all trials, the average degree of compression was 65.02% for an average gain in accuracy of 0.33%. Additionally, the final solutions of top fitness were also the solutions with highest accuracy, with the exception of the 10% and 30% trials where a 0.05% and

0.01% loss in accuracy were traded for an additional 18.22% and 6.02% compression over their respective top fitness solutions.

In the 10% trial, shown in Figures 5.4a and 5.4b, we can see a steady increase of average population accuracy, with an exchange of accuracy for compression seen in generation 3. On the accompanying compression plot, we can see the matching compression increase in generation 3 for the solution of top fitness solution. The top fitness solution possesses accuracy and compression far above the respective population averages for the entire duration of the trial.

The 30% trial population accuracy, shown in Figure 5.4c, shares much in common with the 10% trial, where a small range with the top fitness solution follows the upper bound. Compression also ends with the top accuracy solution having below average compression with the top fitness solution having above average compression.

The accuracy range is still highly centred around the starting point as the degree of initial damage is quite small, and quickly repaired over the generations. In fact, we can see this repair process in the top fitness solution during the first generation in the trials from 10% to 70% initial damage. In these trials, the top individuals' accuracy improves while simultaneously eliminating many weights.

At the end of the 10% trial we can see that the best solution has an accuracy of 95.6%, higher than that of the initial network's accuracy of 95.4%, with near 90% compression. The 70% and 90% trials also share much in common, with both showing a large accuracy range, more so in the 90% trial due to the higher initial damage rate. We can see through the lower bound of the compression and upper bound of the accuracy that critical weights in the initial solution are reintroduced to the other members of the population. Compression in this trial shows the top fitness starting at 0% compression but ending at 70% compression - the lowest compression of all solutions in the trial. In all trials, even in the cases with high initial damage such as 70% and 90% initial damage, the mean compression ratio of the population trends higher. Interestingly, in these cases there are no instances where the highly compressed individuals breed with the undamaged individual to fill the intervening range. What is generally observed is that the individuals with initial damage tend to stay at the same level of compression. The population accuracies do trend upwards across the generations in compensation.

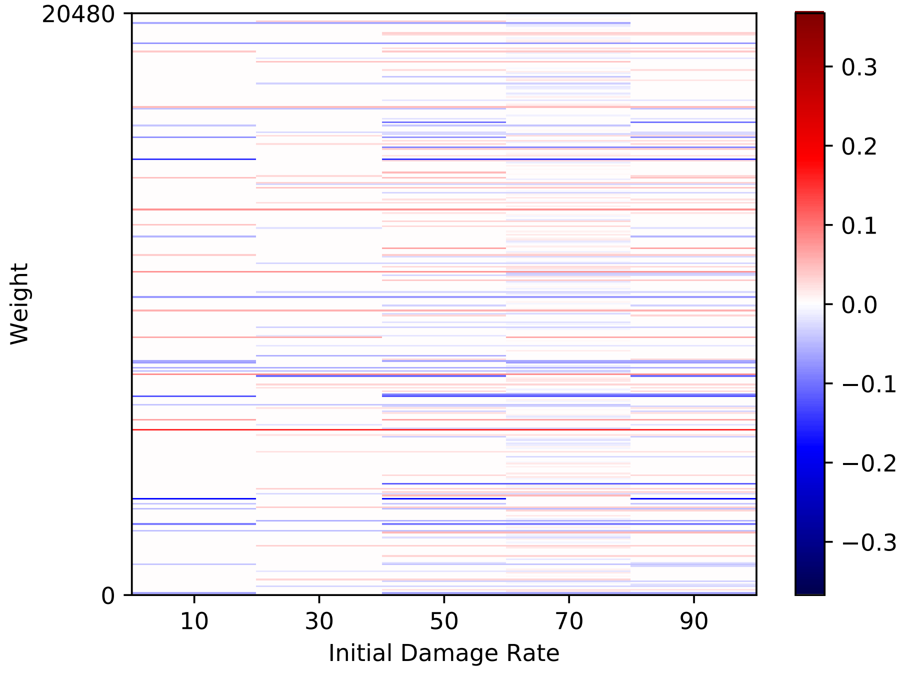
As the rate of initial damage increases, we see an increase in mean compression and a decrease in mean accuracy. This may indicate that the populations with higher initial damage are unable to reconstruct the pruned weights and the GA instead optimizes based on the weights within the population.

A key finding presented in these figures is that, for all trials, a solution with higher test set accuracy is discovered by the algorithm through pruning.

Heatmap of Trial Solutions

The heatmap of each trial's highest fitness solution is presented in Figure 5.5, and shows an interesting perspective on the utility of the weights – all the solutions are optimizing towards the same goal, yet the pruned weights vary among the solutions. In this figure, the initial damage rates are presented along the X-axis, with the weights of the solutions flattened along the Y-axis. The order of the weights is maintained across the trials to show direct correspondence of which weights were removed per trial. The values of the weights are shown as a coloured gradient, presented in the legend. Sections of the figure that are white are weights that have a value of zero, indicating that they have been removed by the algorithm. Among the solutions, we can see that different weights have been removed, but all achieve a gain in fitness and accuracy over the initial undamaged weights. We can also see that there are weights that contribute to the final solution of all trials, indicating that these weights are the ones most critical to the function of the layer. The weights not shared among the final solutions can be considered ancillary due to their interchangeable nature.

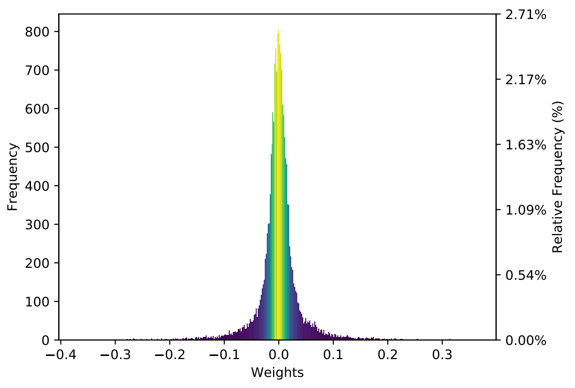
Figure 5.5: Heatmap of Final Weights



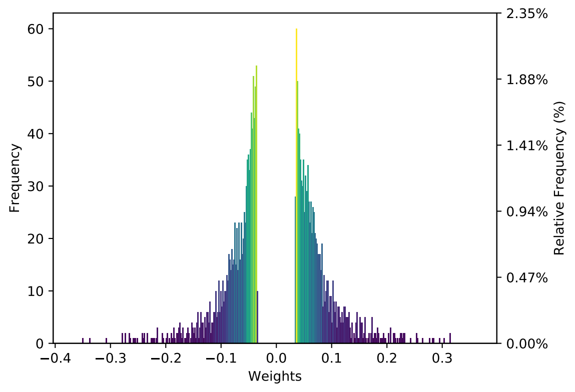
Histogram of Trial Solutions

Next, in Figure 5.6, I present the histograms of the initial and final weights of the best solution per trial to illustrate the effects of the selective deletion of low magnitude weights by the pruning algorithm. Please note that zero-valued weights are omitted.

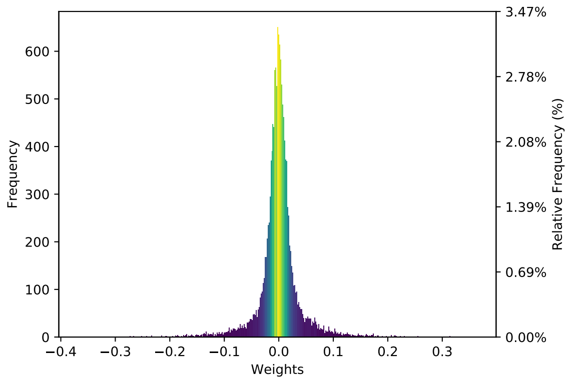
Figure 5.6: Top Fitness Solution Histograms per Trial



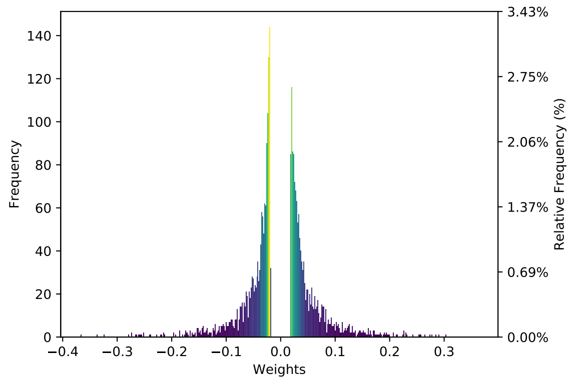
(a) 10% Initial Weight Histogram



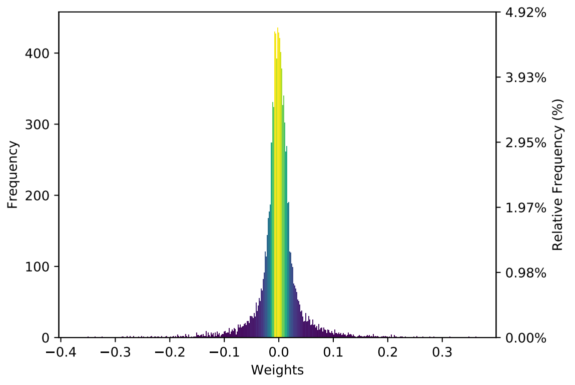
(b) 10% Final Weight Histogram



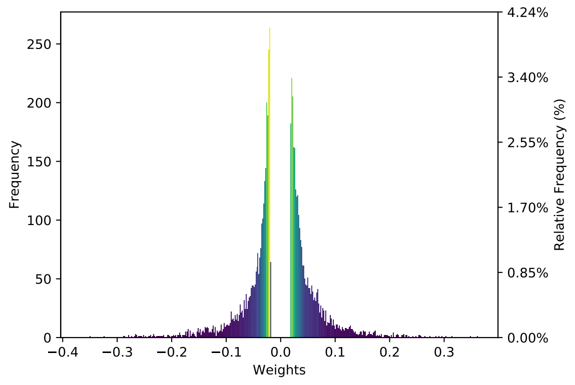
(c) 30% Initial Weight Histogram



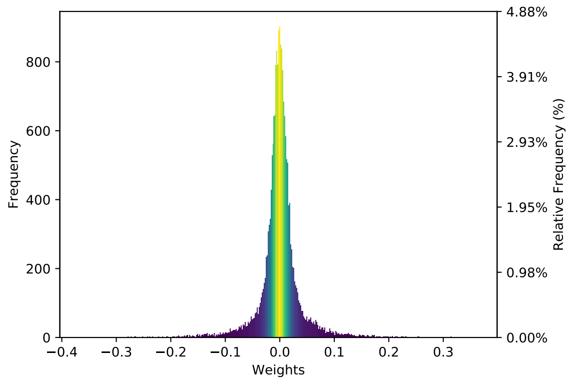
(d) 30% Final Weight Histogram



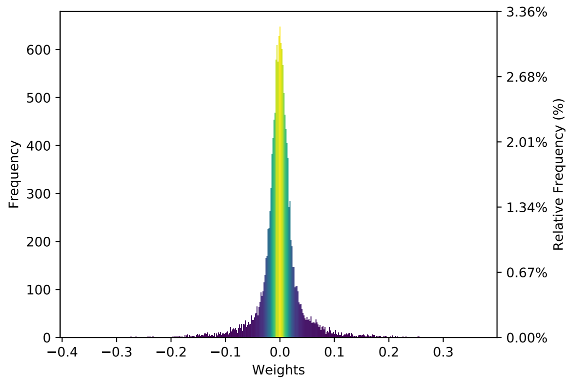
(e) 50% Initial Weight Histogram



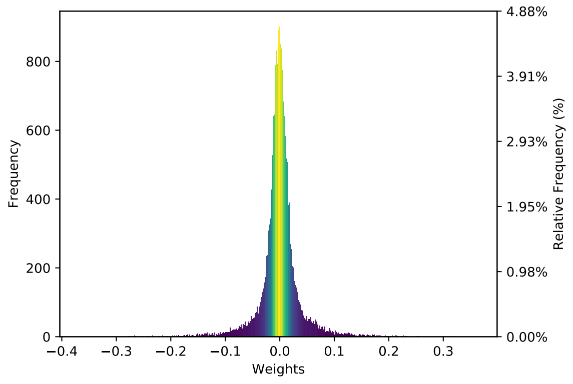
(f) 50% Final Weight Histogram



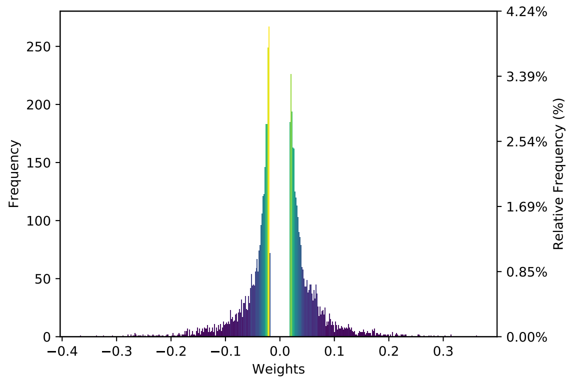
(g) 70% Initial Weight Histogram



(h) 70% Final Weight Histogram



(i) 90% Initial Weight Histogram



(j) 90% Final Weight Histogram

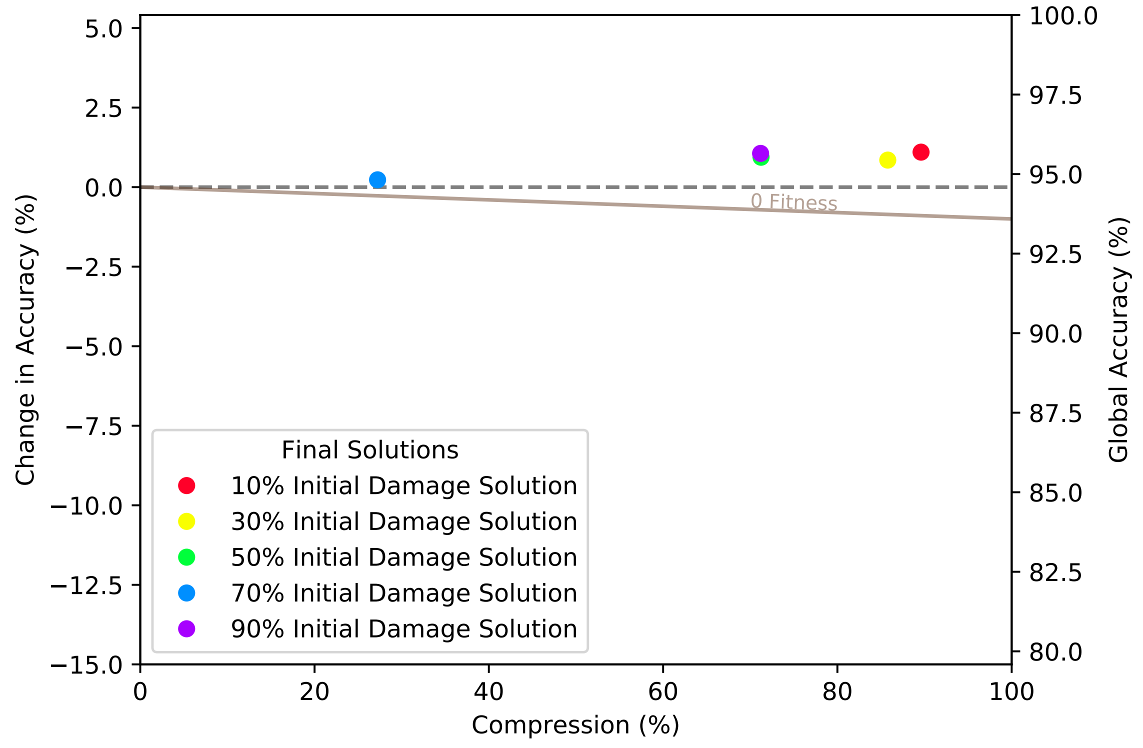
From the figure, the compression in each trial can be plainly seen as the distributions all show a substantial gap near the 0-mark. Once again, the non-zero weights in the final solutions are the weights most vital to the layer. For all final weight histograms, the pruning can be seen in the valleys on either side of the central 0-weight bin, with the exception of the 70% final weights due to the limited success of the pruning algorithm for that population. This exception proves a unique perspective as unlike the other solutions, it maintained many of the low value weights and did not have the signature gap separating the two non-zero distributions on either side of the central near-zero bin.

