INTERPRETING AND PRUNING COMPUTER VISON BASED NEURAL NETWORKS

School of Electronic Engineering and Computer Science

Queen Mary, University of London

PhD Thesis

Woody Bayliss

Supervisors: PROF. Ebroul Izquierdo, Dr Marta Mrak, Dr Qianni Zhang

2023ACKNOWLEDGEMENT OF SUPPORT

Engineering and Physical Sciences Research Council

This work was supported by the UKRI EPSRC, grant number 2246465.

STATEMENT OF ORIGINALITY

I, Woody Bayliss, confirm that the research included within this thesis is my own work or that where it has been carried out in collaboration with, or supported by others, that this is duly acknowledged below, and my contribution indicated.

Previously published material is also acknowledged below.

I attest that I have exercised reasonable care to ensure that the work is original and does not to the best of my knowledge break any UK law, infringe any third party's copyright or other Intellectual Property Right, or contain any confidential material.

I accept that the College has the right to use plagiarism detection software to check the electronic version of the thesis.

I confirm that this thesis has not been previously submitted for the award of a degree by this or any other university.

The copyright of this thesis rests with the author and no quotation from it or information derived from it may be published without the prior written consent of the author.

Signature:

Date: 21/08/2023

Details of collaboration and publications:

1. Bayliss, W., Murn, L., Izquierdo, E., Zhang, Q. and Mrak, M., 2022, May. Complexity Reduction of Learned In-Loop Filtering in Video Coding. In 2022 IEEE International Symposium on Circuits and Systems (ISCAS) (pp. 506-510). IEEE.

ABSTRACT

Computer vision is a complex subject matter entailing tasks, such as, object detection and recognition, image segmentation, super resolution, image restoration, generated artwork, and many others. The application of these tasks is becoming more fundamental to our everyday lives. Hence, beyond the complexity of said systems, their accuracy has become critical. In this context, the ability to decentralise the computation of the neural networks behind cutting edge computer vision systems has become essential. However, this is not always possible, models are getting larger, and this makes them harder, or potentially impossible to use on consumer hardware.

This thesis develops a pruning methodology called “Weight Action Pruning” to reduce the complexity of computer vision neural networks, this method combines sparsity pruning and structured pruning. Sparsity pruning highlights the importance of specific neurons and weights, and structural pruning is then used to remove any redundancies. This process is repeated multiple times and results in a significant decrease in the computing power required to deploy a neural network, reducing inference times and memory requirements.

Weight Action Pruning is first applied to deblocking neural networks used in video coding. Pruning these networks with Weight Action Pruning allowed for large computational reductions without significant impacts on accuracy.

To further test the validity of Weight Action Pruning on multiple datasets and different network architectures, Weight Action Pruning was tested on the generative adversarial U-Net used in a seminal paper in the field. This work showed that the ability to prune a neural network relies not only on the neural network’s architecture, but also the dataset used to train the model.

Weight Action Pruning was then applied to image recognition networks VGG-16 and ResNet-50, this allowed Weight Action Pruning to be directly evaluated against other state of the art pruning methods. It was found that, models that were pruned to a set size had higher accuracies than models that were trained from scratch with the same size.

Finally, the impact of pruning a neural network is investigated by analysing weight distribution, saliency maps and other visualizations.

It must be noted that Weight Action Pruning comes at a cost at training time, due to the re-training required. Additionally pruning may cause networks to become less robust, as they are optimised by removing the learnt “edge cases”.

ACKNOWLEDGEMENTS

I would like to thank my 3 supervisors, Ebroul Izquierdo, Marta Mrak, Qianni Zhang, for providing me with invaluable information and guidance with regards to my PhD and publications.

I would like to extend a further thank you to Luka Murn, from the BBC, for continuing to support my studies, and being a bridge to industry that all PhD students should have the pleasure to enjoy.

I would like to thank my family for supporting me throughout my studies and my life in general, hopefully, with the completion of this work, that infinite favour can begin to be repaid.

I would like to thank the open-source community, without this community this work would not have been possible.

Finally, I would like to thank James Bruce, if he had not mentioned to me late in the pub one night that he knew of a PhD opportunity at QMUL, I would have never even considered the possibility of starting a PhD, let alone finishing one.

LICENSE

TABLE OF CONTENTS

LIST OF KEY FIGURES

LIST OF KEY TABLES

List of Equations

LIST OF ABBREVIATIONS

ADAM – Adaptive Moment Estimation
AI – Artificial Intelligence
BBC – British Broadcasting Company
BD – Bjøntegaard-Delta
CIFAR – Canadian Institute For Advanced Research
CNN – Convolutional Neural Network
CTU – Coding Tree Unit
CU – Coding Unit
CV – Computer Vision
FCN – Fully Connected Network
FLOPs – Floating-Point Operations Per Second
FPS – Frames Per Second
GAN – Generative Adversarial Network
H.265 – High Efficiency Video Coding
IOU – Intersection Over Union
MAE – Mean Absolute Error
ML – Machine Learning
MLI – Machine Learning Imaging
MNIST – Modified National Institute of Standards and Technology
MRI – Magnetic Resonance Imaging
MSE – Mean Squared Error
NN – Neural Network
NR – Noise Reduction
PSNR – Peak Signal to Noise Ratio
QP – Quantisation Parameters
RD – Rate Distortion
ReLU – Rectified Linear Unit
ResNet – Residual Network
SGD – Stochastic Gradient Descent
SOTA – State of The Art
SR – Super Resolution
SSIM – Structural Similarity Index Measure
U – Blue Projection
UCLF – Uni-Component Loop Filter
V – Red Projection
VMAF – Video Multimethod Assessment Fusion
VGG – Visual Geometry Group
VTM – VVC Test Model
VVC – Versatile Video Coding
WAP – Weight Action Pruning
Y – Luminance

1 INTRODUCTION

The progress of machine learning over the last 10 years has seen many improvements over a wide range of applications. We have seen algorithms that can drive vehicles [1], master the oldest and most computationally complex board game GO [2]. And now learning algorithms can even beat professional human players at modern computer games [3].

The surge in machine learning applications has also seen models become too large and unwieldy for many researchers to interact with. This results in having to use smaller less accurate versions of the models or having to pay for expensive equipment. Neural network pruning allows for larger models to be condensed into smaller forms allowing them to be used more universally.

There are methods that can interpret specific learning algorithms, helping to explain these systems to an extent. These include counterfactuals [4], Shapley values [5] and model simplification and summation methods [6]. However, these methods are not applicable across the whole suite of learning algorithms, and most are for very specific use cases. Additionally, some of these methods must be included in a machine learning model before training occurs. This is a compounding issue, meaning that models developed without interpretability in mind are exempt from these interpretation methods.

Although similar, interpreting and explaining are different things in machine learning. “Interpretability is about the extent to which a cause and effect can be observed within a system”, whereas explaining is “The extent to which the internal mechanics of a machine learning system can be explained in human terms” [7]. This means that an interpretable machine learning algorithm is not necessarily explainable.

Interpretability is an important aspect when it comes to pruning neural networks, but it is extremely difficult to build a universal system that can provide interpretability to all flavours of machine learning models. Instead, this work focuses on abstraction of this problem to subvert the issues associated with interpretability. This is done by inspecting weight values and neuron activity and deriving importance from these values.

Pruning allows for a smaller model with similar accuracy to be derived from a larger model, reducing its size and complexity. This work lays out a 3-step process to achieve such a task:

- Network sparsity reduction.
- Insignificant Neuron Identification.
- Network retraining.

These processes are repeated in a loop until the network performance degrades to an unrecoverable state, or the neural network is reduced to a desired size.

1.1 MOTIVATION

1.1.1 Neural Network Pruning

Machine learning has allowed for vast improvements in many fields [1,2,3], this thesis will focus on computer vision machine learning systems. Recently large text to image models have been developed by companies like stability AI, these models are so large that most consumer grade graphics cards (GPUs) cannot run these models [164]. This poses an issue as the only solution is to pay a cloud supplier, like google, to host a GPU for you. Or to pay for a service that removes all access to edit the model and just provides you with a prediction of the model [165].

This is not an issue limited to text to image models, models from many niches are often overparameterized [167]. This overparameterization is what results in models with high memory usage and slow run times.

Pruning allows for models to be reduced in complexity (in terms of size) whilst preserving performance. It must be noted that everything comes at a cost and when it comes to pruning this cost is at training time, as models will take longer to train and prune. The models also may not be able to deal with edge cases that they previously could, this is due to the pruning process optimising the knowledge contained in the model [168].