The 10%, 30%, 50% initial weight histograms correspond to an individual that incurred their trial's initial random damage, while the 70% and 90% clearly show that the initial weights belong to the undamaged individual within each population.

Solution Quality

Finally, to see how the trial solutions relate to the fitness function, I present the solution fitnesses overlaid in Figure 5.7.

Figure 5.7: MLP Trial Final Fitnesses



Astonishingly, for all trials the algorithm provides successful solutions that result not only in substantial pruning, but also increase the accuracy of the models on the test data set.

The 10% trial achieved the highest fitness, as that population had the greatest diversity and greatest opportunity to selectively prune without pressure to reconstruct vital weights. The 30% trial follows closely for both accuracy and compression, and had similar final accuracy and compression ranges indicating that it was also a good starting damage rate. With 50% initial damage it becomes readily apparent that the population struggles to reconstruct vital weights, with only a few individuals meeting the accuracy of the initial undamaged weights. 70% and 90% were unable to create any individuals to match the accuracy of the undamaged starting weights, returning a final solution based on the undamaged starting weight individual.

The 10% initial damage rate trial presented the highest fitness, and was able to prune 6.81% of the weight range of the layers, resulting in the elimination of 72.4% of the parameters.

From analyzing the progress of the individuals across the trials, it is clear that lower starting damage is better as it allows the population to selectively eliminate weights that do not significantly harm model accuracy, rather than aggressively discarding random weights and being forced to recover critical weights if available. The algorithm was designed with specific mutators for pruning non-zero weights, presented in section 4.1.2, which, if activated, would always prune some parameters. The restoration of weights had no specific mutator and was left entirely to crossover. This limited ability to restore pruned weights was compounded by the innate sparsity of individuals in populations initialized with high initial damage rates. This is to say, sparse individuals are more likely to swap empty weights during crossover as they have less useful information to share.

5.1.2 Convolutional Network

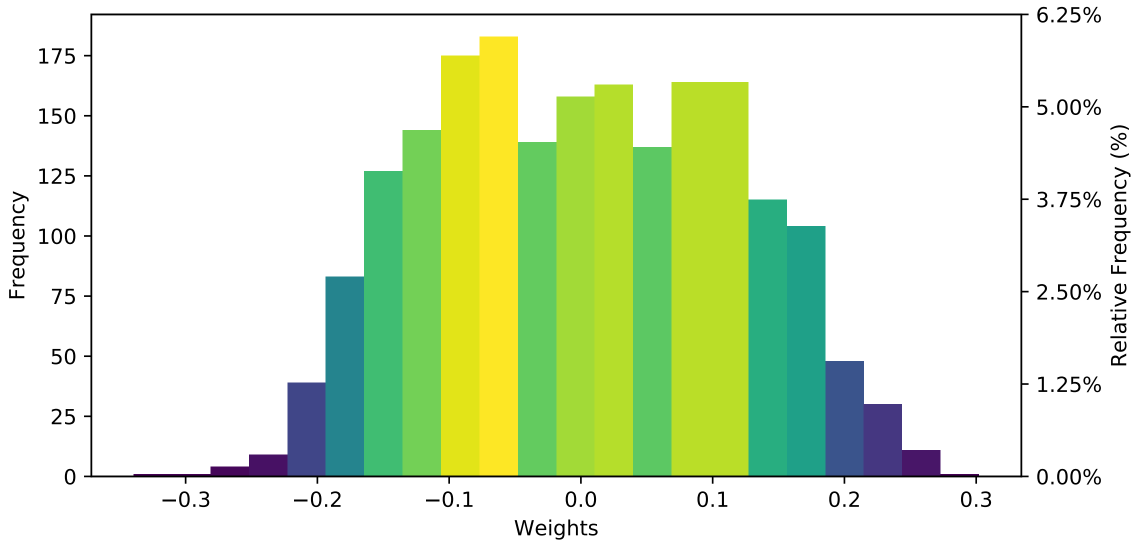
Next, I applied the Weight Pruning Genetic Algorithm to the final fully connected layer of a simple Convolutional Neural Network (CNN). As described in section 2.3, CNN's are better able to handle larger inputs through the use of weight sharing *filters*, naturally suited for image data. The model was composed of two convolutional layers with 32 and 64 5x5 filters, respectively. Each convolutional layer used a hyperbolic tangent activation and was followed by a max pooling layer for downsampling. These layers were followed by a two-layer MLP, the first mapping the weights of the final convolutional layer (576 parameters) to 200 fully connected neurons, and the second layer mapping the 200 down to the 10 output classes.

As with the MLP trials above, the CNN model was trained with 60 000 images from the dataset with stochastic gradient descent and a learning rate of 0.05 per sample; 10 000 images were reserved for testing. The trained CNN had an overall accuracy of 98.39% on the test data, which will again stand as the baseline for accuracy of the trials. Analysis of the algorithm performance is presented in the same format as above, and utilized the same fitness function as in the MLP trials.

Initial Conditions

To begin the analysis I first investigated the weight histogram of the trained CNN. In Figure 5.8 the histogram of the second fully connected layer's weights is presented. The range of weight values in this layer are between -0.3395 and 0.3022, similar to the range seen in the starting weights of the MLP layer. There is a significant concentration of low magnitude weights, though a far lower degree of peakedness is

Figure 5.8: Histogram of Initial Weights

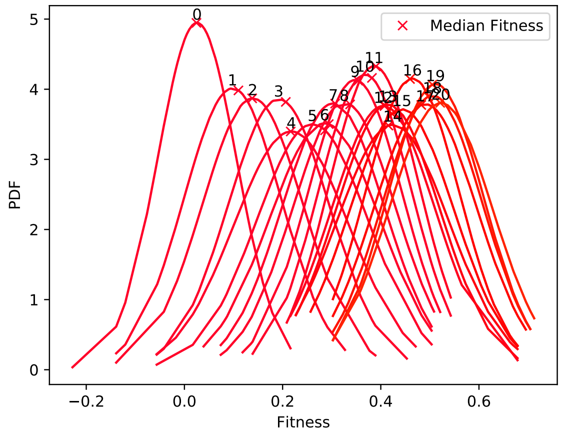


present in the network, with a kurtosis of -0.862 and only 17.95% of weight magnitudes in the bottom 10%. Five trials were done in the same manner as with the MLP, with identical initial damage rates of 10%, 30%, 50%, 70%, 90%, and a population of 50 individuals run for 20 generations. The random damage was applied to all individuals excluding the first individual to allow the population to reconstruct crucial parameters that may have been lost during the initialization.

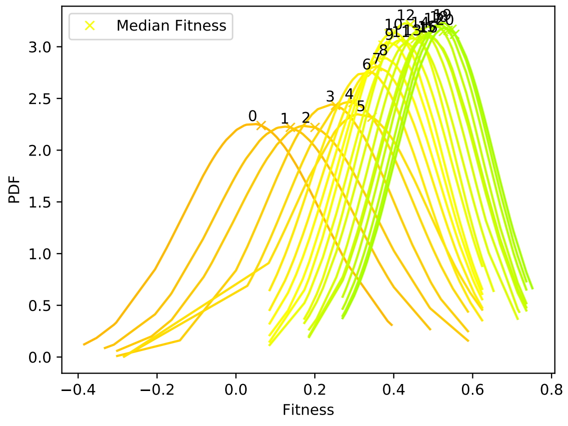
Fitness

The distributions of population fitnesses across generations for the CNN trials are presented in Figure 5.9. Fitness is presented on the X-axis, with relative likelihood presented on the Y-axis, communicated via the probability density function (PDF) of the generation fitnesses. For each distribution, the generation index is printed at its peak.

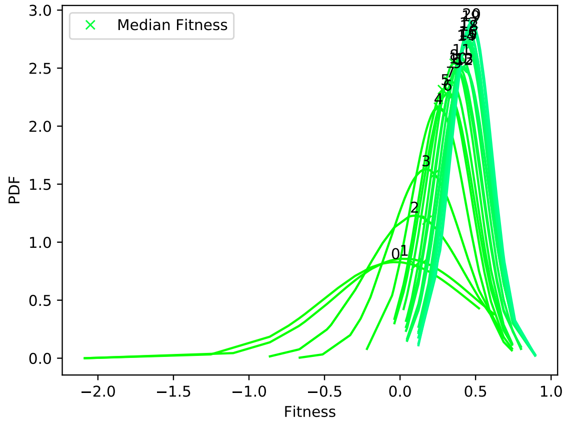
Figure 5.9: Population Fitness Distribution per Generation



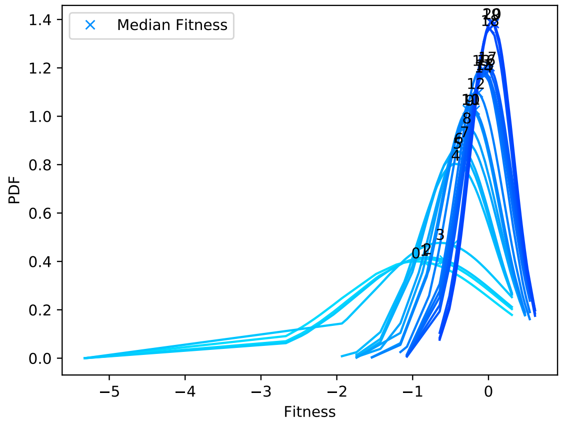
(a) 10%



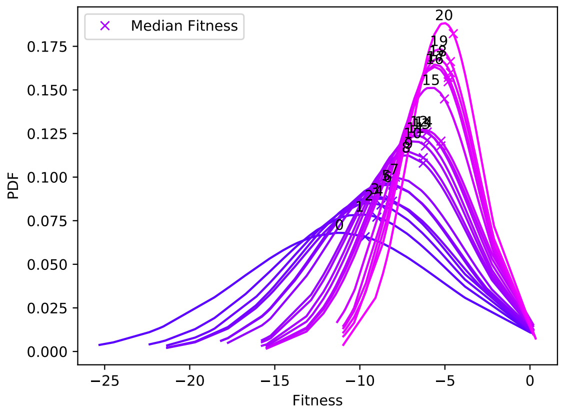
(b) 30%



(c) 50%



(d) 70%



(e) 90%

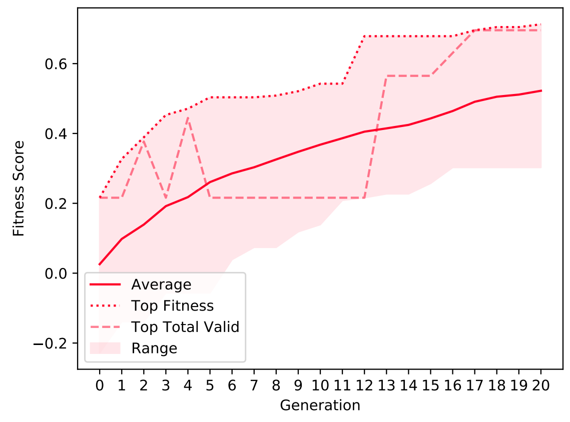
The 10% trial shows a steady, rightward–bound trend towards greater fitness with a strong starting generation, ranging between −0.228 to 0.216, and 0.303 to 0.713 in the final generation. In the 30%–90% trials, it can be observed that as initial damage rate increases, it becomes progressively harder to breed out over-damaged solutions. This can be see through the fitness range of the final generation of each trial, presented in Table 5.1.

Table 5.1: Summary of Initial and Final Population Fitness

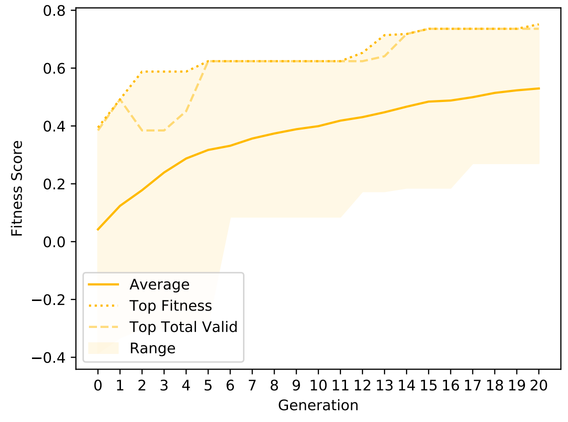
Damage Rate	**Fitness**					
	Initial			**Final**		
	min	mean	max	min	mean	max
10%	−0.2280	0.0257	0.2160	0.3025	0.5223	0.7125
30%	−0.3845	0.0424	0.3945	0.2705	0.5296	0.7515
50%	−2.0870	−0.0291	0.5225	0.1220	0.4735	0.8940
70%	−5.3250	−0.9569	0.3080	−0.6440	0.0472	0.6135
90%	−25.3005	−11.2025	0.0000	−10.9710	−5.0511	0.3340

The number of individuals with non-negative fitness in the final generations of the trials clearly indicates the threshold beyond which damage is solely detrimental to the population. In the 10% to 50% trial, all individuals in the final generation of the populations successfully evolved positive fitness values. The 70% initial random damage trial was able to generate positive fitness for 29 of its 50 individuals in the final generation. The 90% trial had only a single individual of positive fitness in the final generation; unsurprisingly this was the undamaged individual, where the rest of the individuals had an average fitness of -5.161.

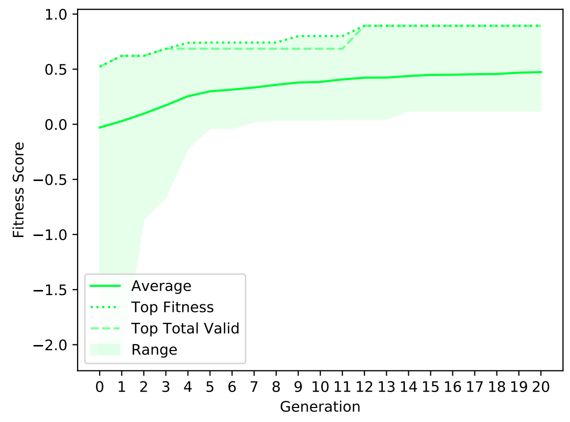
Figure 5.10: Population Fitnesses per Generation



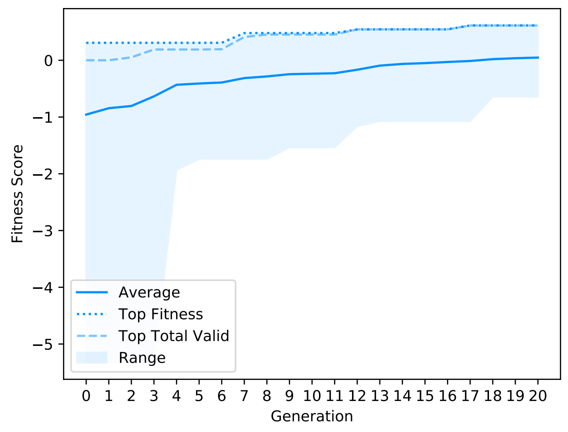
(a) 10%



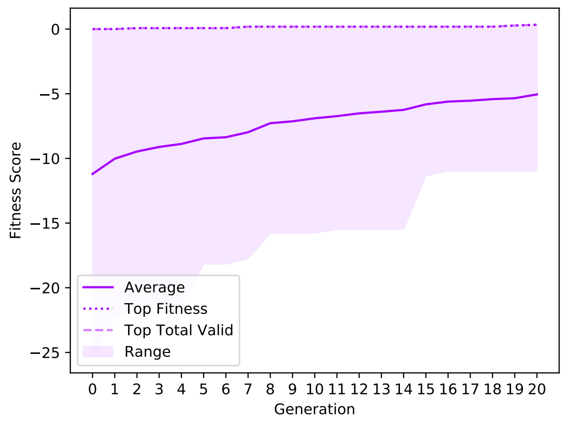
(b) 30%



(c) 50%



(d) 70%



(e) 90%

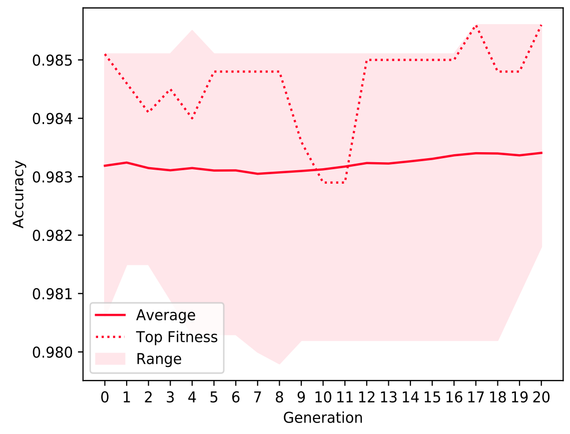
Overall, solutions were evolved more reliably than for the MLP across all trials, having shorter, higher valued fitness ranges and greater probability densities per generation. It is also likely that, given more generations, the trials would have shown further consolidation as elitism would have guarded the high accuracy solutions.