1.1.2 Interpretability

In society we do not accept that someone just knows something. When someone is especially talented at a certain task with seemingly no explanation, they are called a savant or a genius. The rest of humankind must study books and papers, and practice… and then have it all validated by an exam. By today's standards most machine learning algorithms would be considered savants.

To understand the need for interpretable, explainable, machine learning algorithms it is important to understand that deeper insights into ML algorithms have far reaching applications. For example, in the application of driverless vehicles, when a crash occurs, it is important to know how the algorithm will react. Why did it react in that way? Should it have done something different [8]?

Another issue in machine learning is ascertaining whether an algorithm is biased, and understanding what makes it biased. By having more insight into a specific machine learning model, an engineer can tune and optimise said model for the application more effectively. Interpretability casts a wide net in terms of the potential ways it can help researchers and industry. More research into interpretability methods might change the way we fix and identify biased algorithms. It may also affect the underlying architecture and optimisation methods we use in machine learning algorithms.

1.1.3 Video Compression as a Test Case

Before a large neural network can be interpreted, a suitable network must be found. Many of the larger neural networks like VGG and AlexNET are already well understood, for this reason a different sufficiently large, but poorly understood network needed to be found. Video compression algorithms provide the perfect platform for this, machine learning methods have been applied to aid video compression however the method these techniques use is self-learnt. This means that without interpretation the learnt compression method remains a mystery.

Video compression isn't new. There has been an evolution of algorithms starting in 1991 with MPEG-1 [10] to the latest published standard H265 finished in 2019 [11]. Each algorithm adds new techniques to reduce the amount of disk space and bandwidth used by video whilst keeping the visual quality the same.

With the recent boom in the applications of artificial intelligence, researchers turned to compression algorithms to see if it would be possible to use machine learning to aid in this process. This has produced a similar evolution of machine learning models used in combination with video compression algorithms, producing compression software that contains elements of machine learning [12].

Algorithms that are intrinsic to the compression process can be used to increase the performance of a video codec [13]. It has been shown that a neural network can be used to replace 3 hardcoded filters in the standard Versatile Video Codec (VVC), namely the Deblocking Filter, Sample Adaptive Offset filter, and the Adaptive Loop Filter [13]. But this work is not without issues.

The key issues with the implementation of the current state of the art as follows.

- Currently the method runs alongside the normal filters in VVC, meaning that the analysis of the images for video coding are performed twice.
- Additionally, the neural filtered image is not used in its entirety. Instead, a Rate Distortion calculation is performed between the two methods and the best patch for a specific region is picked.
- Since the neural network is processing images at the same time as the other filters. When tests are performed on the test set, the proposed method takes 30% longer to process when aided by a GPU, and over 1000% longer when processed on a GPU.

The problem here is twofold, if the network is aimed at replacing the function of these filters, why are they still in the codec? It seems reasonable to assume that their function should at least be disabled when performing compression with the neural network. Secondly, the proposed network is very complex. This means that with state-of-the-art equipment, inference can be performed within arguably acceptable time increases. However, if a graphical processing unit is not present then this method is completely unfeasible.

This sort of neural network fits our needs perfectly.

- Its functionality is poorly understood.
- It’s too large and complex to deploy at consumer level.
- A decrease in complexity would aid industry and research in this field.

Therefore, this work will initially focus on reducing the complexity of this network with the aim of reducing the time component and lowering the barrier of needing a graphical processing unit.

1.1.4 Further Experiments

As this research progresses so will the complexity and universality of the pruning algorithm developed. After establishing a performant pruning algorithm for deblocking networks this research will focus on testing different implementations of the pruning algorithm. Many models, datasets and pruning types will be tested to confirm the efficacy and adaptability of the algorithm.

This work culminates in controlled experimentation against state-of-the-art pruning methods. This is done to create concrete conclusions about the usefulness of the pruning algorithm.

1.2 Research Question

Can sparsity pruning be used in combination with neuron activity as a guide for structurally pruning neural networks? Can this pruning improve the computational efficiency of these networks?

1.3 Thesis Structure

Chapter 2 – Literature Review

This first chapter covers key topics in detail, focusing on neural networks, interpretability, pruning and video and image compression methods. The reader will gain a foundational understanding in the key techniques used throughout this research.

Chapter 3 – Development of Pruning Algorithm and Interpretation Tools

This chapter covers the initial development of an interpretation tool called Machine Learning Imaging and the development of the basic pruning algorithm. Additionally, the intuition behind the pruning is explained in detail, and justified through observation. This work provides the reader with a fundamental understanding of the pruning and interpretation methods used to prune neural networks in this thesis.

Chapter 4 – JPEG Restoration using Convolutional Neural Networks

This chapter uses machine learning methods originally developed to aid video compression methods and applies them to aid with JPEG compression instead. This was done to test the universality of the method. Additionally, these experiments serve to further confirm the observations found in chapter 3. The technique shows that training a network in a universal fashion not only reduces the number of parameters required, but also creates a more coherent overall network. Additionally the reader is provided with an introduction to image processing in machine learning systems and proves the universal efficacy of using MLI when interpreting neural networks.

Chapter 5 – Pruning a Video Deblocking Neural Network

This chapter expands on the work in chapter 4 and prunes a neural network used to reduce blocking artefacts in video compression. The network was reduced in complexity, inference time and memory consumption. This work was published at the ISCAS 2022 conference. This exposes the reader to the first fully realised version of the pruning algorithm and demonstrates its benefits.

Chapter 6 – Pruning an Image Translation Generative Adversarial Neural Network

This chapter applies the pruning algorithm used in chapter 5 on a Generative Adversarial Network and on 3 datasets. These experiments provide an insight into dataset complexity and the role it plays when pruning neural networks. The universality of the pruning algorithm and capabilities and issues when pruning Generative Adversarial networks are also investigated.

Chapter 7 – Pruning Classification Networks

This chapter applies the pruning algorithm to classification networks, these networks were selected by replicating the test methodology used by state of the art pruning research. These methods are compared and depending on the scenario our pruning algorithm performs the same as if not better than other state of the art methods. This is true for all tests except when using the Res-Net50 model. Theories are investigated as to why the method failed for this network. This chapter gives clarity to the reader with regards to how the developed pruning algorithm compares to others that currently exist in the literature.

Chapter 8 – Impacts of Pruning Neural Networks

This chapter asks the question “what does pruning a neural network do to the internal representations inside the network?”. Experiments explore this concept by examining saliency maps, XRAI maps, maximisation maps and weight distributions of the network. Additionally, suggestions are also made as to what is being removed from the network in terms of “knowledge”, when pruning is applied.

Chapter 9 - Summary of Contributions and Future Work

This final chapter provides the reader with a summarised version of the main conclusions from each chapter and suggests some future work, building on the most viable concepts developed throughout the thesis.

2 LITERATURE REVIEW

This research focuses on three overarching tasks, firstly, a suitably abstract problem must be identified to solve with an existing machine learning model. Then this model will be analysed using established methods using ML interpretability, and this will hopefully identify areas where the model can be improved. For the purposes of this research this problem will initially focus on using ML to aid in video compression (specifically image filtering and de-blocking).

Secondly, this research will try to identify new novel methods to perform interpretability and analysis on basic, well established machine learning models. Once new tools have been created through this method, they will then be applied to the video compression task and other tasks to see if the method is truly model and application agnostic.

Because of the two aspects above, and because an in-loop filter replacement neural network was selected as a starting point, it will be important to have a deep understanding of video encoding, machine learning and interpretability methods. The literature review will cover these aspects in detail.

Additionally, a knowledge of neural network pruning will be required, this will be used to leverage the understanding gained through interpreting networks. The final step of this thesis will focus on testing the developed pruning algorithm on an image translation task, and image classification. Efficacy of the pruning algorithm will be determined through comparison with SOTA classification pruning algorithms. Individual chapters will have expanded literature reviews where needed.

2.1 NEURAL NETWORK PRUNING

Network pruning is a process whereby unimportant parts of a neural network are identified and removed from a neural network. The aim is to reduce the level of complexity required to compute the output and generally lower the requirements when a model is deployed.

2.1.1 Taxonomy of Pruning Methods

Pruning neural networks is an incredibly large field, this field contains many techniques that can be applied in a modular fashion meaning that grouping pruning methods can be difficult. However, pruning will always fit into one of the following 3 categories Structured, Unstructured, Semi-Structured.

2.1.2 Structured Pruning

Structured Pruning is the process of removing sets of channels, filters, or neurons in each layer of a neural network. These components are removed in their entirety and in fact structured pruning can remove layers entirely [81,82].

This kind of pruning does not require any kind of special hardware or software optimisations to achieve computational complexity reduction, and results in neural networks that are faster and more memory efficient. However generally structural pruning is difficult to implement without affecting the accuracy of neural networks.

This kind of pruning will be the target for this thesis as it achieves the goals set out in the research question.

2.1.3 Unstructured/Sparsity Pruning

Unstructured or sparsity pruning focuses on a model's weights, this pruning will select from a set of weights and set a certain percentage of these weights to 0 [22,85,86]. This optimisation can be built directly into the learning function and so pruning can happen in tandem with training.

This kind of pruning can provide speed up to machine learning models [112], however this requires specialised hardware and or software to achieve [49]. Sparsity pruning does provide a decrease in the size of model files when saved to disk. This is achieved by exploiting the amount of 0's that have been stored in the weights file with run length encoding.

Overall, this process does not inherently tackle the issues laid out in the research question and therefore will not be used as a final solution. It must be noted however that sparsity pruning has much less impact on the accuracy of neural networks and so this method still might prove to be useful.

2.1.4 Semi-Structured Pruning

Semi structured pruning (sometimes called pattern-based pruning) is an advanced version of sparsity pruning. This technique uses pre-defined "patterns" in its convolutional layers to optimise throughput of a machine learning model [83,84].

A simple way to consider this guided version of sparsity pruning whereby the metric that guides the pruning process will also take these patterns into account when zeroing out weight values.

Again, like sparsity pruning this method needs additional software to achieve any runtime increases, however it is still of interest.

2.1.5 Neuron/Weight selection

Deciding how to select a neuron, or a weight connection for removal is key when pruning neural networks. This selection broadly falls into one of the following 3 categories, Magnitude, Sensitivity, Loss Change.

Magnitude based pruning simply selects the lowest magnitude weights and removes them from the model. Sensitivity has different definitions depending on the specific works, but generally the importance of a neuron or channel is derived from the magnitudes of the weight connections to that neuron. Finally, loss change is a process of trial and error whereby a neuron or weight is removed, and evaluation is performed. If the loss goes down or stays the same due to this change, then the change is made permanent. If the loss goes up, then the change is reverted.

2.1.6 Leveraging 0 weight values for pruning

Whilst the pursuit of neural network speedup may seem futile when considering sparsity pruning there are some situations where one of these techniques may help the other. It is possible that 0 weight values could be leveraged to reduce inference time and memory use, below is the general form for neuron activation in a neural network. Assuming all weight values for one neuron are 0.

$$f\left(b + \left(\sum_{i=1}^{n} x_i w_i\right)\right) = Act$$

Equation 1: Activation of a neuron given the activation function f the bias value b and the weights w and the activation of previous neurons x.

If the weight values here are 0 then Equation 2 can be simplified to Equation 3.

$$f(b) = Act$$

Equation 2: Simplified representation of a neuron where all connected weights are 0.

Because bias is normally small and insignificant to the overall activation calculation the bias can be completely ignored, resulting in Equation 3 simplifying to Equation 4.

$$Act = 0$$

Equation 3: A further simplified representation of a neuron where all connected weights are 0 and bias values are assumed to be 0.

Removing these weights makes sense, but it would require an adaption to the current sparsity pruning methods to identify and remove these neurons (this would combine both sparsity and structural pruning). It may be perhaps that once pruned it is not guaranteed that all weight values for a given neuron will become 0, in this case a different selection criterion must be decided upon.

2.2 Interpretability

2.2.1 Current Methods

There are already many ways of interpreting ML models but very few are model agnostic, and even if they are model agnostic, they are not application agnostic, and may for example be limited to interpreting only specific tasks.

Additionally, there is also the issue of displaying the explanation of a model, for image recognition tasks this might take the form of highlighting the “interest” the algorithm has had in certain pixels [14]. But explaining and interpreting why an ML model can play a modern computer game is not as simple [3]. This is because the model is assumed to also have some inherent knowledge of the game, this knowledge is what we wish to interpret in that specific case.

2.2.2 Decision Tree Summation and Simplification

Decision trees are large structures that lead to an answer, called a leaf, with multiple decision points, these make up a subtree. When these trees are small the answers can be easy to track back and interpret, but often trees are combined and as this happens the complexity becomes too hard for a human to understand [16].

The basic way of simplifying and interpreting decision trees is to individually evaluate each decision point and give a summary of how each affected the result of the tree. But there are multiple ways to achieve this goal, and all have different benefits.

One of these techniques is called cost complexity pruning [16], this is a two-stage process where many trees are generated from the original. These are the same as the original tree, however each subsequent tree generated has the subtrees of the original tree replaced with leaves, until the last one generated is just a leaf. Then each tree is evaluated, and a final one is picked as the “pruned” tree [15]. This gives a much more simplified tree that can be easily interpreted by a human.

Another approach is reduced error pruning, this process uses the original tree and replaces each non-leaf subtree of the main tree with the best possible leaf; achieved by looking at the change in miscalculation of this new tree. If the new tree gives equal or fewer errors then this subtree is replaced with the leaf, this continues until the error only increases. This once again reduces the tree and makes interpretation easier.

Similarly, pessimistic pruning is where the initial error rate of the tree is determined from a training set. A subtree from the original tree is then replaced when a misclassified case is within one standard deviation error of the tree's error rate. This method is faster, and it does not require a test set of trees to be generated, it also simplifies the tree for interpretation.

Finally, there is a method called simplifying to production rules. This does not prune the original tree but replaces it with production rules [16], this involves first creating the rules and then grouping them together for evaluation. Rules are generated by finding only leaves that have a positive influence towards the solution, any solution with a negative leaf is not a production rule. This has the benefit of knowing the reasoning behind "true" cases for a given tree but erases all explanations for "False" cases.

These are all good ways of evaluating decision trees, and can to an extent, be abstracted to neural networks, however, this can only apply to linear neural networks and many neural networks are not linear.

2.2.3 Distillation

This is a method of reducing a large ML model into a smaller model that is almost as accurate as the original model. The idea is that the original model will be too massive to interpret, but the smaller model that replaces the larger one may be easier to understand. Distillation is not a directly interpretable method for machine learning but can be applied to aid other techniques and remove any redundancy of large models [17]. Some would argue however that it is an interpretation method for machine learning because the smaller network extracts the interpretable features from the original model and therefore the new model is intrinsically interpretable.

2.2.4 LIME

Lime focuses on interpreting a model by approximating the model depending on the predictions of the model. Lime produces a local linear explanation model that produces simplified inputs considered "interpretable". These simplified inputs are mapped onto the input space in different ways depending on the input space. This method has been proved to be very useful in revealing exactly what a classifier has learnt. For example, LIME has been used prove that a dog/husky classifier is just detecting snow instead of huskies, as the model only contained training data with huskies in snow [18].

2.2.5 Shapley

Shapley values were used very successfully for ML explanations in 2017 [19]. This method focuses on 3 main things.

- Explaining a model's prediction as a model itself.
- Calculating SHAP values which are a measure of feature importance.
- New SHAP values are calculated from these initial values that are better aligned to human intuition.

This explainer model can in some ways be compared to the simple model created in distillation, it also focuses on feature extraction. These SHAP (Shapley Additive explanation) values can be used as a measure of feature importance of the original model. The additive and negative values of these values are calculated for a given input and can be presented as such shown in Figure 1.

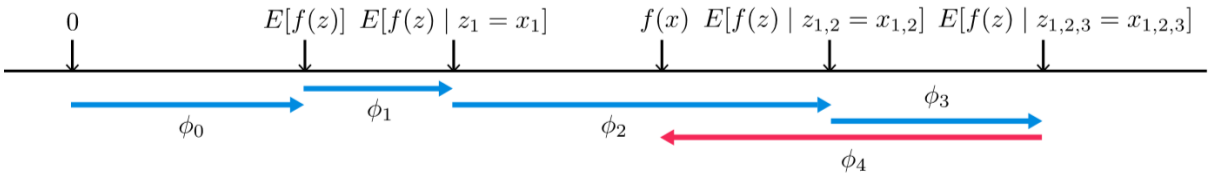


Figure 1-Visulisation of SHAP values showing how the importance of a certain feature can be visualised. [19]

2.2.6 Symbolic Metamodels

This work combines two previous methods of distillation and model summation [21], the idea is to create a "white box" model out of a "black box" by creating an approximate version of the black box contained within a Meijer G-function. This method is completely model agnostic. The approach is to use a trained neural network alongside an untrained network, then by varying the inputs to the trained network a smaller "meta model" is trained that contains the capabilities of the larger model. This is distillation, the larger model is reduced in these meta models that can perform the tasks of the larger model at similar accuracy. Because the structure and complexity of the meta models can be controlled, they are kept small and linear, this allows for a simple conversion of the meta models to a Meijer G-function. This aids simplification, the initial Meijer G-function that is created is very complicated with many terms, these can be exponential depending on the size of your network. However, because this function is now linear and in equation form, the process of removing complex hard to interpret parts becomes as simple as removing insignificant terms. Now this ML system becomes a formula that needs very little processing power and removes any doubt of the reason behind an output.