The fitnesses are also presented as a time-series in Figure 5.10, with generation on the X-axis and fitness on the Y-axis. Observing these figure, it can be noted that the algorithm rapidly found a set of prunable parameters in the first few generations with sparse improvements thereafter for the MLP, whereas the CNN trials show a gradual, continued improvement across the generations. All trials, with the exception of the 90% trial show a separation of the top accuracy and top fitness solutions across the generations, indicating that the algorithm favourably trades compression for accuracy. To better view the individuals in the populations, the lineages across generations are presented in Figure A.4. Additionally, it can be seen that poor initial solutions are bred out much more successfully than the population fitnesses of the MLP trials due to the significantly lower number of parameters.

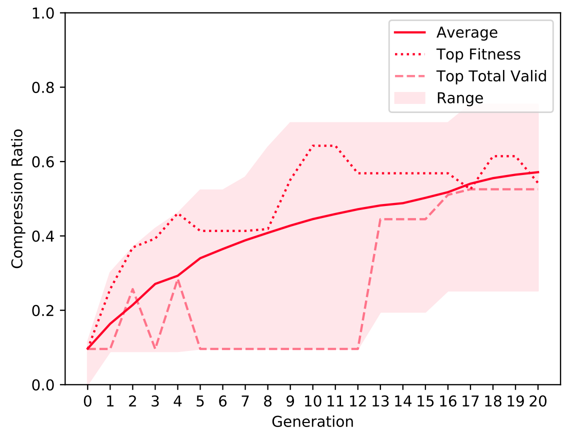
Accuracy and Compression

Next, I present analysis of the progression of accuracy and compression in the populations per generation, presented in Figure 5.11. For each trial, the figures for accuracy and compression are displayed side-by-side, their respective components presented on the Y-axis, and generation presented on the X-axis.

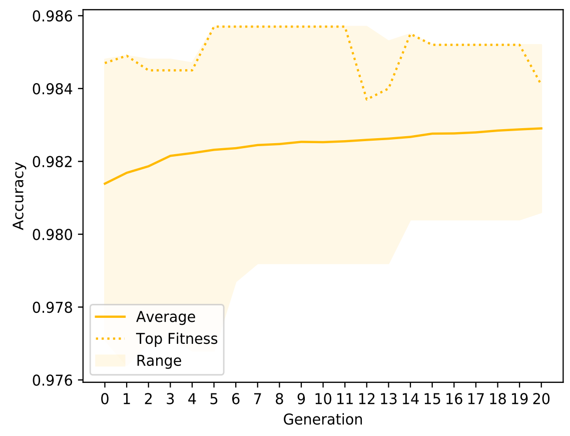
Figure 5.11: Population Accuracy & Compression per Generation



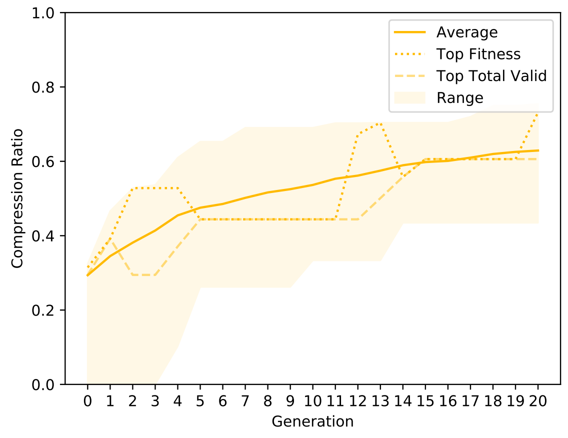
(a) 10% Accuracy



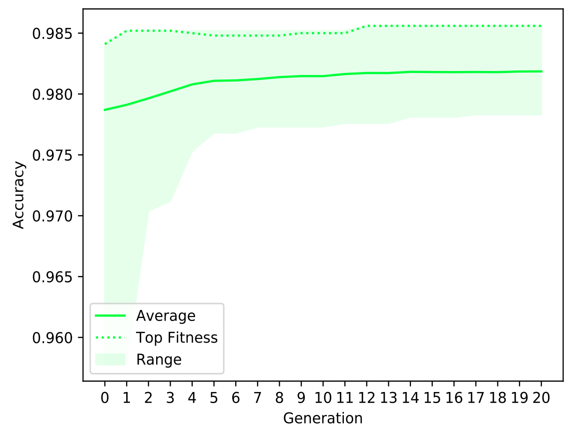
(b) 10% Compression



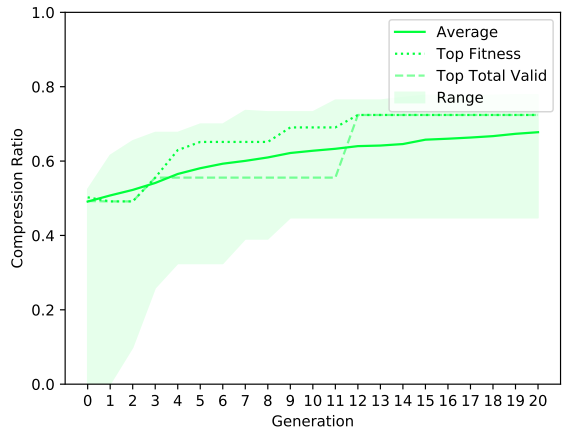
(c) 30% Accuracy



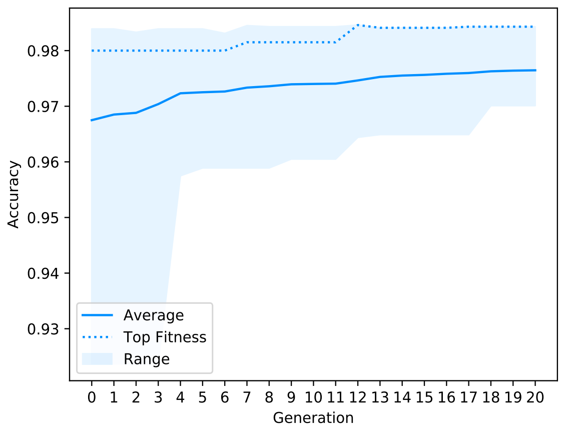
(d) 30% Compression



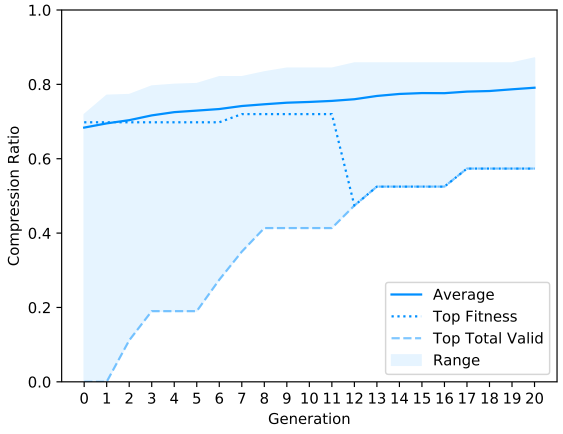
(e) 50% Accuracy



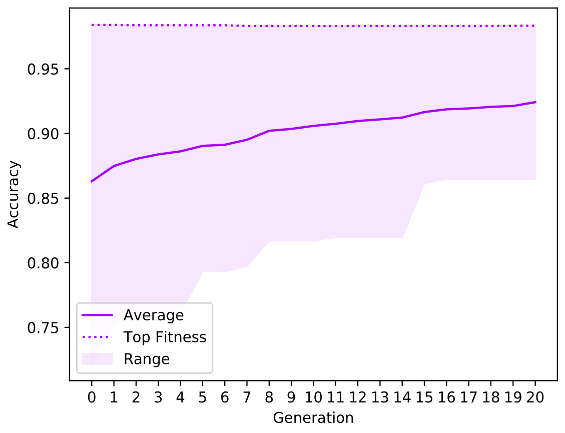
(f) 50% Compression



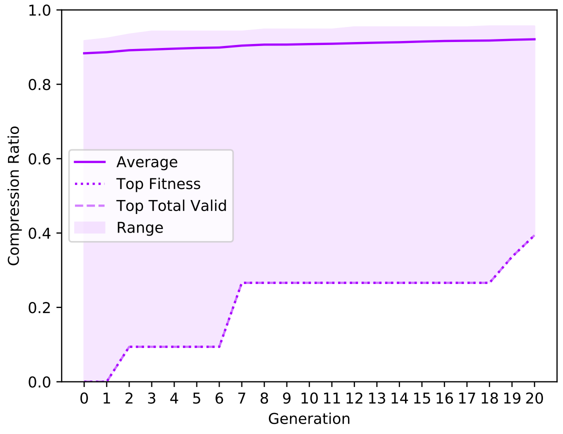
(g) 70% Accuracy



(h) 70% Compression



(i) 90% Accuracy



(j) 90% Compression

Generally, it can be observed that the 30%–90% trials present a visible increase in accuracy across the generation, due to their higher initial random damage and the corresponding penalty to model accuracy. Observing the minimum initial accuracy of the trials, trials 10%–70% show gradual decline until 90% where a substantial drop occurs. Surprisingly, even the population with 90% random weight pruning had an average 86.3% accuracy in the first generation. Furthermore, for the 70% and 90% trials, the maximum accuracy in the starting generation can be seen to be the undamaged individual, meaning that none of the individuals with random damage for those populations resulted in an increase in accuracy. This is contrary to the 10%–50% trials, where all individuals in the initial generation possessed test-set accuracy greater than that of the starting weights.

Table 5.2: Summary of Initial and Final Population Compression & Accuracy

DR	**Accuracy (%)**						**Compression (%)**				
	Initial			**Final**			**Initial**		**Final**		
	min	mean	max	min	mean	max	mean	max	min	mean	max
10%	98.06	98.32	98.51	98.18	98.34	98.56	9.69	11.50	25.25	57.15	75.35
30%	97.71	98.14	98.48	98.06	98.29	98.52	29.36	32.45	43.45	62.90	75.45
50%	95.78	97.87	98.42	97.83	98.19	98.56	49.11	52.40	44.75	67.77	77.85
70%	92.37	96.75	98.39	97.01	97.65	98.43	68.37	71.90	57.35	79.10	87.10
90%	72.19	86.30	98.39	86.48	92.42	98.33	88.35	91.70	39.40	92.11	95.65

Minimum compression for all trials is 0%.

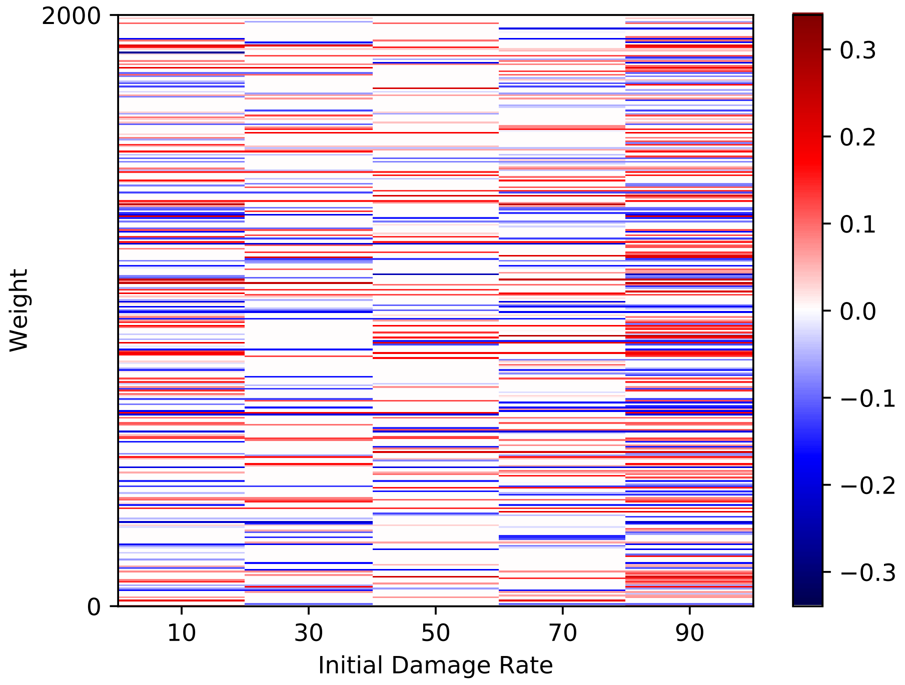
The 10% initial random damage trial shows a very flat, steady progression of average population accuracy, starting from 98.319% and ending at 98.341%, visible in Figure 5.11a and in Table 5.2. The average accuracy across all generations deviates from the starting accuracy of 98.39% within a tight envelope, from −0.085% to −0.049%. Compression begins at 9.69%, as expected with 10% random damage for 95% of the population, shedding an average 2.373% parameters per generation, with the final generation having an average 57.15% compression. The individual with the highest compression pruned 75.35% of its parameters for a 0.12% loss in accuracy, and the least compressed solution pruned 25.25% of its parameters for a gain in accuracy of 0.05%. This presents an interesting view on the degree of redundancy in the model, where a 50.1% difference in parameters accounts for only 0.17% model accuracy. The highest fitness solution pruned 54.25% parameters and yielded an accuracy of 98.56%.

In the 10% trial, the population develops 2.35% pruning and 0.001% increase in accuracy per generation, excluding the undamaged individual. This trend becomes exponential with the damage rate, with 1.64% pruning and 0.008% gain in accuracy for the 30% trial, 0.91% pruning and 0.016% gain in accuracy for the 50% trial, 0.49% pruning and 0.046% gain in accuracy for the 70% trial, and 0.15% pruning and 0.312% gain in accuracy the 90% trial. The same trade-off between compression and accuracy can be seen in the final generation of the other trials, with a 32.00% difference between the minimum and maximum compressed solutions of the 30% trial constituting a loss in accuracy of 0.22%, and 33.10% difference constituting a loss of 0.26% accuracy for the 50% trial. Interesting to note is that the best fit solutions of the 30% and 50% trials eliminated 73.15% and 72.40% of the layer weights, respectively, 54.1% of which were removed in common. From Table 5.2, we can see that the 10%–50% trials did not find any solutions beyond approximately 75% pruning. The 70% and 90% final populations had individuals with higher levels of pruning; however, when considering the individuals with only positive fitnesses the average compression of the final generation for these trials were 78.47% and 39.40% respectively. Additionally, the average compression and accuracy of all positive fitness solutions in the final generation of the trials was 65.03% and 98.2%, respectively; this results in an average fitness of 0.46, trading 0.0029% accuracy per percent compression, or 0.000095% per pruned parameter.

Heatmap of Trial Solutions

The heatmap of the weights of the highest fitness solution presented in Figure 5.12, with the weights of the layer flattened along the Y-axis, and initial damage rates along the X-axis. Once again, we can see that multiple pruning solutions can yield similar results. Additionally, the similarity in compression of the 30% and 50% trials noted in the section above can be observed here. Across all solutions, 9.45% of the parameters were pruned in common.

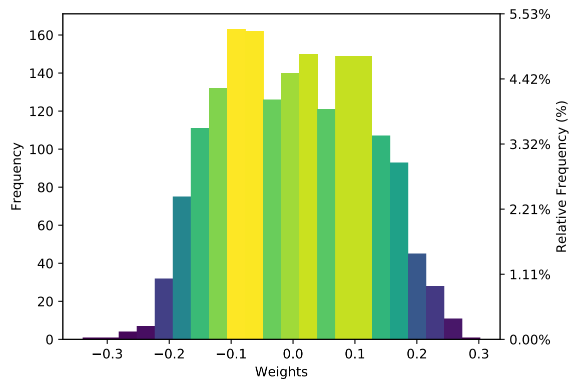
Figure 5.12: Heatmap of Final Weights



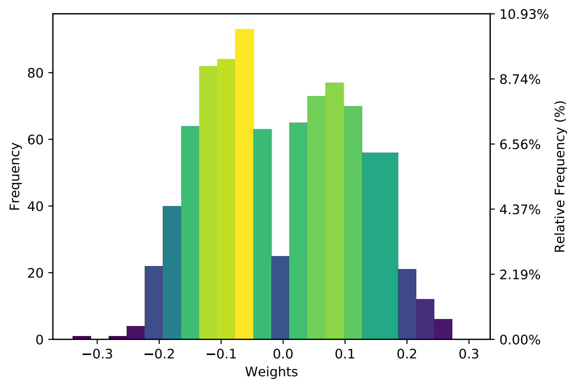
Histogram of Trial Solutions

Next, in Figure 5.13, I present the histogram of the initial and final weights of the best solution per trial. Please note that zero-valued weights are omitted.