Whilst this symbolic metamodeling is very powerful, there is the issue that it is not built into the neural network structure natively, meaning after a network is trained only then can this process take place. Additionally, so far this work has only been demonstrated on a clinical basis, diagnosing a patient, and it does well but there is a maximum of 100 datapoints on an individual. So, in some ways dealing with interpreting an image (which can have over 2 million pixels per frame for 1080p video), could be considered more difficult than interpreting the diagnosis of a patient.

2.3 Coding Methodologies

As mentioned, video codecs have seen a recent rise in use of machine learning models to aid in the encoding decoding process. These techniques range from frame interpolation to super-resolution, but the use case we are interested in is deblocking filters. Although focused on the machine learning aspect it is important to have a base knowledge of the encoder used. Versatile Video Coding (VVC) started development in April of 2018, the successor to H.265 [25]. Many of the techniques used in VVC are transferable to other codecs, and so perhaps too are the machine learning integrations they have made. It will be used as the codec for any tests in this thesis.

2.3.1 YUV files

For the purposes of this research the video file structure used for tests will be a .YUV file. These are uncompressed video files that use the specific colour space of YUV. Files are stored as Luminance Y, and chrominance U and V. This is different to the RGB colour space, instead of all the components containing aspects of colour, only two do. However it continued to be the norm because the human eye is more sensitive to the detail contained in the luminance (Y) than the colour components (U,V) [26]. This is mostly due to the fact the human eye has twenty times rods than cones, rods are cells in the eye responsible for black and white vision and cones are responsible for colour [27]. Because of

this the colour components can be exploited and have very different compression applied to them in comparison to the Y components of the video.

2.3.2 Block structure in video compression

When compressing a video many methods are used, for this first stage of research we are not concerned with the compression contained between subsequent frames, called inter prediction. I will only be focusing on the methods used in a single frame, called intra prediction. It is also important to note that the method used by the AI in this research does not differentiate in its approach to dealing with these coding methods. It deals with one frame at a time and does not take advantage of any inter-prediction.

When processing a frame of video that frame is initially broken up into smaller blocks called a coding tree unit (CTU), in the case of VVC the size of these can be set between a max of 128x128 or a min of 32x32. This size is set for the whole sequence of video and cannot change frame to frame. Once the initial size of the CTU has been set, the block will be further partitioned into smaller blocks using a quadtree (QT) split, the idea is to make blocks of similar content so a general coding mechanism can be applied to the block. Next these blocks are split further through a recursive multi-type tree (MTT) this is done through a process of binary or ternary splits, this process can be seen in Figure 2.

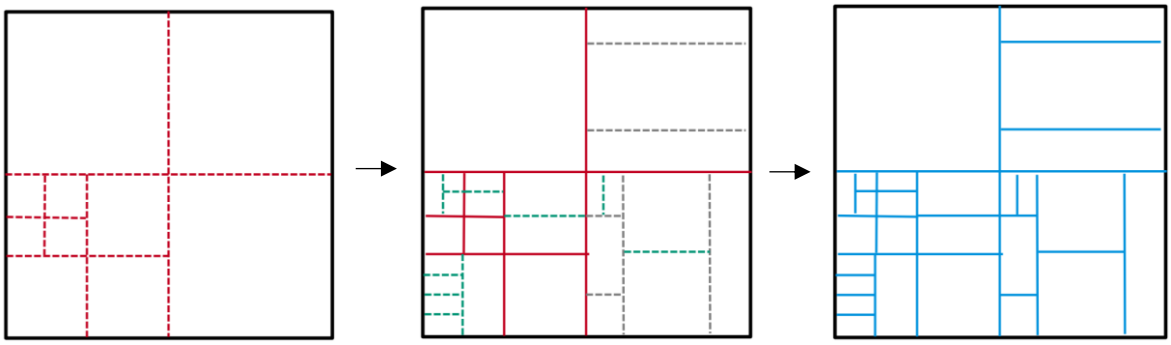


Figure 2: Example CTU at multiple stages of sectioning.

Eventually when the block structure has been decided through the process above, a lot of coding units (CU's) will exist over the whole image. These range from the size of the max CTU 128x128 and 4x4. The blocks should now contain similar parts of the image in terms of frequency profile, amongst other things. It is also important to mention that two of these are made per frame, one is used for the Y components and the other is used for the U and V components.

The reason why I have described these in detail is because they are used by the machine learning algorithm described later and give deeper information of the image being processed. The argument could be made that this process should be left to the algorithm, but this will be discussed later.

2.3.3 Filters

Finally, the last process before writing the file to disk is application of filters, these are aimed at addressing issues that are known to arise from the coding processes performed on the CTU's. The algorithm discussed later aims at replacing 3 filters, namely, Deblocking Filter (DBLF), Sample Adaptive Offset Filter (SAO) and the Adaptive Loop filter (ALF).

2.3.4 Deblocking Filter (DBLF)

This filter focuses on removing blocky artefacts between the different CU's, this can occur because very specific coding has been applied to each CU, when these are side by side the CU block structure can become visible shown in Figure 3.

Figure 3: Examples of blocking artefacts due to modern encoding techniques

2.3.5 SAO

This filter adds to the work done by the DBLF filter, it further smooths the pixel intensity, it does this depending on the edge shape of the pixels. Depending on this edge shape one of 32 bands is selected, this then indicates how drastically to offset the pixels in question to reduce the edginess of the pixels. These bands can be different for Y, U and V.

2.3.6 ALF

This is the last processing stage for each picture and is the tool that can be considered as fixing all the artefacts created as a side effect of the previous filtering stages. Like with many coding methods a cost is associated with each CU called the rate distortion (RD) cost. This can be considered for this explanation a comparison between the coded frame in its current form and the original frame. This calculation is used to compare each 4x4 block structure of the CU's, these are grouped into one of 25 filters that attempt to reduce the RD cost (bringing the coded frames pixel values back to the true frames values as much as possible). This can be seen as minimizing the Peak signal-to-Noise Ratio for any individual frame, this is the metric that VVC use on their tests and validation, so it makes sense that it is the last and most valued step. However, there are other metrics used by other companies that may prove more useful when calculating the quality of a frame of video [28][29].

2.3.7 Artificial Intelligence (AI) and video

When trying to leverage the benefits of AI for use in a video domain, it is important to bear in mind what AI is good at, prediction. And whilst it might not be initially obvious where prediction occurs in video coding, once the problem is broken down the benefit of AI in this space becomes obvious.

AI has been used for facial recognition for years now, with the advent of software like YOLO [30], general object detection has become relatively trivial [115]. And although developing these methods highlighted key facts, like the importance of convolutional components in machine learning systems, when dealing with images, these methods do not directly relate to how AI is leveraged in video coding [116].

The key difference for video coding, is that generally the task consists of taking large high-quality video files. Then using techniques that shortcut human vision and perception, a much smaller file that is still agreeable for humans to view can be created. This is done by removing components in the video that we do not notice. This is key to keep in mind, our video coding systems are hyper focused on human perception, this is exhibited in the framerate and the colour space [117,118]. Video codecs have been engineered to take advantage of the adequacy and inadequacy of our sight, for instance if a Pigeon were to watch a movie it would perceive a slideshow, and a dog sees far fewer colours than humans [119,120]. They would have both created very different video coding conventions to us.

2.3.8 Super resolution

Super resolution is the task of taking a low-resolution image and using a neural network to predict the components of the image that are not there. This task lends itself well to AI because datasets can be created easily that consist of artificially generated lower and higher resolution images [121].

2.3.9 Frame Per Second upscaling

Frame per second (FPS) upscaling is when an AI system is used to predict extra frames in a video sequence. Once again, the reason AI lends itself well to this task is that creating a dataset is simple, one could remove frames from an existing sequence. Slow motion footage is often used, this is recorded at much greater frame rates than normal video [122]. This is currently not used in video coding systems however it could very easily be implemented, in video coding there are already the idea of different types of frames, typically called I, B and P [32]. These stand for Intra frame, Bi-directional predicted frame, and Predicted frame. I frames contain the most data, predicted frames and Bi-directional frames, as the name suggests, are predicted from previous frames, and therefore contain less data. When streaming video or watching television you may see artefacts on the image that look like movement is still happening correctly, but the image of what is on screen is incorrect. This is because an I frame containing the grounding for the whole process has not been received properly, and the other frames have continued to predict with the information that was available. It makes sense with the success that has been had in this field, much like the prediction process used by I B and P frames, you could use AI to predict these frames. However, currently the research seems more focused at making super high framerate video. This produces slow motion sequences of video without the need for an expensive high framerate camera [33].

2.3.10 In Loop filtering for video compression

A network that performs well as an in-loop filter can take up to 72 times the amount of time the standard hand-coded algorithms take [13]. This makes these approaches completely unusable in a general use scenario. Machine learning methods leverage different information from the encoder, be that optical flow information [35,36] or CU map information [13].

For the purposes of this research, it is important to understand the use and generation of a CU map. A CU map is generated by VVC during the encoding process. The CU map is a segmentation map of the frame that has been encoded. The map is used to identify to the decoder where to apply specific filters and to allow for the removal of harsh boarders between regions of the image that have had opposing filters applied to them. By leveraging the information in the CU map as a guide for the filter process though attention, large performance gains can be had [13].

One drawback of these approaches is the time component added when using a neural network to perform such processes on a CPU. These networks must therefore be shrunk, or the fundamental knowledge extracted so that they can be applied more generally.

2.3.11 AI Loop Filtering

The final technique used in video coding that currently leverages AI, is that of replacing the manually coded loop filtering algorithms with AI systems. Much like the two previous techniques creating a dataset is trivial when one has access to the loop filters being replaced, because VVC is being used for this research and the code is open source this is a simple process.

Video coding work in this thesis focuses on state-of-the-art deblocking networks [13]. Deblocking networks were chosen because they take some of the techniques mentioned above and use them to replace the filtering process in VVC, generally improving the filters that were already in place.

Deblocking networks often use similar network architectures as super resolution networks [37], these networks are then combined with attention based neural networks and used to signal the network the regions it must pay attention to.

In short, the network takes in raw Y, U, V components of the video after the normal VVC coding methods have been applied, but before any of the filter processes have been completed. It also takes a flat quantization parameter (QP), which indicates to VVC what level of compression to perform on the video. Additionally, the block structure of each frame is also input into the network and is used as the attention mechanism. The network then outputs YUV components and tries to achieve an image as close to the original image before VVC coding was applied. Figure 4 shows the inputs and output variables of such a NN, Figure 5 shows the NN structure developed to complete the deblocking task.

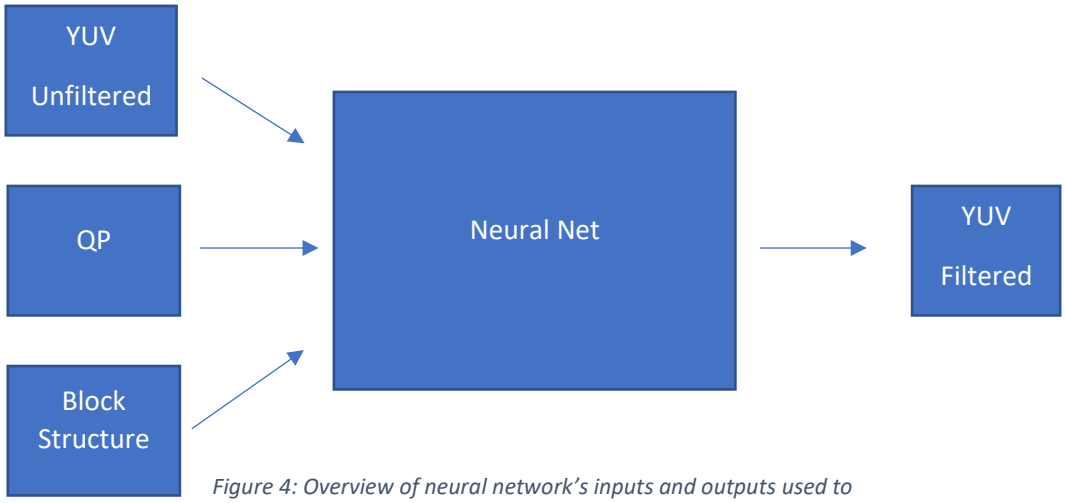


Figure 4: Overview of neural network's inputs and outputs used to implement the machine learning techniques [13].

The network structure in Figure 5 splits up the processing of Y U and V. This is to allow the network to create separate processes for the colour processing and luma processing [13].

The network was trained using the DIV2K dataset [38], the exact method behind making the dataset is not detailed. It is stated the network is trained at 4 QP settings and that the frames to be used for training were selected randomly from the whole dataset and cropped into 48x48 patches. The last main point to take away from this paper is that the loss function used is mean absolute error (MAE), additionally the weighting is such that Y values attribute 4 times the amount of loss than U and V shown in Equation 5. This is done because in the VVC test cases Y PSNR is valued much higher than U and V.

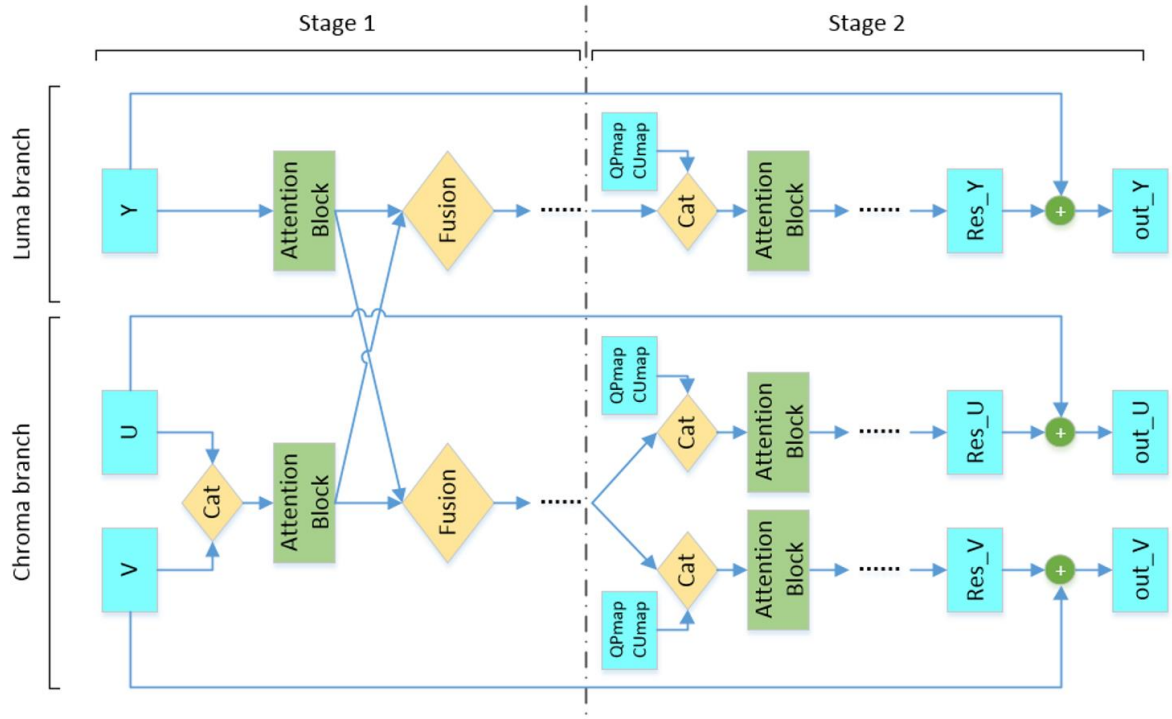


Figure 5: Simplified Neural network structure [13].

$$loss = 4 \times MAE(Y, \hat{Y}) + MAE(U, \hat{U}) + MAE(V, \hat{V})$$

Equation 4: Loss equation used for training the network in Figure 3.

This research shows that this neural net can successfully replace all the filters used in VVC and can even achieve better results than the standard filters. Additionally, the research suggests that while not essential, the QP value and the attention map aid the neural network train faster.

2.3.12 Image restoration

Image restoration is the task of taking a distorted version of an image and returning it to a state where these distortions cannot be perceived. There are many different forms of distortion that can be fixed. Noising, camera distortion and even littering the image with text can be removed with the use of one network [39]. The authors also set out a method whereby the ground truth does not need to be known to train the network. This allows potential for creating an image restoration workflow that can update its output whilst being deployed.

Whilst not a traditional image restoration task (like de-noising), the network in figure 5 performs a very similar task. Essentially frames are intercepted before being fully decoded and the network attempts to restore the image at this intermediate point. Because the ground truth is known, the unfiltered, blocky image represents “noise”, and the aim is the original, ground truth frame that was fed into the encoder.

When it comes to image restoration there are a few network architectures that lend themselves well to the task, residual networks seem to appear frequently [40]. The thought process behind this approach is that there is already a lot of information that you require for the output in the input image. This makes it possible to leverage a residual network, the network will incrementally add data back to the image.