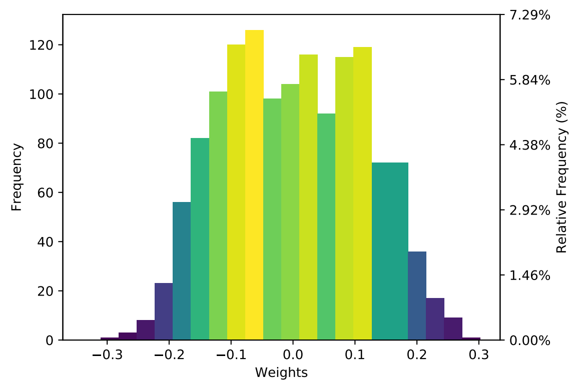
Figure 5.13: Top Fitness Solution Histograms per Trial



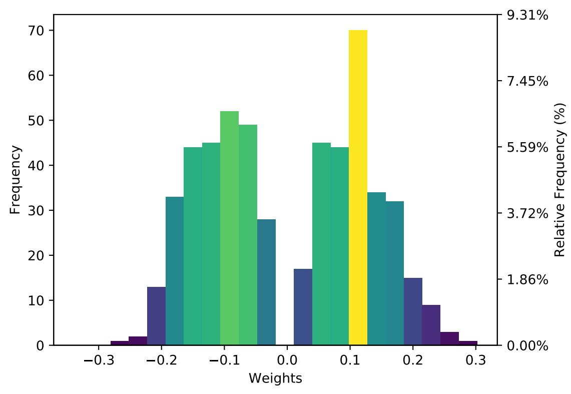
(a) 10% Initial Weight Histogram



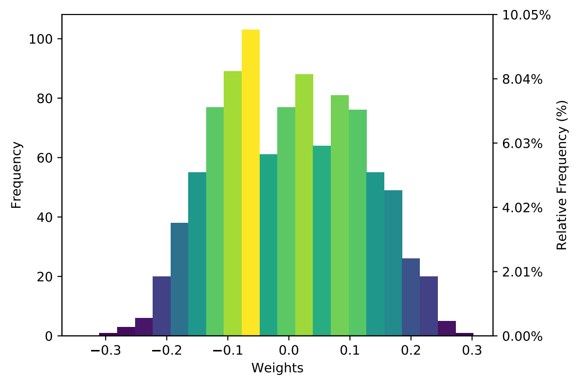
(b) 10% Final Weight Histogram



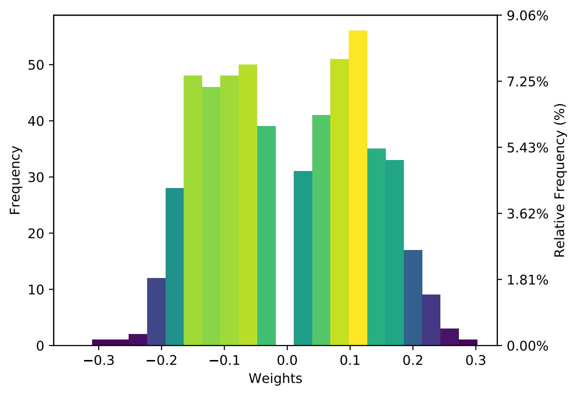
(c) 30% Initial Weight Histogram



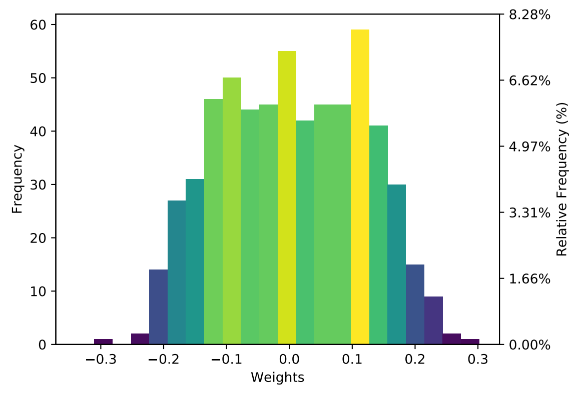
(d) 30% Final Weight Histogram



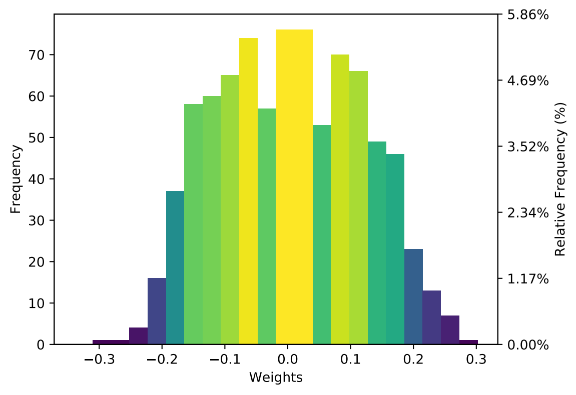
(e) 50% Initial Weight Histogram



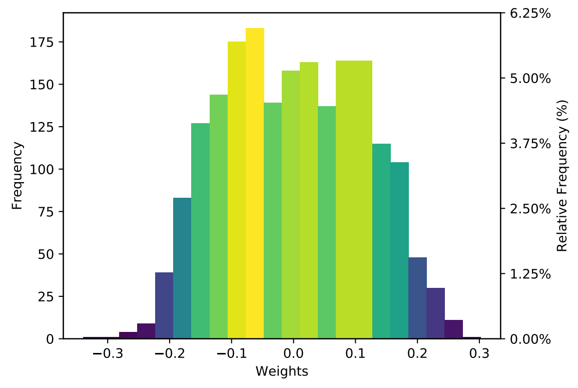
(f) 50% Final Weight Histogram



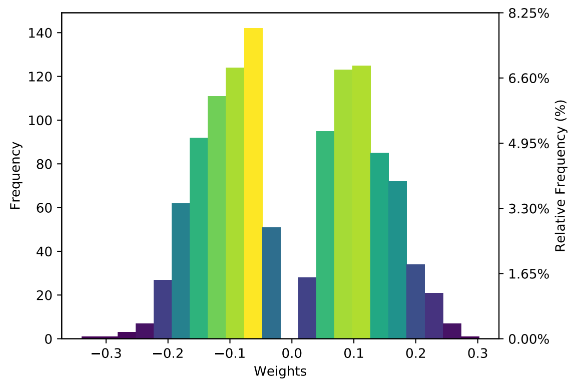
(g) 70% Initial Weight Histogram



(h) 70% Final Weight Histogram



(i) 90% Initial Weight Histogram



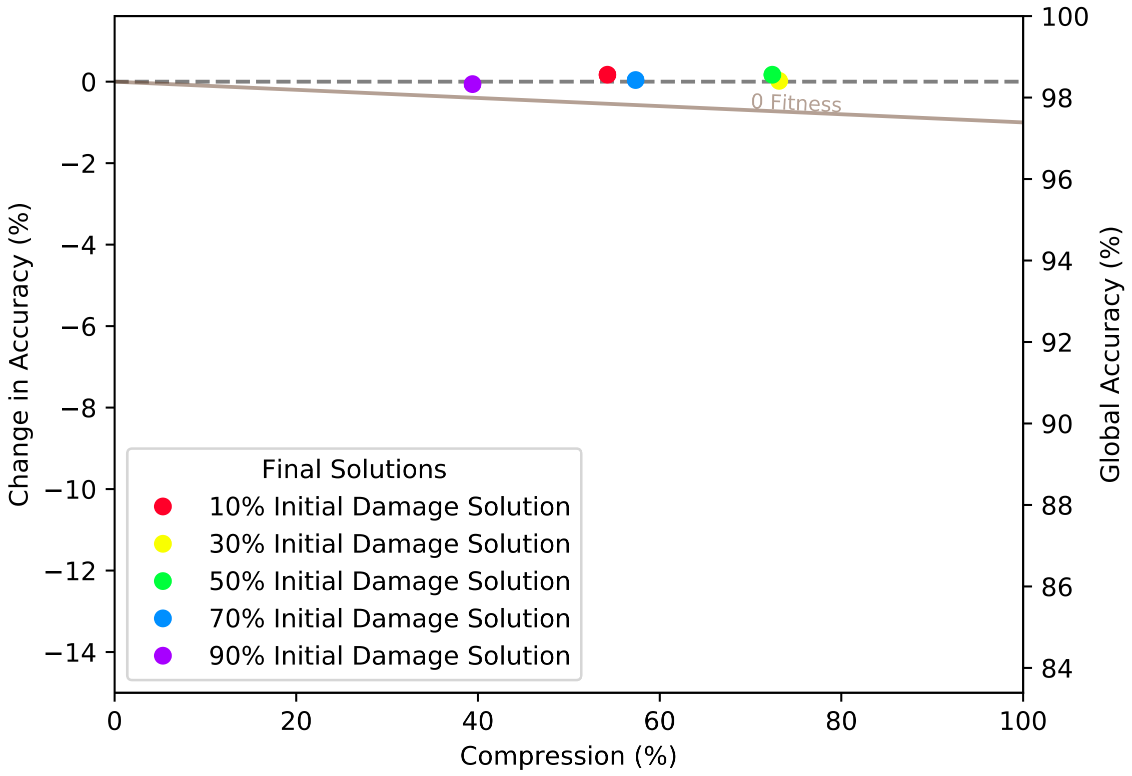
(j) 90% Final Weight Histogram

The 10%, 30%, 50%, and 70% best solutions all begin as randomly damaged individuals, but the 90% starts as the undamaged individual. This shows again that the very high initial damage rates are unsuitable starting points as they incur too much damage and the population wastes time trying to rebuild the lost parameters. From looking at the 90% population figure presented in Figure A.4, it can be observed that the undamaged individual gradually evolved some pruning, while the other individuals attempted to restore their lost information. The 70% trial also shows many individuals around the -1 fitness mark at the start, but are able to evolve greater accuracy with increasing compression, presumably removing parameters which most contribute to test error. The final weights for all trials show the same significant pruning of weights, shown clearly through the large central bin containing all the near-zero weights in the histograms.

Solution Quality

Finally, in Figure 5.14 I present the the trial solutions overlaid on the fitness function plot.

Figure 5.14: CNN Trial Final Fitnesses



Once again, for all trials the algorithm provides solutions that successfully and substantially prune the layer and increase the test set accuracy of the network. Across the trials, the 50% initial damage rate experiment achieved the highest fitness, and second highest degree of pruning. The 50% trial achieved 72.40% compression of the layer while increasing accuracy by a modest 0.17%. This was closely followed by the

30% trial, which achieved 73.15% compression of the layer while increasing accuracy by 0.02%. The 10% trial achieved the same increase in accuracy as the 50% trial, though with a considerably lower degree of compression of 54.25%. The 90% trial had the lowest fitness of all trials, with 39.40% compression with a loss in accuracy of 0.06%. The best solution across all CNN trials had a fitness of 0.894, moving the accuracy from 98.39% to 98.56% with 72.40% compression. For comparison, the best MLP solution had a fitness of 2, improving the accuracy from 94.59% to 95.69% with 89.6% compression. To the success of the algorithm, it was able to perform substantial repair of the population, raising the average loss in accuracy from 12.33% in generation 0, to only 0.061% loss in the final generation. An excellent example would be in the worst individual in the starting generation of the 90% trial, which had an accuracy of 72.19% and 89.95% random compression, and ended with 89.02% accuracy with 93.50% compression.

Overall, when looking at the progression of individuals in the population it can be again concluded that each layer has a critical point beyond which random damage is detrimental to initialization. This is visible through lineages of individuals in the 70%–90% initial damage rate MLP trials and 90% CNN trial in Appendix A, where the undamaged individual in each population was the sole individual able to reach a positive fitness. The success of the 10%–30% MLP and 10%–50% CNN trials show that initial random damage can be useful in initializing the populations, most visible through the success of the 50% initial random damage CNN trial. Through the experiments performed, both models exhibit substantial redundancy that can be removed to the benefit of accuracy and footprint without retraining or finetuning.

5.2 Energy-based Filter Pruning

The filter pruning algorithm was applied to both VGG and ResNet [Janjic et al., 2018] with results presented in section 5.2. Experiments performed on the VGG-16 model post-filter pruning to investigate interaction with dropout in the fully connected layers are presented in section 5.2.1. Additionally, the results of the filter pruning algorithm are presented in fitness-function style in Figures A.7 and A.8.

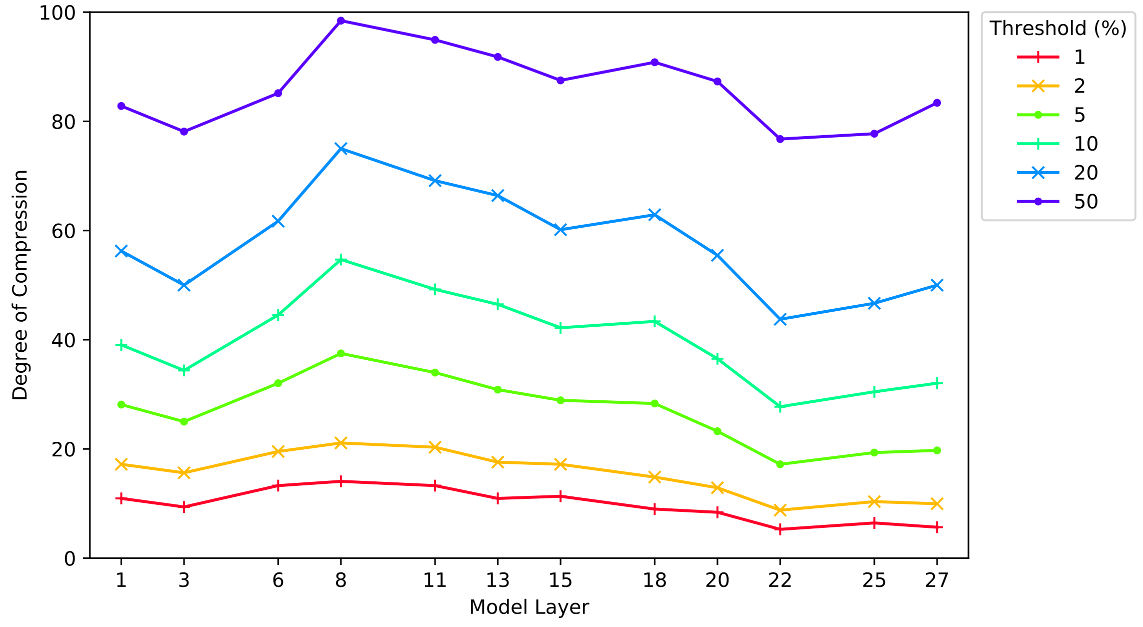
5.2.1 VGG-16

The following sections show the results of running the filter pruning algorithm on a pretrained VGG model, providing analysis on the compression achieved and the accuracy post pruning. An additional trial was conducted to explore the relationship between filter pruning and the fully connected layer in the model.

Model compression

Figure 5.15 depicts the degree of compression resulting from the filter pruning algorithm applied per layer on the pretrained VGG model. In this figure, and in subsequent figures, the model is laid on its side such that the X-axis represents the layers of the model, where 1 is the first layer of the network. As models are not purely comprised of spatial convolution layers, any spatial pooling, dropout, or fully connected layers are skipped by the algorithm but their indices maintained. As a result, the data points on the figures represent the indices of the convolutional layers in the model. The Y-axis represents the ratio of pruned filters with respect to the original number of filters in that layer; this can also be thought of as the degree of compression achieved by the pruning algorithm. All points for a given minimum energy threshold are joined with a line, the values of which can be found in the legend. For example, a 1% energy threshold would mean that for each convolutional layer of the model, when sorted, all filters in the model whose energy contributes less than 1% to the total cumulative sum energy are removed from that layer, and their weights in the following convolutional layer are also removed.