Another type of network used in image restoration is a generative adversarial network (GAN) [123]. GAN's are very good at this task as they create data where none existed before. This, combined with the adversarial nature of the network, allows for very specific tuning depending on the type of distortion being removed.

Encoder decoder networks are often used in the similar task of image super resolution and are successful because of their ability to identify image features at different levels in the encoder/decoder architecture [39,41]. This allows for many textures and patterns to be learnt by the network and leveraged to restore the image.

The network structure replicated for testing in chapter 3 is a residual network at its core, but it has an additional aspect of attention added to the residual blocks [13]. The attention implemented allows the network to adjust the contribution of individual filters depending on the input, this allows for residual filters to become highly specialised without penalising the network for learning edge case scenarios.

2.4 VIDEO METRICS

An important point that has not been addressed yet is that of video metrics, when comparing two images there are many metrics that can be used to compare the similarity between the two images, all these metrics are focused on measuring different specific aspects of the image, a list is below.

Video metrics are often used directly by machine learning systems as part of the loss function. For example, the work in sections 4 and 5 both use MAE as a loss function. However, each metric accounts for different aspects of the image and so a model trained for MAE might perform poorly when measured using a different metric.

2.4.1 Mean Squared Error

Mean Squared Error (MSE) is a widely used metric for measuring the quality of a predictor, it represents the average of the squares of the errors between the predicted values and the actual values. The MSE quantifies the difference between the estimator and what is estimated, offering a way to measure the performance of a prediction model or estimator on a continuous scale.

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2$$

Equation 5: Mean squared error loss equation. A vector of n predictions, Y are ground truth values $\hat{Y}$ are predicted values. n is the number of observations.

2.4.2 Mean Absolute Error

Mean Absolute Error (MAE) is a metric used to measure the accuracy of predictions, it calculates the average magnitude of errors in a set of predictions, without considering their direction. The MAE is the mean of the absolute values of the individual prediction errors on a set of predictions and their corresponding true values. This metric provides an idea of how far the predictions are from the actual values, on average.

$$MAE = \frac{\sum_{i=1}^{n}\left|Y_i - \hat{Y}_i\right|}{n}$$

Equation 6: Mean Absolute Error loss equation. A vector of n predictions, Y are ground truth values $\hat{Y}$ are predicted values. n is the number of observations.

2.4.3 Peak Signal to Noise Ratio

Peak Signal-to-Noise Ratio (PSNR) quantifies the ratio of the maximum possible power of a signal to the power of corrupting noise that affects the fidelity of its representation. In essence, PSNR is a measure of how accurately an image or video has been reconstructed after compression, compared to the original. The higher the PSNR, the better the quality of the compressed or reconstructed image or video.

$$PSNR = 10 \times \log_{10}\left(\frac{MAX_I^2}{MSE}\right) = 20 \times \log_{10}\left(\frac{MAX_I}{\sqrt{MSE}}\right) = 20 \times \log_{10}(MAX_I) - 10\log_{10}(MSE)$$

Equation 7: Peak signal to noise ratio equation. MSE is mean squared error, MAX is the maximum value possible in the dataset.

2.4.4 Structural Similarity Index

Structural Similarity Index (SSIM) is an advanced metric used to measure the similarity between two images. Developed to provide a more accurate and comprehensive way to assess the perceived quality of digital images and videos, SSIM considers changes in structural information, luminance, and contrast, rather than relying solely on pixel differences. This approach allows SSIM to align with human visual perception more closely.

$$SSIM(x, y) = \frac{(2\mu_x\mu_y + c_1) + (2\sigma_{xy} + c_2)}{(\mu_x^2+\mu_y^2+c_1)(\sigma_x^2 + \sigma_y^2 + c_2)}$$

Equation 12: Structural Similarity Index equation. x and y are the two images being compared μ_x and μ_y are the average intensity values of x and y. σ_x^2 σ_y^2 are the variance of x and y. σ_{xy} is the covariance of x and y. $c_1 c_2$ are constants used to stabilize the division.

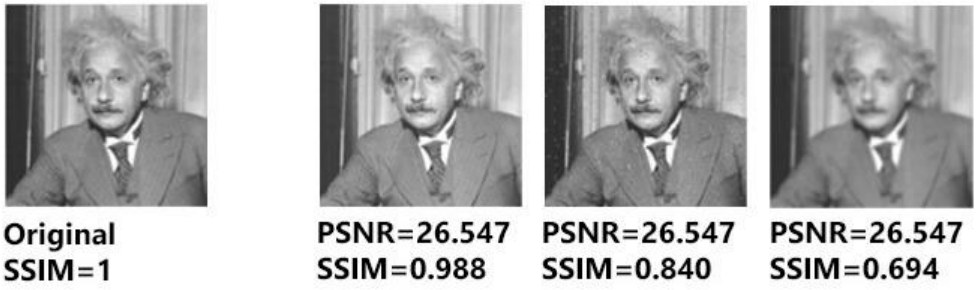


Figure 6: Visualization of PSNR maintaining a consistent value and SSIM changing, indicating how quantitative metrics and qualitative metrics can be disconnected from each other [158].

3 DEVELOPMENT OF PRUNING ALGORITHM AND INTERPRETATION TOOLS

3.1 EXPLANATION OF MACHINE LEARNING IMAGING M.L.I.

Trying to explain the predictions of AI with the current interpretability methodologies has limited scope at best. The methods used are very arbitrary and hyper-focused on the task that the AI is trying to complete [124]. For example, in a text to speech model the metric used simply revolves around whether the AI has predicted the right word, right vs wrong is not explored because the concept of "correctness" of a word in this context is either right or wrong there is no grey area [125]. But the issue is never that simple, even humans often miss-hear things or even understand things very differently simply due to two words having the same pronunciation or spelling.

The problem is best demonstrated in a simple image recognition task: often a confidence score is given to decide what is in a specific image. For example, an AI might be 20% sure a cat is in the image and 80% sure that a dog is in the image, in this case the highest "probability" is taken, and a dog would be selected. But what exactly does it mean to be 80% dog or 20% cat? It seems to be an accepted concept and used as an explanation, but it simply does not make any human sense. However, in the AI world it does. Much like in human perception, when one only has a sense of what the answer to a question could be, one guesses.

Therefore, there is a case to completely rethink the way we currently understand the workings of AI systems. For this reason, this thesis contains development of Machine Learning Imaging MLI. MLI will be a software tool used to perform analysis on a neural network. Unlike many other methods it does not focus on the output and input of the model. This is to ensure this method will application agnostic and therefore have widespread applicability.

MLI focuses on the structure of the neural network, basing all the analysis on the activation functions used, the weights between the layers and the types of structures used. This is particularly advantageous as mathematically all neural networks are just complex graphs. This is because graphs are defined mathematically as "a set of objects, that are related in some sense to each other" often "vertices, nodes and edges" [127]. Therefore, no matter how the field evolves and no matter how many more structures are devised, this form of analysis will always be applicable.

The aim of this methodology is to provide an informative image that represents the usage of the neural network, this structure can be simulated for many different inputs, and the activations of individual neurons can be observed. The idea being that much like Magnetic Resonance Imaging (MRI) machines that are used to identify parts of the human brain that are specific for certain tasks, so too could MLI be used to identify parts of the graph network that are responsible for certain tasks.

3.1.1 Machine Learning Imaging Example Visualisation

For now, the software is in a proof-of-concept stage and only creates a 2D visual representation of the model in question. Figure 7 shows the current visualisation.

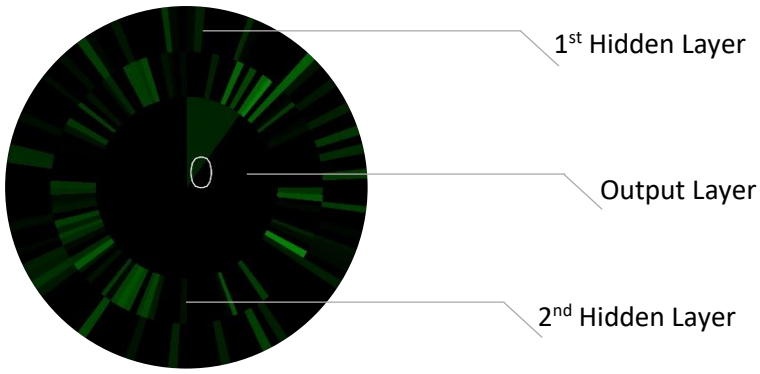


Figure 7: Initial examples of the Machine Learning Imaging visualisations.