Figure 5.15: VGG Compression Ratio per Energy Threshold



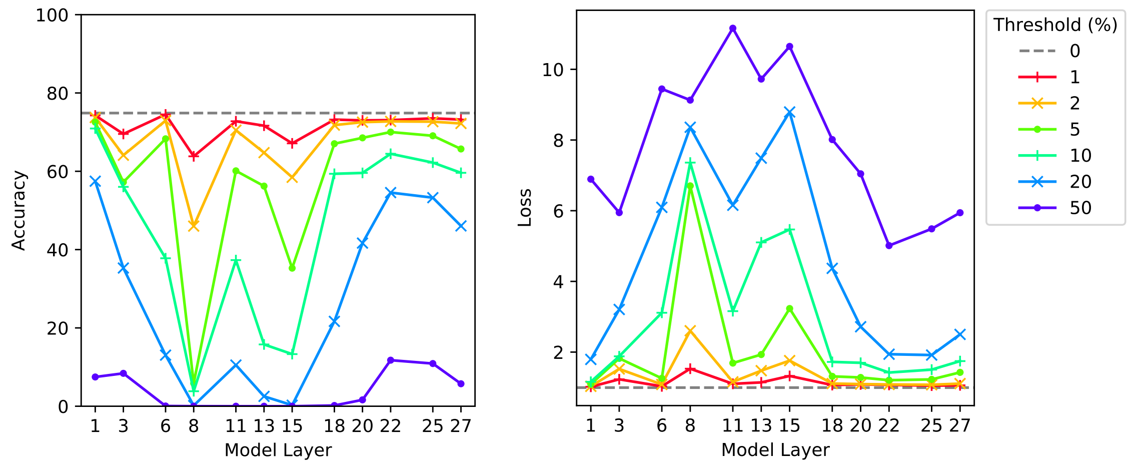
Surprisingly, even the lowest energy threshold of 1% removes almost one-tenth of the filters on average when applied to each convolutional layer independently.

When looking at the average ratio of compression each layer undergoes over the different cumulative energy thresholds, we see that the number of filters removed per layer differ significantly.

Since the representation at each level has not been explored, no clear hierarchy has been observed in the contribution per layer; this is to say there is no clear bottom-up or top-down hierarchy. Further investigation of this is difficult as a full understanding of network hierarchy is dependent on a multitude of factors; these include structure, activation type, initialization, and curriculum to name a few. Furthermore, the representation held by an arbitrary neuron can vary as well and is not predetermined.

Effect on Accuracy

Figure 5.16: VGG Accuracy & Loss per Energy Threshold



In the experiment performed on a pretrained VGG16 network, we can see the effects of removing the low energy filters, presented in Figure 5.16, on the loss and top-1 classification accuracy of the model, with no post-pruning retraining. Comparing the pruning curves with the loss curves, we can draw several interesting insights. Many intervals share a closely matched slope at the noticeable peaks in layer 8 and 15 that become pronounced at higher energy thresholds. The trend of higher loss with higher pruning plainly indicates that the degree of pruning is too large and is highly detrimental to the model. However, this correlation, especially at layers 8 and 15 where the loss becomes extreme and the model is completely broken, shows a very important attribute of those layers - most or all filters contribute to the model's function. Looking at the most extreme case of layer 15, we see that the model's loss during testing suffers greatly even from pruning the filters which contribute the lowest 1% cumulative sum energy. This shows that the model has saturated the filters of this layer. We also see that layer 15 is subject to the most pruning for every energy threshold. Such cases where there is high pruning and high loss indicate that the filters of the layer are closely related in their energy; consequently, when the energy threshold is applied to the cumulative sum energy of the layer, many valid filters are pruned. Therefore, layers with high pruning and high loss can be considered saturated and pruning is not recommended.

Conversely, cases where there is high pruning and low loss in error indicate that the layer has great redundancy and a large number of filters can be removed without incurring much, if any, error. Consider the loss seen across all energy thresholds for layer 1 - even in the exaggerated case of 50% energy pruning the model incurs a loss in accuracy of only 5 points despite over 81% of the filters of that layer being pruned. This clearly shows that the first layer, where basic filters relevant to the input data are learned [LeCun et al., 1998a], contains a phenomenal degree of redundancy. Considering the filters in this layer are 3x3 convolutions it may be the case that the effect is less prevalent with larger filter sizes.

Table 5.3: Effect of Varying Rates of Dropout on Model Accuracy

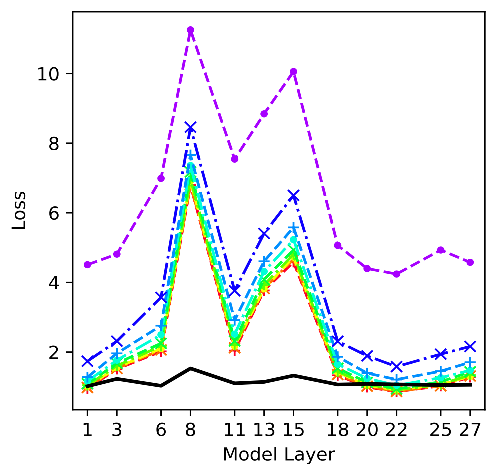
$P_{\textbf{dropout}}$	**Loss**	**Accuracy**	$P_{\textbf{dropout}}$	**Loss**	**Accuracy**
0.0	1.334	68.14	0.5	1.334	67.84
0.1	1.188	70.74	0.6	1.430	66.32
0.2	1.207	70.45	0.7	1.622	63.05
0.3	1.234	69.88	0.8	2.137	55.77
0.4	**1.268**	**69.31**	0.9	4.998	31.45

Interaction with Dropout

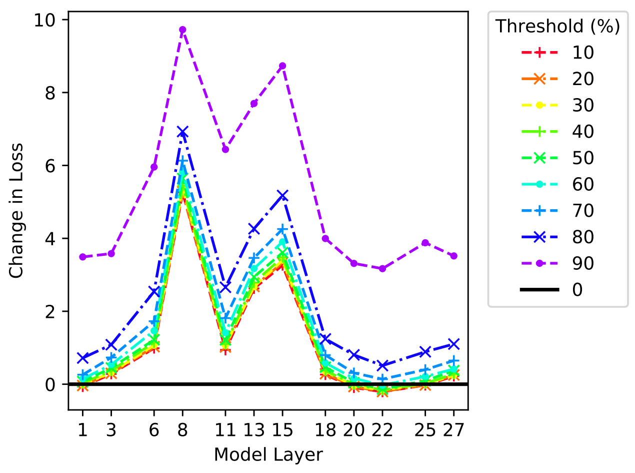
As the VGG model contains dropout connections in its fully connected layer, we can further investigate the interaction of filter pruning on the accuracy of the network per layer. Dropout probability controls the probability of a neuron being disabled for every sample during the training of the model. First, to provide a baseline for comparison, Table 5.3 shows the effect of varying levels of dropout probability enabled on the fully connected layers on the models' test accuracy. As the fully connected layers are attached to the final convolutional layer, the dropout rate here effectively modulates the signaling of the convolutional layer. As observed in section 5.1, the interesting case of disabling neurons without retraining yielding an increase in accuracy is clearly visible again. We can see that as much as 40% dropout in the fully connected layers leads to an increase in model accuracy.

Trials were done to study the effects of 1% and 10% energy-based filter pruning with vary levels of dropout enabled in the fully connected layer, presented in Figure 5.17 and Figure 5.18 respectively. The evaluations were done with the following probabilities of dropout: 10%, 20%, 30%, 40%, 50%, 60%, 70%, 80%, 90%.

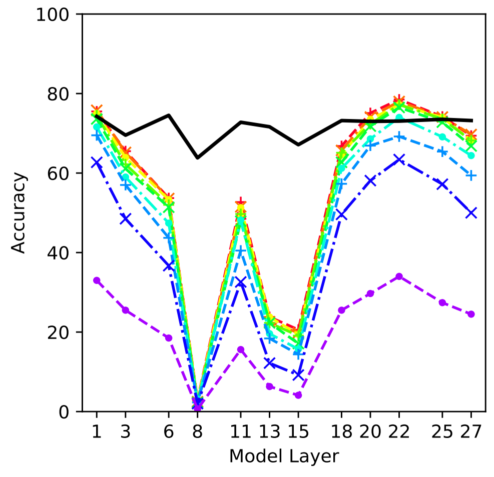
Figure 5.17: Dropout with 1% Energy Pruning



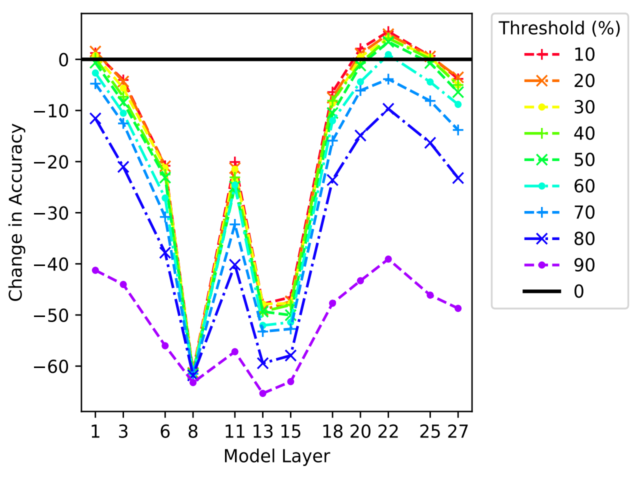
(a) Loss



(b) Change in Loss



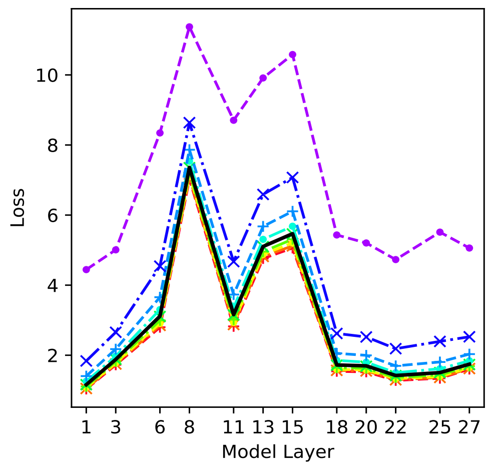
(c) Accuracy



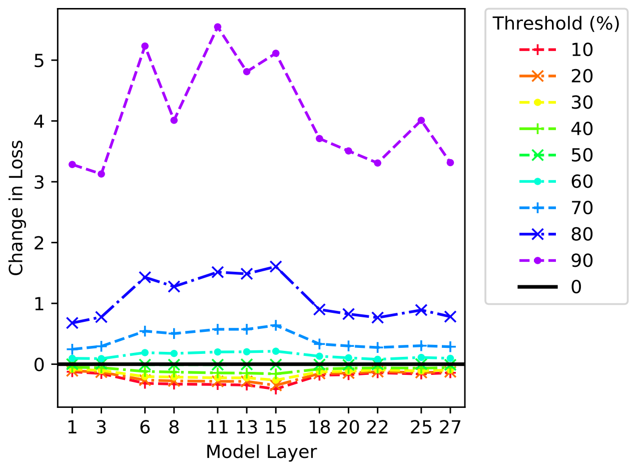
(d) Change in Accuracy

For each trial in Figure 5.17 and Figure 5.18, the loss and accuracy are present on the left, with the rate of dropout represented in the legend. The loss and accuracy of the 1% and 10% VGG filter pruning trials, presented above in the *Effect on Accuracy*

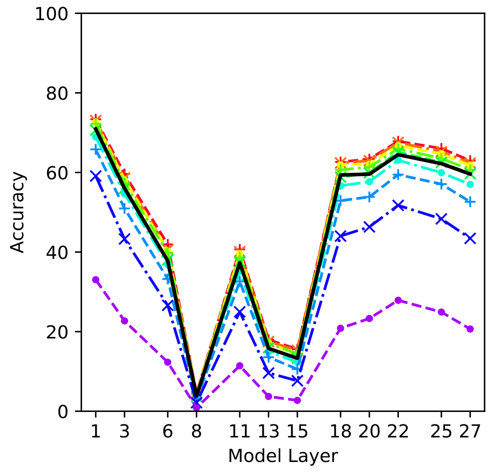
Figure 5.18: Dropout with 10% Energy Pruning



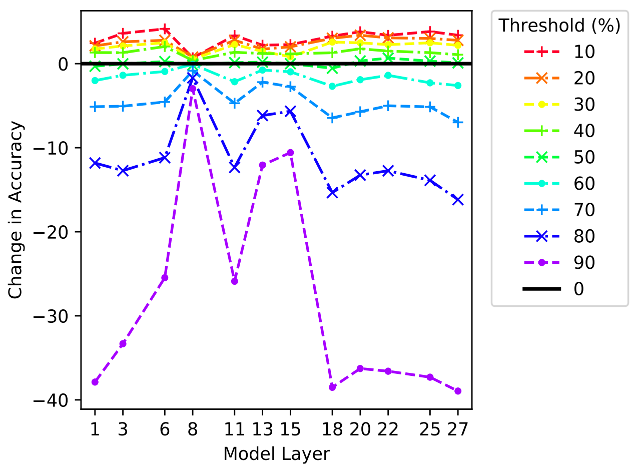
(a) Loss



(b) Change in Loss



(c) Accuracy



(d) Change in Accuracy

subsection, are shown as solid black lines for comparison in their respective plots. The change in loss and accuracy due to the various dropout probabilities with respect to the VGG filter pruning for the 1% and 10% cumulative sum energy thresholds are presented in the plots on the right.

For the 1% filter pruning dropout trials we can see that, in general, the process results in significant damage to the network; however, there are some layers which seem to benefit modestly from the application on a dropout probability of 50% or less being applied. In Figure 5.17a we can see that layers 1, 20, and 22 gain accuracy in experiments where the dropout probability in the fully connected layers is 40% or less.

These cases are far more pronounced in the dropout trials based on 10% filter pruning, presented in Figure 5.18a. In these trials, once again 40% dropout probability rides the edge of improvement. What's significant in this trial is that while the 10% filter pruning trials inflict significant damage, especially when compared to lighter degrees of pruning such as at the 1% level, the application of light dropout during testing shows a universal gain across all pruned layers. This gain is also consistent with the accuracy results of the non-filter pruned model presented in Table 5.3.

Summary of VGG-16 filter pruning results and solution quality are provided following the ResNet results in subsection 5.2.2.

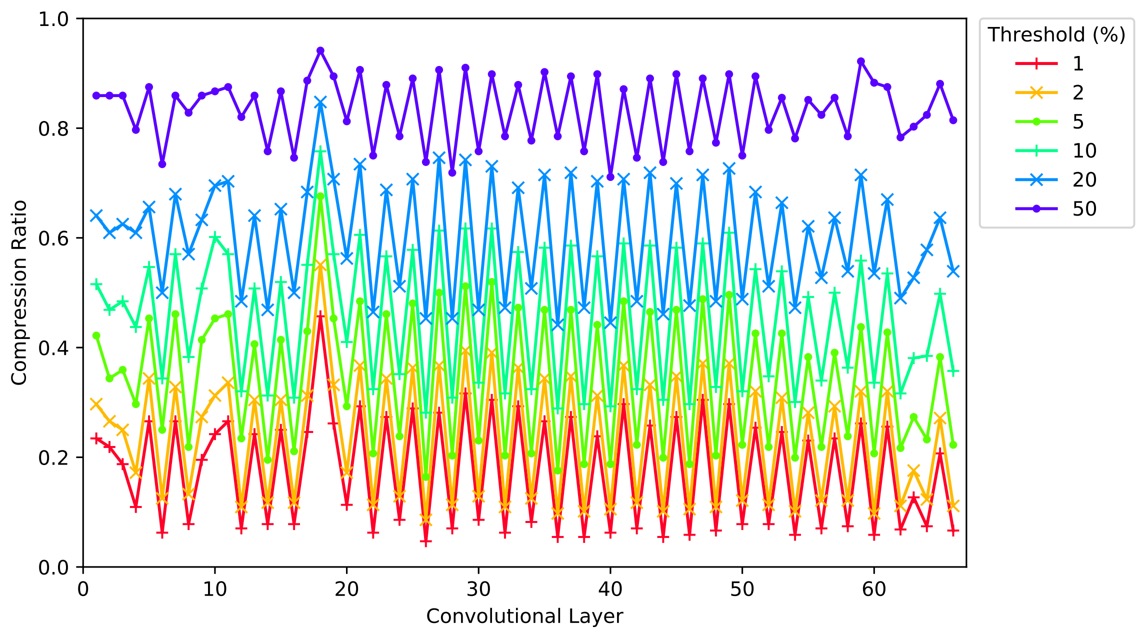
5.2.2 ResNet-101

The following sections show the results of running the filter pruning algorithm on a pretrained ResNet model, providing analysis on the compression achieved and the accuracy post pruning. The algorithm will be applied on each residual module, removing filters from one layer and their corresponding feature dimensions in the following layer. The following cumulative sum filter energy thresholds were evaluated: 1%, 2%, 5%, 10%, 20%, and 50%.