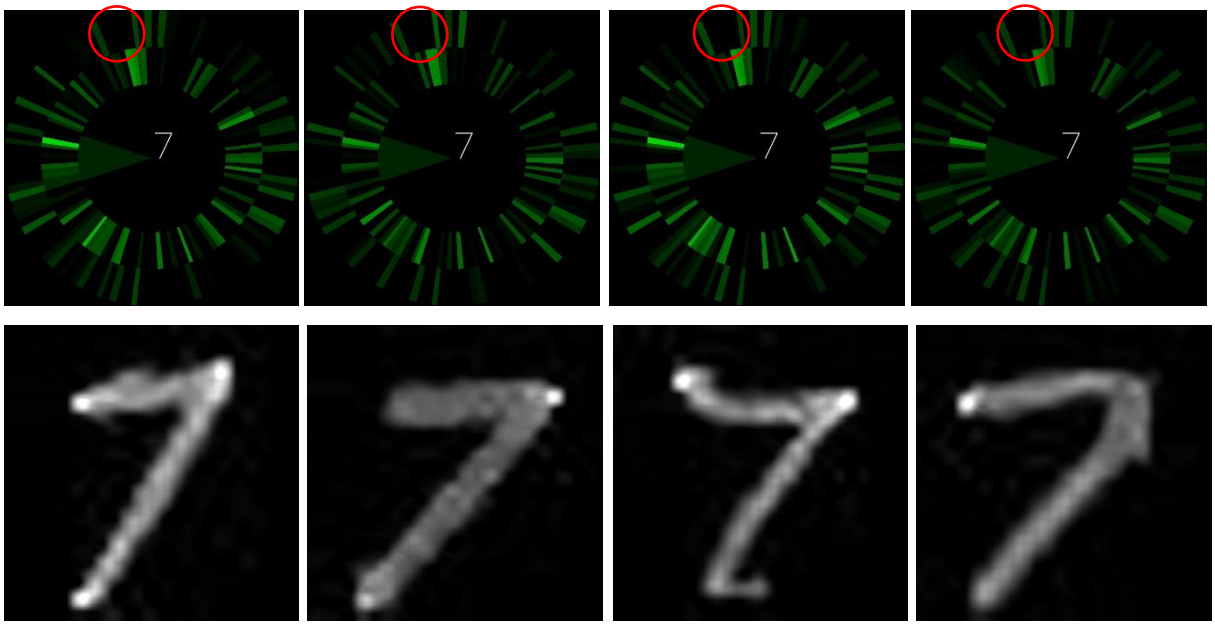


Figure 8: Machine Learning Imaging visualisations (upper row) changing depending on their input (lower row). Circled in red are regions of the network that do not activate for any classification of 7. (Full animation shown in appendix C.2)

The visualisations in Figure 8 show the neuron activations for a given input image. This network was trained to identify drawn numbers between 0-9. Certain classes, 7 in this case, have a "fingerprint", The term "fingerprint" is used because even though the 7's drawn are of different shapes and sizes, they still elicit similar responses from the network. This suggests that the network has perhaps not learnt as many would expect in a cohesive way [128], but rather that there are specific sections of the network that are responsible for each number. The question arises would it be possible to extract these specific parts? Would it be possible to identify the parts of the network responsible for detecting a 7, remove it, and employ it separately from the rest of the network? This process could make neural networks reusable and recyclable, perhaps producing an AI system that are more a part of their sum than a sum of their parts.

Additionally, as can be seen in Figure 8, it's possible to identify "dead" parts of the network that do not perform any specific task. Perhaps by removing these parts of the network, models can become smaller and simpler to train and deploy.

MLI is a visualisation of the network activations and although useful for smaller networks, viewing all the activations of larger networks is impossible, there are too many. Therefore, this method will be used as a tool in the future to assign an importance value to a neuron or filter in a neural network. Low activation values indicate a low importance value and high activation values indicate a high importance value.

3.2 Interpreting Computer Vision Networks

This process is not as simple as it may first seem when trying to interpret an AI system that is based in vision. The standard approach is to use the activations in the neural network to create a map of "interest", this is then visualised (normally as a heat map) back onto the input image. This kind of heat map is shown in Figure 9, in networks that perform classification tasks these maps can give confidence to the user. This is needed so that the user knows the neural network has detected certain characteristics that are important for the specific classification.

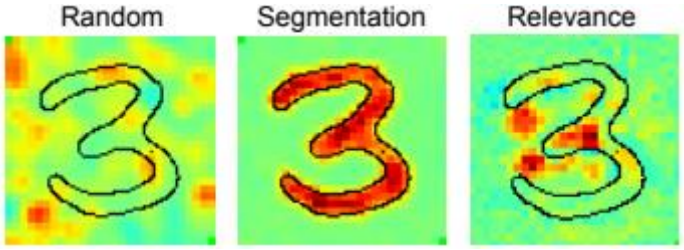


Figure 9: A simple number detector explained by using red pixels to indicate more network interest for the given scenario and green less.

These methods are very effective and can give a high confidence that the network has learnt exactly what differentiates the classifications. In Figure 9 when segmented, the network determines that the parts of the image that identify the three are the exact pixel values, although as this is not good enough as the same pixels would light up for an 8, 0 or 9 [129]. The metric on the far right uses what can be considered a counterfactual argument, by looking at the parts of the image that most strongly disagree with the image being a number that isn't 3. The network has determined that the two parts of the three not being joined together towards the middle determine that it is most probably not an 8. Additionally, the top right not being joined to the middle also removes the possibility of the number being a 9. These regions can be seen in the far-right plot, highlighted by the red regions.

This technique of proving why the network classifies in this way is effective but cannot be applied to all NN tasks directly, this is mainly because classification is commonly not the desired outcome. There is no way to trace importance through the network in the same way for a super-resolution network because the network does not have set "classifications" to compare against [130].

Other methods must therefore be developed, for pixel augmentation tasks like Super Resolution (SR) and Noise Reduction (NR), it is possible to compare the image before filtering and after filtering. This is traditionally done by directly comparing how much the pixel values have changed. This process could be repeated and compared to non-AI approaches that achieve the same task. By combining these visualisations, it may become apparent how a machine learning solution differs from a non-machine learning method [121].

Additionally, an approach has been adopted recently where the convolutions of the neural network have been visually represented [43], this does not require the network to be a "classifier ". The rationale is that the convolutions in the network will inherently contain the structures that the

network is looking for in an image, shown in Figure 10. However, this approach will have to be refined if used for pruning. As there can be thousands of different and separate convolutional parts of a neural network.

Figure 10: Convolutional filters that have been identified as important for a classification of a dog [43].

Another approach is visualisation of the network itself; this approach removes the issue of input and outputs to the network. This method aims to produce a representation of the structure of the neural network. The exact structure will be derived from the weights, biases, connections, and datatype presented to the neural network. The hope being that information can be directly extracted from this representation like overfitting, undertrained networks, redundant layers and neurons, unsuitable network structures [132]. The benefit of using this method is that this information can be learnt from any neural network from the structure, weights, and biases alone. Therefore, interpretation would not have to focus on the input and output of a specific network and could be more universally used.

3.3 Conclusions

This chapter introduces the interpretation method of MLI, this method aims at providing a model and data agnostic approach to determining the utilisation of a neural network. Whilst this kind of visualization is very informative for small networks, the usefulness of such a method will decrease exponentially with model size.

Additionally, whilst this method clearly illustrates the neurons most utilised for a given prediction, it does not give a concrete explanation of what is happening. Sections of the network are specialised for a given class, but explaining the prediction of the network in detail still needs to be addressed.

Because of these two facts, MLI will be used to give a “neuron importance score”, the observations in Figure 8 indicate that some neurons are more important than others. For a full visualization of the network on different classes please visit [1].

This will help solve the issue of having to manually inspect neurons to identify “importance” when dealing with large networks. However, this does not directly assist with explaining the predictions of the network, this concept is tackled in chapter 8.

The neuron importance score will consider how much a single neuron is utilised across multiple inputs by averaging the importance value. This score will be utilised to help guide the pruning process described in chapter 4.

[1] - https://www.youtube.com/watch?v=Lw-WSmSzTJc&ab_channel=JustAVoice

4 JPEG RESTORATION USING CNNS

4.1 PREFACE

This chapter aims to provide a stable and easily retrainable machine learning platform focused on a vision-based task to allow for interpretability and pruning tests. Additionally, these experiments test the ability of a state-of-the-art deblocking neural network [13], to adapt to different types of compression artefacts.