Model compression

Figure 5.19 depicts the degree of compression resultant of the filter pruning algorithm applied per layer on the pretrained ResNet model.

Figure 5.19: ResNet Compression per Energy Threshold



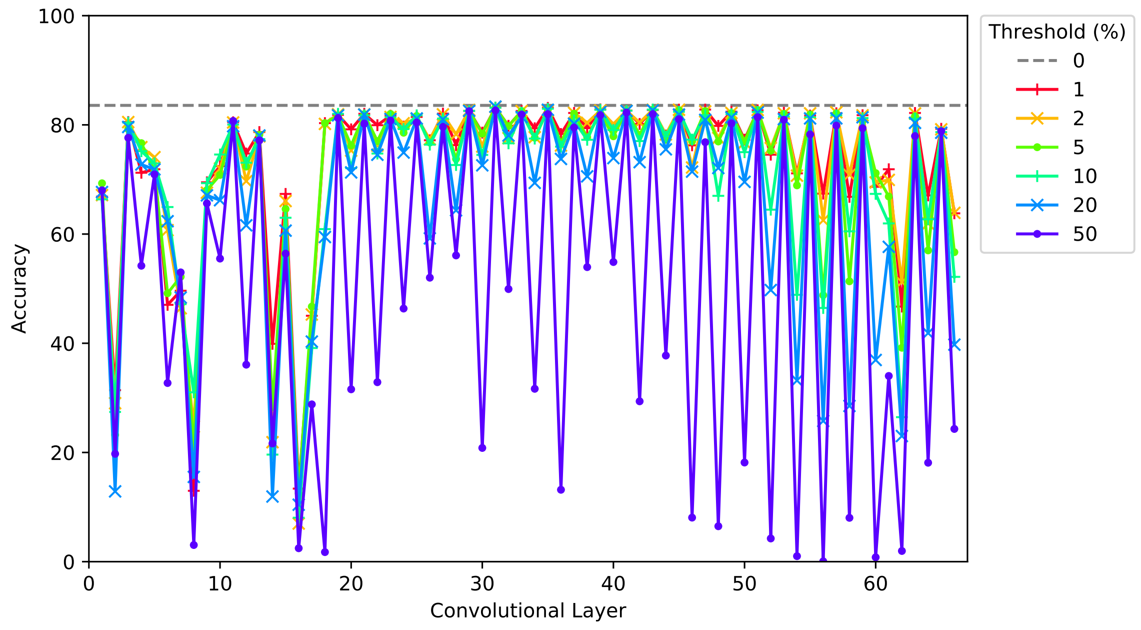
In Figure 5.19 and following figures, the X-axis represents the layers of the model, where 1 is the first layer of the model, and only spatial convolutional layers are analyzed and plotted. The Y-axis here represents the ratio of pruned filters with respect to the original number of filters in that layer. A clear *saw-tooth* trend appears within each residual module, where the first convolutional layer often experiences greater compression per threshold than its following layer. The residual modules consist of three ReLU-activated batch-normalized spatial convolutional layers. Within each residual module, the first convolutional layer acts to reduce the input plane, the second convolutional layer maintains the feature size, in some cases striding convolutions, with the final convolutional layer expanding the output plane back up to match the size of the input plane passed to the module.

Effect on Accuracy

As with the VGG trials, the filter energies are measured, normalized, sorted, and pruned based on a threshold of the cumulative sum energies of the filters.

The general trend seen in Figure 5.19 suggests that these first convolutional layers may contain a large degree of low-energy filters. In Figure 5.20, it can be observed that for the the majority of the cases, the model can incur a high degree of pruning in the first convolutional layer without any substantial shift in performance. Unlike the VGG trials, there are several convolutional layers whose pruning causes catastrophic failure of the model at the 1% energy threshold. Consider the eighth convolutional layer in the Figure 5.20, which at the 1% energy threshold undergoes a pruning of 7.8% of its filters but results in the model accuracy collapsing down 12.98%.

Figure 5.20: ResNet Accuracy per Energy Threshold



One possible explanation for why pruning certain convolutional layers compromises model accuracy so severely is that when the filters are pruned and their corresponding weights removed from the third convolutional layer in the module, downstream modules lose the contribution of these filters.

As seen in the VGG trials, this process can be overzealous for saturated layers that have zero or very few low-energy filters. This is observed in layers with a drastic drop in accuracy post-pruning, as their compression per energy threshold is low as well, such as in layer 2 or 8. Furthermore, the layers that experience little pruning generally result in massive loss in model accuracy post-pruning.

Another interesting example is the 19th convolutional layer which for every energy threshold, it undergoes a high degree of pruning, yet incurs next to no loss in accuracy. This layer, at the 1% energy threshold, undergoes 26.17% compression with a resulting model accuracy of 81.83%. Even at the highest tested threshold level of 50%, the measured accuracy was 81.25% despite the layer being pruned from 256 filters down to only 27 – an 89.45% degree of compression. This layer does not have any special structural significance, nor does it lie at the transition in filter sizes occurring three times at layers 7, 15, and 61.

Table 5.4: ResNet Example Trend Exceptions

Convolutional Layer	**Filters**		**Compression**	**Accuracy**
	Preserved	**Total**		
16 – B_2^8	236	256	7.81%	13.38%
17 – B_1^9	193	256	24.61%	45.04%
18 – B_2^9	139	256	45.70%	80.32%
19 – B_1^{10}	189	256	26.17%	81.83%

Convolutional Layer refers to the flattened index of the convolutional layer of the ResNet model, with the superscript and subscript of B indicating the index of the residual block in the model and the index of the convolutional layer within the block, respectively.

Table 5.4 highlights the exceptions to the *saw-tooth* trend first noted in Figure 5.19. In this table, Block indicates the number of the residual module in the model, Layer indicates the layer within that particular residual module. The sequential index of each convolutional layer is also given for to maintain correspondence with Figure 5.19 and Figure 5.20. Here we can observe the opposite effect to the general *saw-tooth* trend where the second convolutional layer undergoes much higher compression than the first convolutional layer. However, in these cases both layers experience very little pruning as the majority of their filters pass all thresholds.

Summary

From the experiments run on the VGG and ResNet models, it is clear that extensive redundancy exists in both models. Overall, the Energy-based Filter Pruning algorithm succeeds at identifying and pruning redundancy per layer. VGG-16's 4$^{\text{th}}$ (of 12) convolutional layer, and ResNet-101's 18$^{\text{th}}$ (of 101) convolutional layer were highly sensitive to pruning. Another interesting observation is that following these pruning–sensitive sections of the models, there are a number of layers which appear to have a great deal of redundancy and can incur a significant degree of compression with negligible loss in accuracy. Due to the extensive pruning applicable to these layers, it may be the case that the entire layers can be removed without significant loss in accuracy. For VGG this trend is visible in the last 5 layers, and in ResNet convolutional layers 19–59 show this trend of robustness.

The trials on the VGG-16 network showed that the 1% energy threshold removed 9.83% of the filters in each layer with an average loss in accuracy of 3.23% (2.52% excluding layer 8). The 10% energy threshold removed 40.06% of filters per layers, while the loss in accuracy incurred was 16.88% (9.48% excluding layer 8). The ResNet trials yielded fascinating results, showing that for many convolutional layers even the most aggressive energy threshold resulted in a high degree of compression at negligible loss in accuracy. At the 50% energy threshold, it was possible to prune 87.94% from 25 of the 66 convolutional layers for a single digit loss in accuracy of 3.30% per layer. VGG in comparison, experienced catastrophic failure for all layers at the 50% energy threshold, with the lowest single loss in accuracy being 63.11%.

Chapter 6

Other Approaches

A particularly clear and important takeaway from the presented trials is that there exists a significant amount of information in the models which can be removed without a large effect on the network's ability to function. This work has also shown the ability to remove large portions of this excess structure for the purpose of testing and general use of the models.

Below, two additional ideas are discussed pertaining to the pruning of networks that extend upon the trials, but were, however, limited in their success.

6.1 Iterative Pruning without Retraining

Across the trials presented, the networks appear to incur minimal loss from the pruning of individual layers; however, applying iterative pruning without retraining appears to accumulate egregious error on all tested energy thresholds. Trials were run on both VGG and RESNET, removing filters at the 1% cumulative sum energy level targeting layers that were known from the individual pruning trials to incur a loss in accuracy of less than 2%. In all trials, the models experienced catastrophic failure with single digit accuracy, after a few consecutively pruned layers. This shows that the small error incurred from the pruning must be absorbed or otherwise handled before further pruning can be done.

6.2 Evolving Replacement Filters

As with the Multilayer Perceptron weight pruning optimization, a Genetic Algorithm can be used to evolve filters to replace pruned filters in an attempt to boost network performance without having to perform retraining. The advantage of such a process would be that it only requires a smaller, representative test data set rather than the full training set.

This approach was tried, but ultimately resulted in an optimization that was only able to reconstruct the filters it sought to remove, or generate similarly useless, low-energy filters. It is likely the case that the replacement filters caused instability in subsequent layers not adapted to the input from the newly generated filters. This is to say, the network performed best when left unaltered. Once again, the conclusion is that some degree of retraining is required to wean the network off relying on the vestigial filters, or when introducing new features.

Chapter 7

Conclusion

In this thesis I explored the nature and extent of redundancy in contemporary neural networks. The topic has the innate challenge that neural networks are opaque in how concepts and abstractions are internally represented. To meet this challenge I contributed two novel approaches to pruning both the weights of fully connected layers and the filters of convolutional layers, applicable to all contemporary trained convolutional neural networks.

The results from both algorithms showed that significant pruning can be applied to both layer types with negligible loss, or in some cases a gain, in overall model test set accuracy. The genetic algorithm was applied to a multilayer perceptron and simple convolutional neural networks trained on the MNIST handwritten digit dataset. The experiments showed that for these two models as much as 72.4% and 89.6% of layer parameters can be pruned without retraining, respectively. For both networks, the genetic algorithm was able to simultaneously yield an improvement in test set accuracies of the models of 1.1% and 0.17%, respectively.

The Energy-based Filter Pruning algorithm was applied to the very deep VGG-16 and ResNet-101 models, trained on the much more intensive ImageNet dataset. The filter pruning experiments showed that these models too exhibit significant redundancy in their convolutional layers. VGG layers incurred an average 3.2% loss in accuracy and average 9.83% compression after pruning the lowest 1% energy filters, and as much as 70.93% compression in the first layer for a 3.94% loss in accuracy at the 10% filter energy threshold. ResNet boasted far more impressive compression results, as layers incurred an average 87.94% compression and 3.30% loss in accuracy for 25 of the 66 layers at the 50% energy threshold; 47.82% compression in 41 of the 66 layers with only a single digit loss per layer. Additionally, some saturation points, where layers were sensitive even to the pruning of their lowest 1% energy filters, were noted in both networks. Similarly, sections of the models were found to be particularly robust to pruning, the most extreme case being a layer where near 90% pruning resulted in a 2% drop in model accuracy. From this, it is likely that the excess and deficiency of filters in convolutional layers exists in all modern networks whose structure is fixed before training.

These results and their derived insights add to the body of work for the efficient use of parameters in neural models, and help to inform the architecture and training decisions of models going forward.

Chapter 8

Future Work

While much as been accomplished, there are still many avenues yearning for further attention. Summarized below are two paths readily traversable.

Energy-based Filter Pruning Genetic Algorithm

A key limiting factor in using a Genetic Algorithm is the evaluation time with intensive fitness functions. In the Weight Pruning Genetic Algorithm this cost was small as the networks tested were simple, and the dataset small in number of samples and image size. For larger networks such as VGG and ResNet trained on the substantially larger ImageNet dataset, evaluation would have been very time consuming compared to the time to generate a new population. Additionally, due to the massive size of the models, only one solution can be put into memory and tested, so the population must be evaluated sequentially and not in parallel.

This challenge could be overcome by using an approximated fitness for some or all evaluations. For example, existing perturbation metrics, such as the Taylor series expansion methods presented in LeCun et al. [1990]; Molchanov et al. [2016], could be leveraged to replace the accuracy component of the fitness function while retaining the compression component as it has a time cost to calculate. Alternatively, a randomized subset of the test data could be used.

Additionally, an approximate fitness could be to the benefit of the filter generating genetic algorithm presented in section 6.2. The algorithm could be repurposed from generating replacement filters for low-utility filters, to generating filters which compensate for the error incurred by pruning for the subsequent convolutional layer.

Iterative Energy-based Filter Pruning with Retraining

Based on the Energy-based Filter Pruning algorithm presented in section 4.2, the efforts of iterative pruning presented in section 6.1 could be reattempted with retraining. The retraining could be done either after each pruning step, or once after pruning multiple layers to see if the model could recover after catastrophic failure.

This diverges from the focus of this thesis as it applies retraining, but it is clear that by removing a substantial number of parameters across multiple layers some form of update is required to mitigate the error introduced by the pruning. Additionally, it would be valuable to explore the degree to which pruning error could be absorbed by the other layers of the model, and how to best prune and retrain.

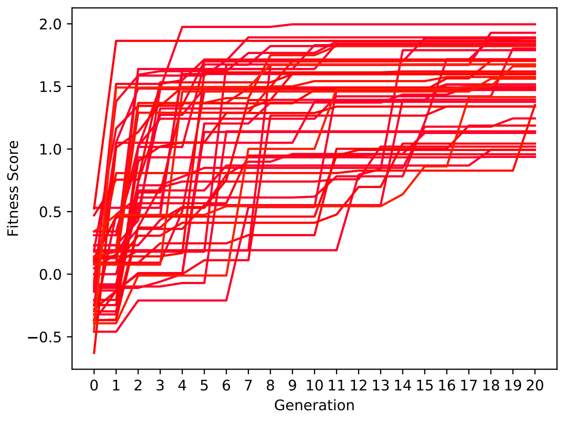
Appendix A

Supplementary Population Figures

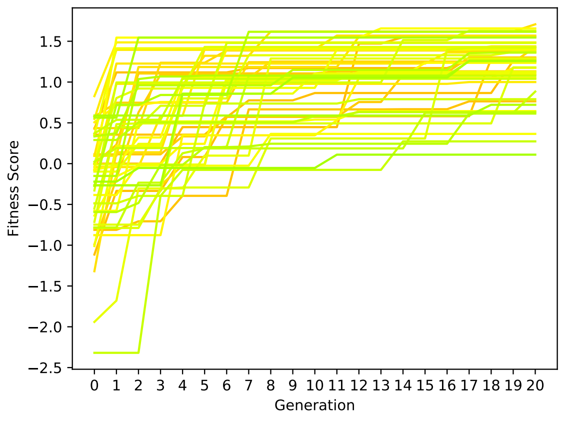
Presented here are supplementary figures for the Weight Pruning Genetic Algorithm in section 5.1. Figures A.1 to A.3 show the fitness, compression, and accuracy of the individuals in the populations of the MLP trials in Figure 5.1.1. Figures A.4 to A.6 show the fitness, compression, and accuracy of the individuals in the populations of the CNN trials in Figure 5.1.2.

Results from the Energy-based Filter Pruning Algorithm trials on VGG-16 and ResNet-101 are presented in the style of the fitness function from the Weight Pruning Genetic Algorithm in Figures A.7 and A.8.

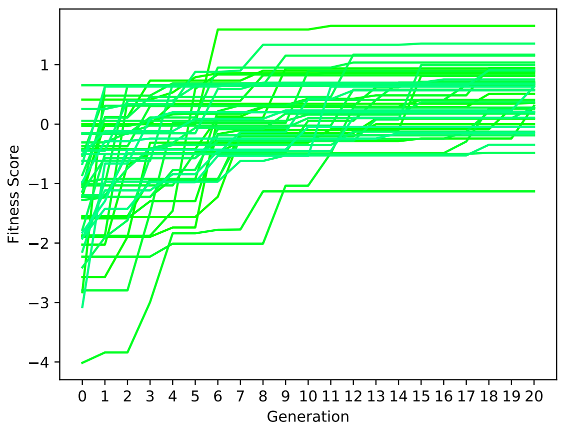
Figure A.1: MLP Population Fitnesses per Generation



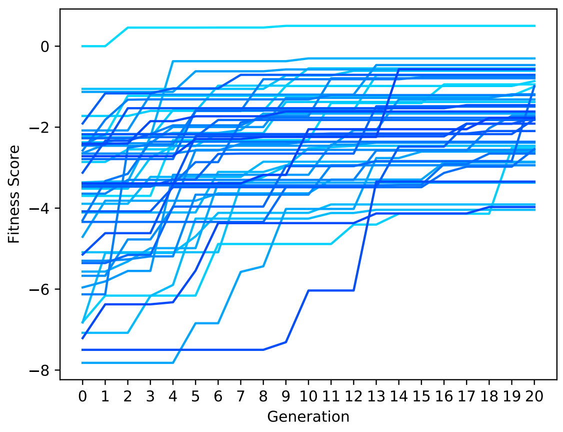
(a) 10%



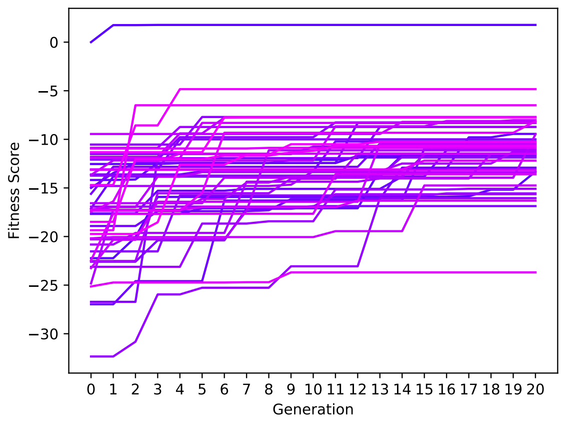
(b) 30%



(c) 50%

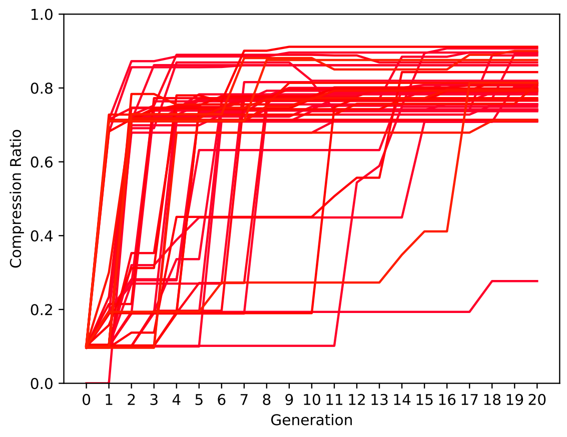


(d) 70%

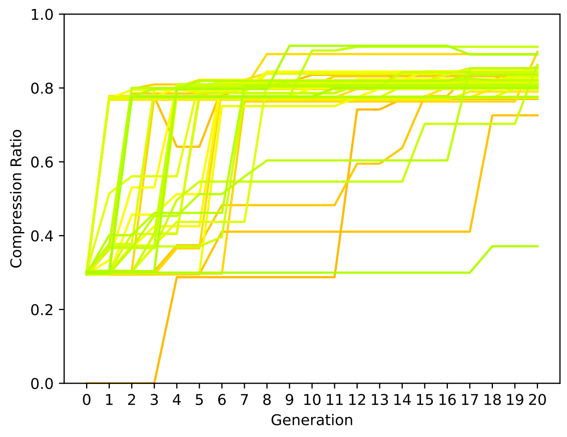


(e) 90%

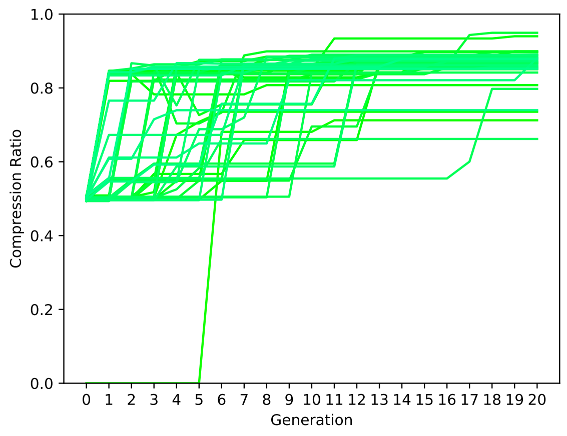
Figure A.2: MLP Population Compression per Generation



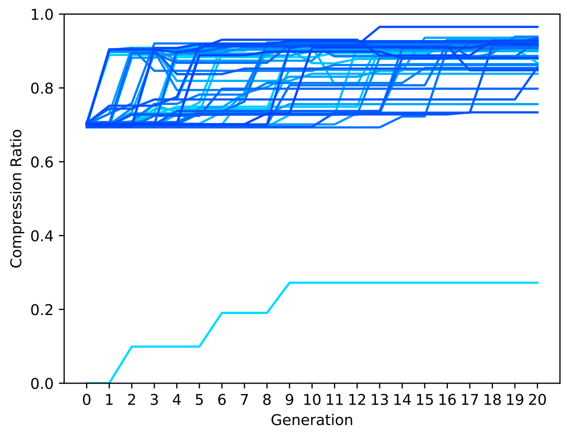
(a) 10%



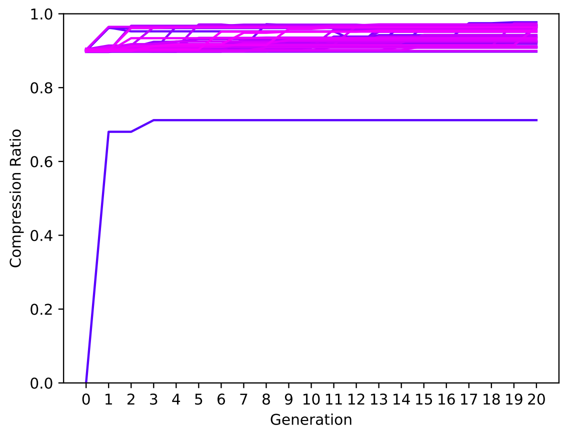
(b) 30%



(c) 50%

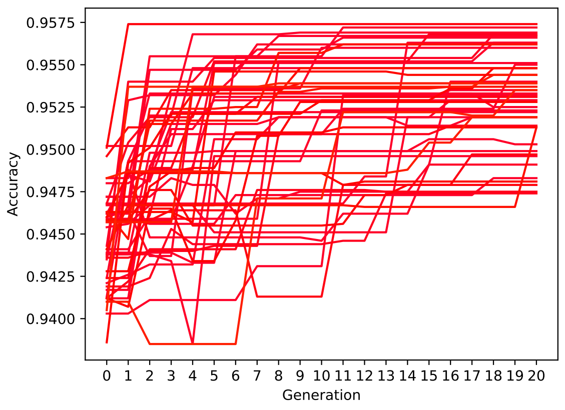


(d) 70%

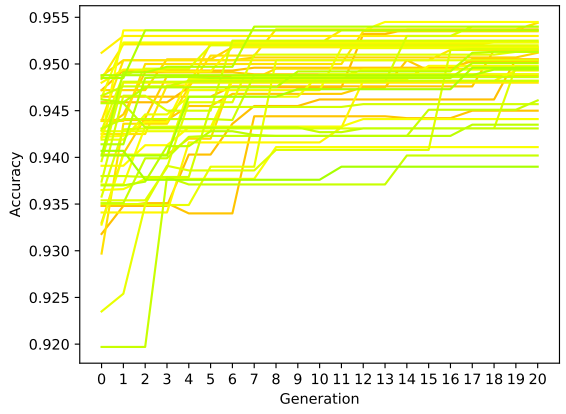


(e) 90%

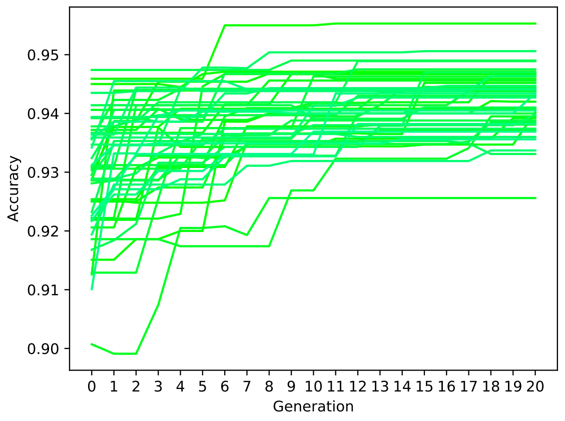
Figure A.3: MLP Population Accuracy per Generation



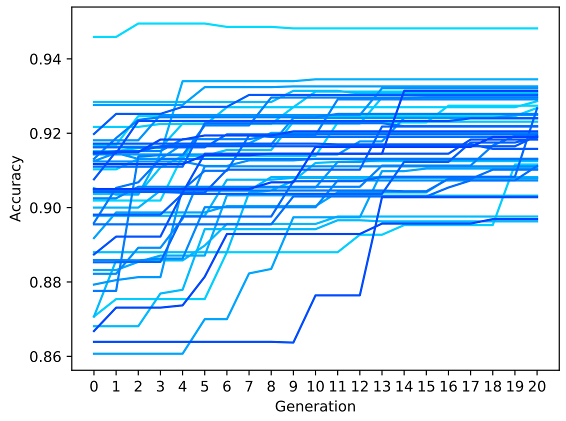
(a) 10%



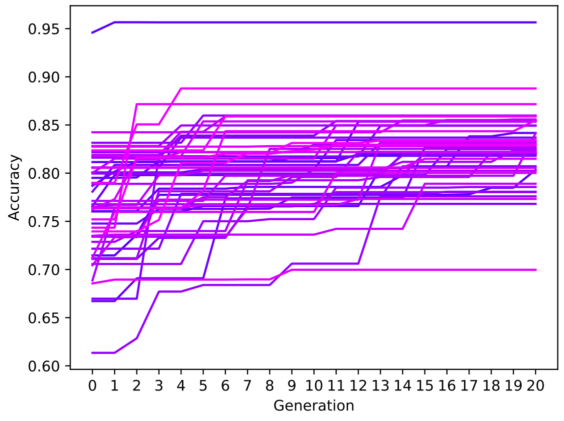
(b) 30%



(c) 50%

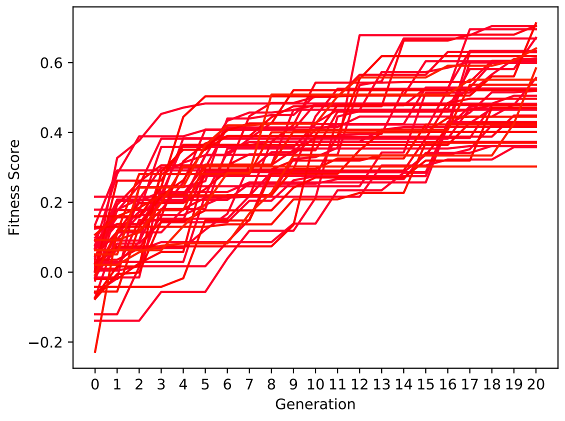


(d) 70%

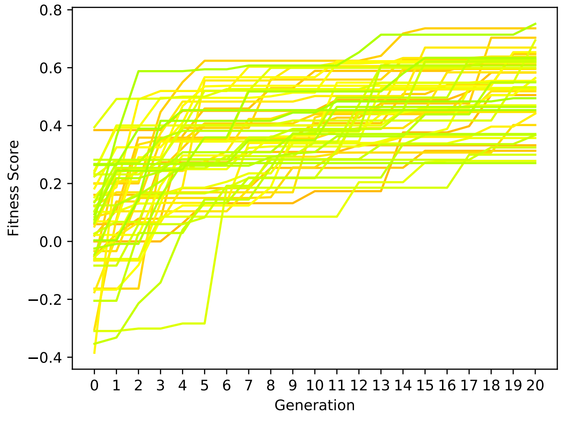


(e) 90%

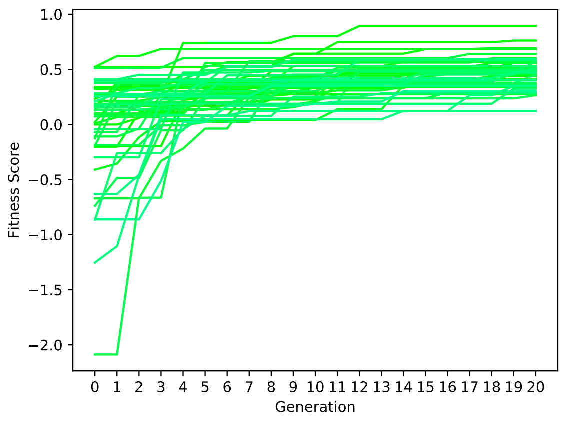
Figure A.4: CNN Population Fitnesses per Generation



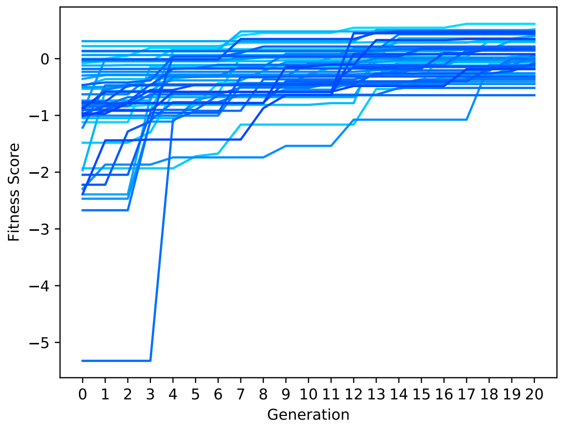
(a) 10%



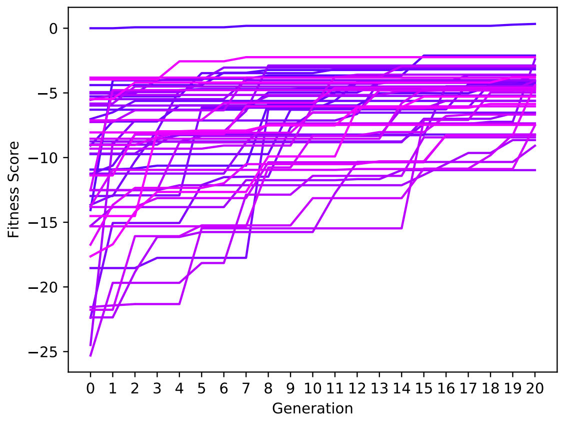
(b) 30%



(c) 50%

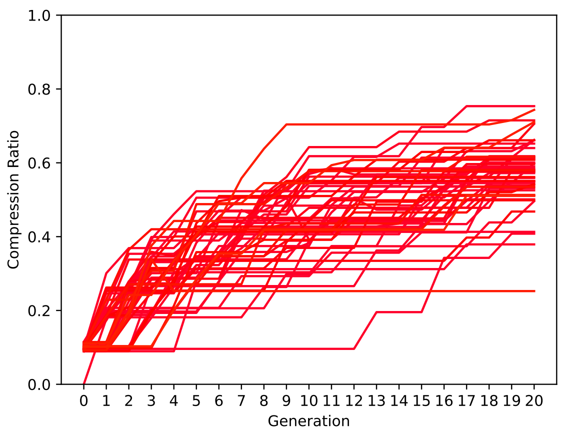


(d) 70%

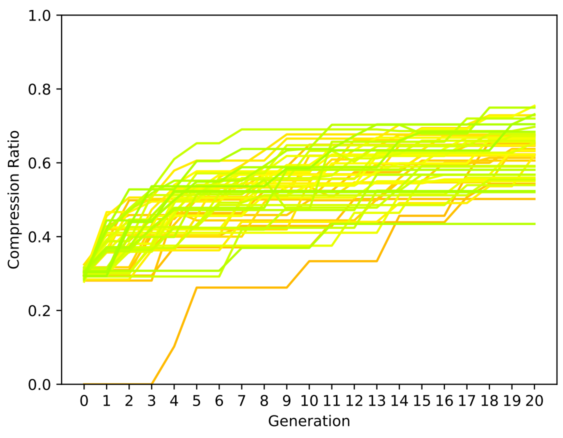


(e) 90%

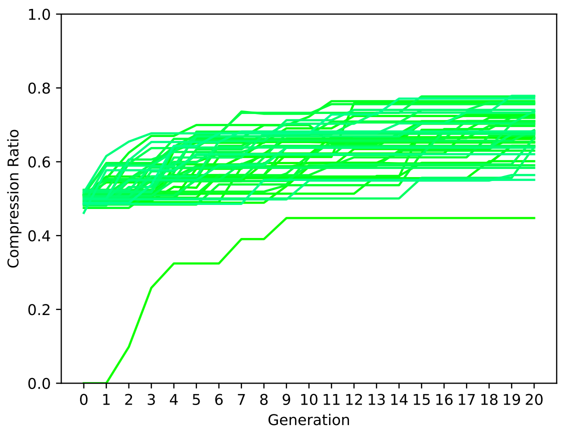
Figure A.5: CNN Population Compression per Generation



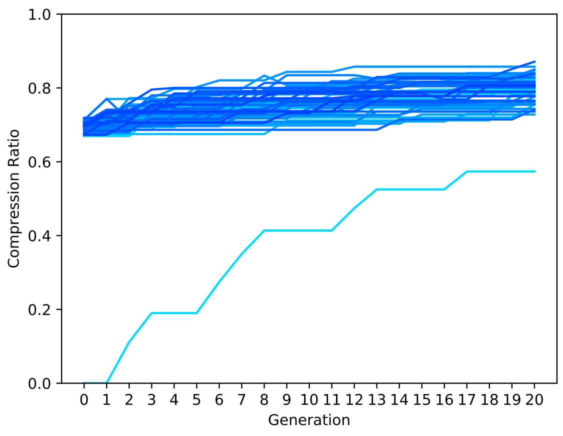
(a) 10%



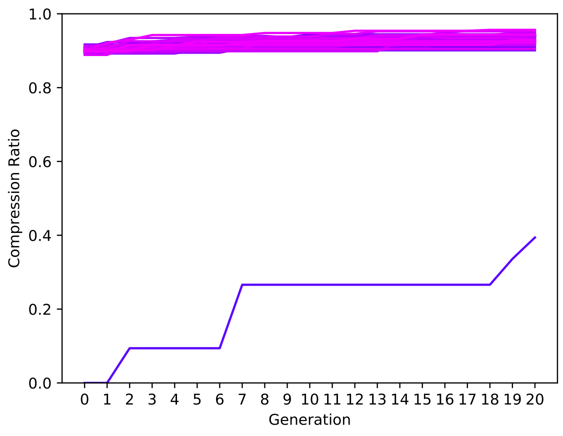
(b) 30%



(c) 50%

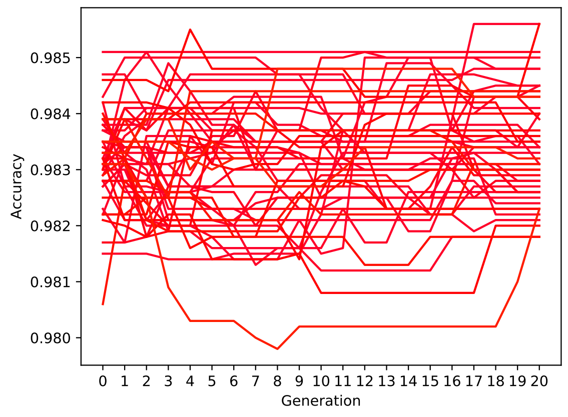


(d) 70%

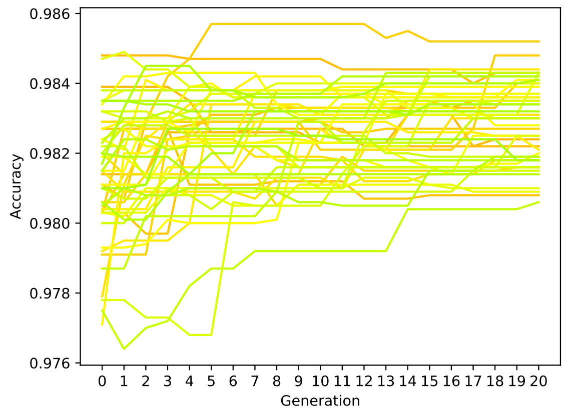


(e) 90%

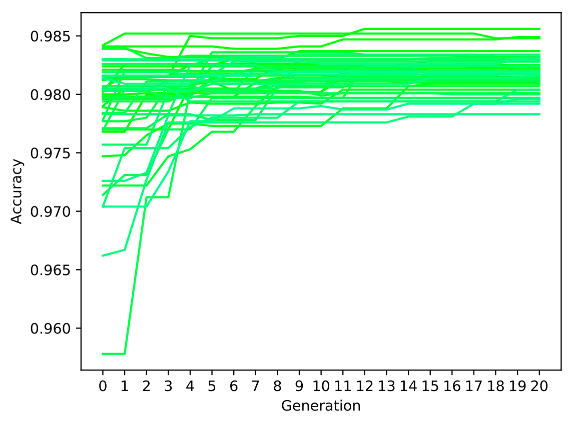
Figure A.6: CNN Population Accuracy per Generation



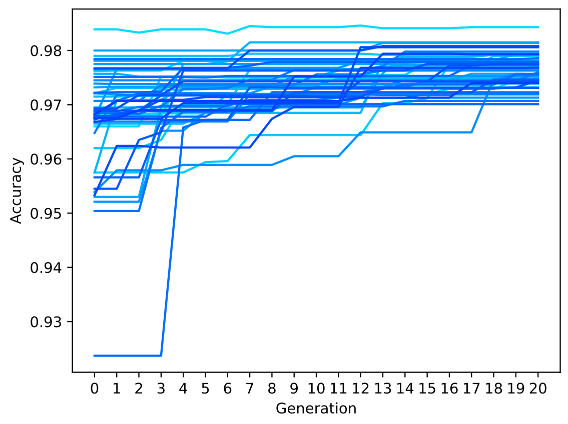
(a) 10%



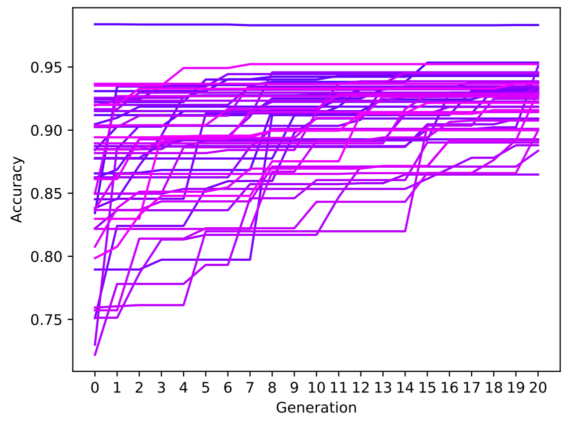
(b) 30%



(c) 50%



(d) 70%



(e) 90%

Figure A.7: VGG-16 Fitness

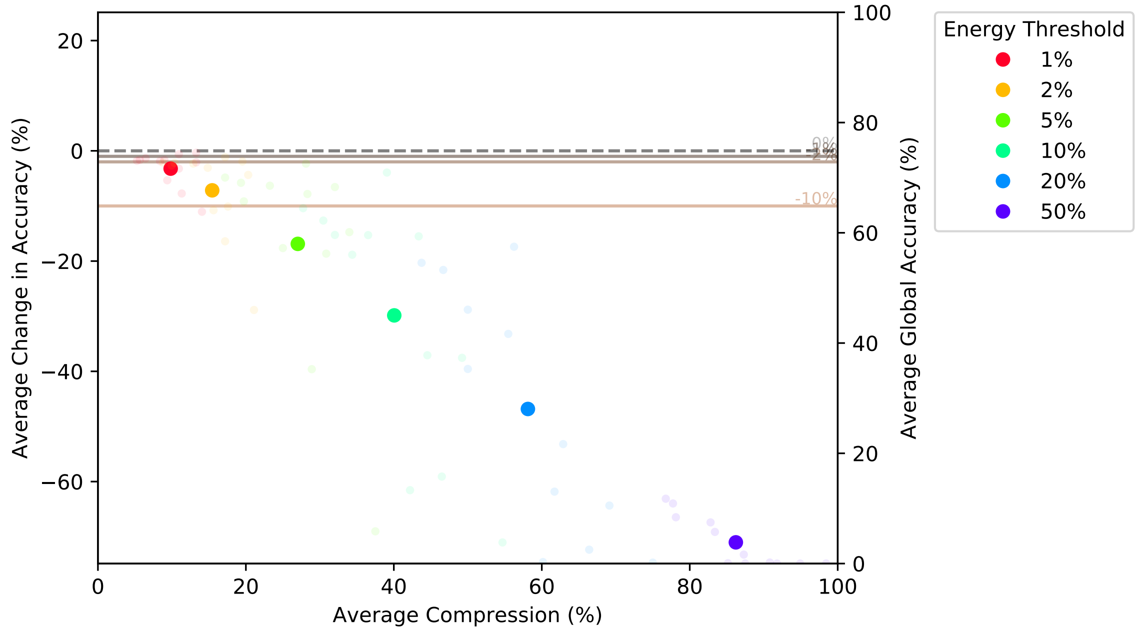
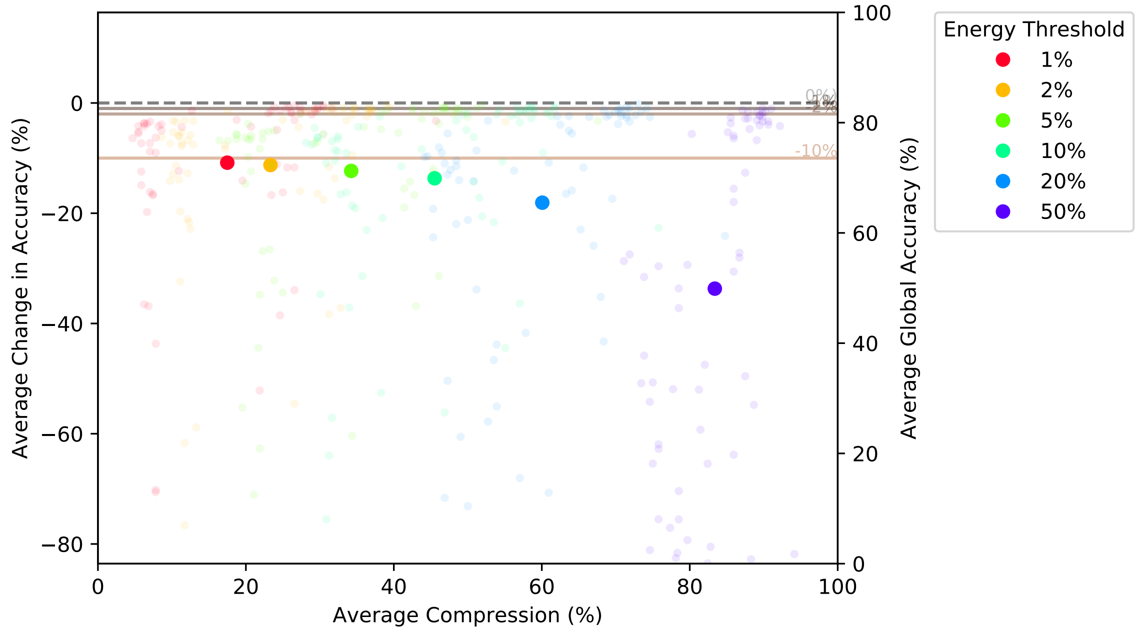


Figure A.8: ResNet-101 Fitness



Bibliography

D.-A. Clevert, T. Unterthiner, and S. Hochreiter. Fast and accurate deep network learning by exponential linear units (elus). *arXiv preprint arXiv:1511.07289*, 2015.

J. P. Cohen, H. Z. Lo, and W. Ding. Randomout: Using a convolutional gradient norm to win the filter lottery. *CoRR*, abs/1602.05931, 2016. URL `http://arxiv.org/abs/1602.05931`.

R. Collobert, K. Kavukcuoglu, and C. Farabet. Torch7: A matlab-like environment for machine learning. In *BigLearn, NIPS workshop*, number EPFL-CONF-192376, 2011.

J. P. M. de Sá, L. M. Silva, J. M. Santos, and L. A. Alexandre. *Minimum error entropy classification*. Springer, 2013.

X. Glorot, A. Bordes, and Y. Bengio. Deep sparse rectifier neural networks. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, pages 315–323, 2011.

R. H. Hahnloser, R. Sarpeshkar, M. A. Mahowald, R. J. Douglas, and H. S. Seung.

Digital selection and analogue amplification coexist in a cortex-inspired silicon circuit. *Nature*, 405(6789):947, 2000.

S. Han, H. Mao, and W. J. Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*, 2015a.

S. Han, J. Pool, J. Tran, and W. Dally. Learning both weights and connections for efficient neural network. In *Advances in neural information processing systems*, pages 1135–1143, 2015b.

B. Hassibi and D. G. Stork. Second order derivatives for network pruning: Optimal brain surgeon. In *Advances in neural information processing systems*, pages 164–171, 1993.

B. Hassibi, D. G. Stork, G. Wolff, and W. Takahiro. Optimal brain surgeon: Extensions and performance comparison. In *Advances in neural information processing systems*, pages 263–270, 1994.

K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

A. W. H. House. Discrete-time signals, May 2004.

A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

S. Janjic, P. Thulasiraman, and N. Bruce. Redundancy in convolutional neural networks: Insights on model compression and structure. In *2018 International Joint Conference on Neural Networks (IJCNN)*, pages 3601–3608, 2018.

A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

Y. LeCun and C. Cortes. The mnist database. URL `http://yann.lecun.com/exdb/mnist/`. Accessed: 2018-05-30.

Y. LeCun, J. S. Denker, and S. A. Solla. Optimal brain damage. In *Advances in neural information processing systems*, pages 598–605, 1990.

Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, November 1998a.

Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998b.

Y. LeCun, Y. Bengio, and G. Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015.

X. Lei, A. W. Senior, A. Gruenstein, and J. Sorensen. Accurate and compact large vocabulary speech recognition on mobile devices. In *Interspeech*, volume 1. Citeseer, 2013.

A. Mallya and S. Lazebnik. Packnet: Adding multiple tasks to a single network by iterative pruning. *CoRR*, abs/1711.05769, 2017. URL `http://arxiv.org/abs/1711.05769`.

A. Mallya and S. Lazebnik. Piggyback: Adding multiple tasks to a single, fixed network by learning to mask. *CoRR*, abs/1801.06519, 2018. URL `http://arxiv.org/abs/1801.06519`.

T. Miconi. Neural networks with differentiable structure. *CoRR*, abs/1606.06216, 2016. URL `http://arxiv.org/abs/1606.06216`.

M. Mitchell. *An introduction to genetic algorithms.* MIT press, 1998.

P. Molchanov, S. Tyree, T. Karras, T. Aila, and J. Kautz. Pruning convolutional neural networks for resource efficient transfer learning. *arXiv preprint arXiv:1611.06440*, 2016.

R. Reed. Pruning algorithms-a survey. *IEEE Trans. Neural Networks*, 4(5):740–747, 1993. doi: 10.1109/72.248452. URL `https://doi.org/10.1109/72.248452`.

O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015. doi: 10.1007/s11263-015-0816-y.

P. Y. Simard, D. Steinkrau, and I. Buck. Using gpus for machine learning algorithms. In *Eighth International Conference on Document Analysis and Recognition (IC-*

DAR'05)(ICDAR), volume 00, pages 1115–1119, 08 2005. doi: 10.1109/ICDAR.2005.251. URL doi.ieeecomputersociety.org/10.1109/ICDAR.2005.251.

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

N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

K. O. Stanley and R. Miikkulainen. Evolving neural networks through augmenting topologies. *Evolutionary Computation*, 10:99–127, 2002.

D. Whitley, T. Starkweather, and C. Bogart. Genetic algorithms and neural networks: Optimizing connections and connectivity. *Parallel computing*, 14(3):347–361, 1990.

D. Whitley et al. Genetic algorithms and neural networks. *Genetic algorithms in engineering and computer science*, 3:203–216, 1995.

C. S. Wiki. Convolutional neural networks (cnns) — computer science wiki,, 2017. URL https://computersciencewiki.org/index.php?title=Convolutional_neural_networks_(CNNs)&oldid=6229. [Online; accessed 14-June-2018].

C. S. Wiki. Multi-layer perceptron (mlp) — computer science wiki,, 2018. URL https://computersciencewiki.org/index.php?title=Multi-layer_perceptron_(MLP)&oldid=8030. [Online; accessed 14-June-2018].

J. Wu, C. Leng, Y. Wang, Q. Hu, and J. Cheng. Quantized convolutional neural networks for mobile devices. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4820–4828, 2016.

J. Yosinski, J. Clune, Y. Bengio, and H. Lipson. How transferable are features in deep neural networks? In *Advances in neural information processing systems*, pages 3320–3328, 2014.