4.2 TRAINING METHODOLOGY

4.2.1 Introduction to JPEG Restoration

Early on it was observed that the kind of image filtering performed by video coding algorithms such as H265 were very similar to the techniques used by the image compression standard JPEG [44]. However, there are some key differences that effect H265 and not JPEG compression. For instance, JPEG works with a fixed window size for processing, as do all previous video encoding techniques. One of H265's innovations rely in part on the ability to vary the size and shape of the window when processing an image [134], this is achievable through the variability in CU size shown in Figure 11.

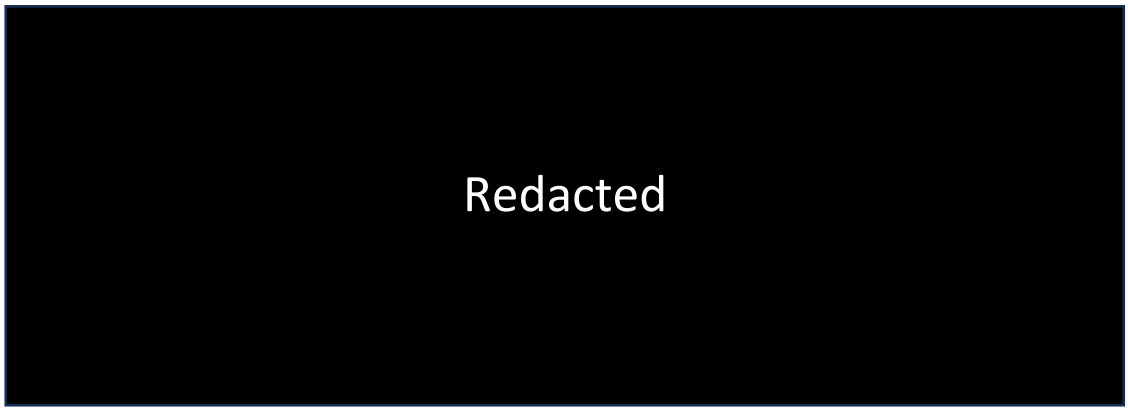


Figure 11 – The left image shows H265 blocking artefacts that are blocky but vary in size. The right image shows JPEG compression artefacts that are consistent square boxes, indicating the difference in the techniques.

JPEG compression works in a fixed 8x8 pixel grid, which is why the artefacts are so much easier to see, especially at a high compression rate. Whilst it is not in the scope of this work to rewrite the whole JPEG standard to use this variable block size for compression, it seems like a very simple way to vastly improve this image compression technique. However, a neural network is not limited to this block size limitation, meaning that filtering can be applied across the whole image and not just the small 8x8 sections.

State-of-the-art deblocking neural networks use the CU map that is generated during the encoding process to aid with artefact removal, this map is not needed for JPEG compression removal [13]. This is because it is completely predictable and unchanging. The network would have nothing to learn by feeding in a segmented 8x8 CU map for the whole image because it would not ever change [135]. Because of this the network proposed for this work uses an Identical structure to that of the state of the art but removes the CU map as an input.

By providing the compression ratio of an image as an input, the network may adapt and be able to process compression ratios that it wasn't explicitly trained for, this is in stark contrast to the other methods developed that use individual networks for each compression level [45,46,47].

4.2.2 Dataset

To train the neural network the DIV2K dataset was used [38], this dataset was split into 48x48 patches. To be comparable to other JPEG compression methods the dataset was compressed using JPEG compression at 6 different compression levels, 10,20,30,40,60 and 80, a lower number indicates a higher compression level.

SET5 was used to evaluate the performance of the network as it is used in many publications related to JPEG compression and image noise removal [136].

DIV2K contains 800 train images which when encoded at the compression ratios of 10,20,30,40,60 and 80 gave a total of 4800 potential images for training, Figure 12 shows the data processing to create this training set. The images were encoded using readily available JPEG compression libraries in python, and the PSNR measurements for SET5 under all compression ratios were compared against academic publications and confirmed to be identical [45].

Figure 12: Dataflow in the image augmentation process for compressed JPEG images.

These 4800 images were split into two datasets, one from now on referred to as “the low-end”, comprised of compression ratios 10,20,30 and 40. The other, from now on referred to as “the broad range”, comprised of the compression ratios 10,40,60 and 80. Each of these datasets contained 3600 images each.

The images were additionally converted into the YUV colour space, this was replicated to match the logic of other deblocking networks which highly valued the Y channel as it contains 2 times the information of the other two channels combined [13].

The Compression Ratio (CR) was presented to the network as a decimal between 1 and 0, this was calculated by dividing the actual JPEG compression ratio, which can be any value between 1 and 100, and dividing by the maximum value of 100. This value was then made into a 2D array the same size as the input image and use as an input where normally the QP and CU maps would have been.

4.2.3 Neural Network Structure

For these tests a SOTA model was adopted, however a few changes were made to the inputs, the CU maps and QP were removed as inputs and replaced by CR [13].

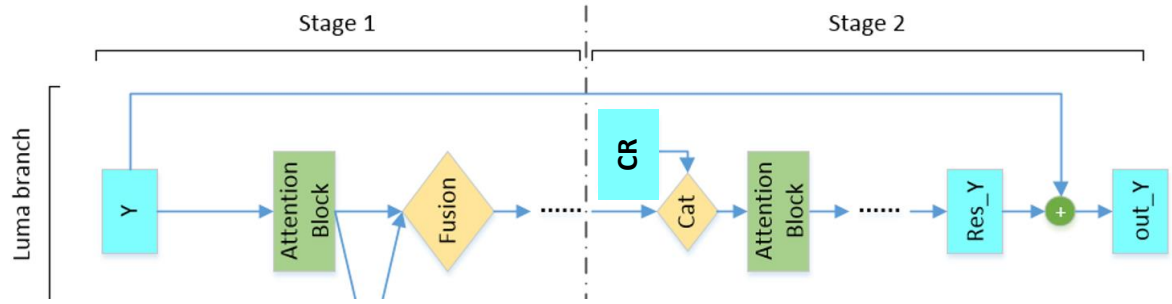


Figure 13: Separated luma (Y) branch of the network that will be replicated to achieve the task of JPEG artefact removal [13].

Additionally, the tests performed only included the Y branch of the original network.This is because the tests performed on JPEG artefacts commonly compared in papers are greyscale and hence have no colour information. This model structure is shown in Figure 13.

4.2.4 Training

The Adam optimiser was used with a learning rate of $1x10^{-3}$ and mean absolute error was selected as the loss function. The network was trained for 100 epochs with a batch size of 128 on the low-end dataset and then again from scratch on the board-range dataset.

This resulted in 2 networks, one highly specialised at very compressed JEPG images and the other which could handle a broader range of compression ratios.

The network was trained on two RTX2080's, two additional models were trained on only one compression setting, this was trained on the compression rate of 10 and 40. This was to test if the model would perform better when only trained on one compression ratio.

4.3 RESULTS

Below is a table of results compared to a few other current SOTA methods performing JPEG artefact removal on a greyscale version on the SET5 dataset.

*Table 1: Comparison against SOTA methods of JPEG artefact removal at various compression ratios best results are in **bold** 2nd best underlined*

QF	Method	PSNR
10	JPEG	27.82
	ARCNN [45]	29.03
	MemNet [46]	**29.69**
	RNAN [47]	29.63
	Ours (Low-End)	29.67
20	JPEG	30.12
	ARCNN	31.15
	MemNet	31.90
	RNAN	**32.03**
	Ours (Low-End)	31.90
30	JPEG	31.48
	ARCNN	32.51
	MemNet	32.97
	RNAN	**33.45**
	Ours (Low-End)	33.17
40	JPEG	32.43
	ARCNN	32.68
	MemNet	33.86
	RNAN	**34.47**
	Ours (Low-End)	34.02

As it can be seen our results are beaten by RNAN methods for all the experiments except the compression ratio of 10, MemNet is better at compression ratios of 20 and 10 but is beaten by our method at ratios of 30 and 40.

However, this is not an entirely fair comparison as these models all have vastly different model sizes as seen in table 2. Additionally, all the other approaches had individual models trained for each compression ratio, meaning 4 models were trained to achieve the same task of our singular model. Training times would still be identical as all models have a similar amount of total training data.

Table 2: Parameter comparison for SOTA methods, the last column considers the fact that all other methods require a model for each compression ratio.

Model	Num of Parameters	Num of Parameters for all models
ARCNN	106k	424k
MemNet	677K	2.7M
Ours	879k	879k
RNAN	7.4M	29.6M

These results for all models suggest that whilst our network can perform with a similar accuracy to MemNet, there are less parameters in our network. Additionally, whilst RNAN has better results than our network it uses more than 7 times the number of parameters.

Testing the network did raise another interesting question, what happens if you input in a different compression ratio to the ML model, than what you know the image to have been compressed at?

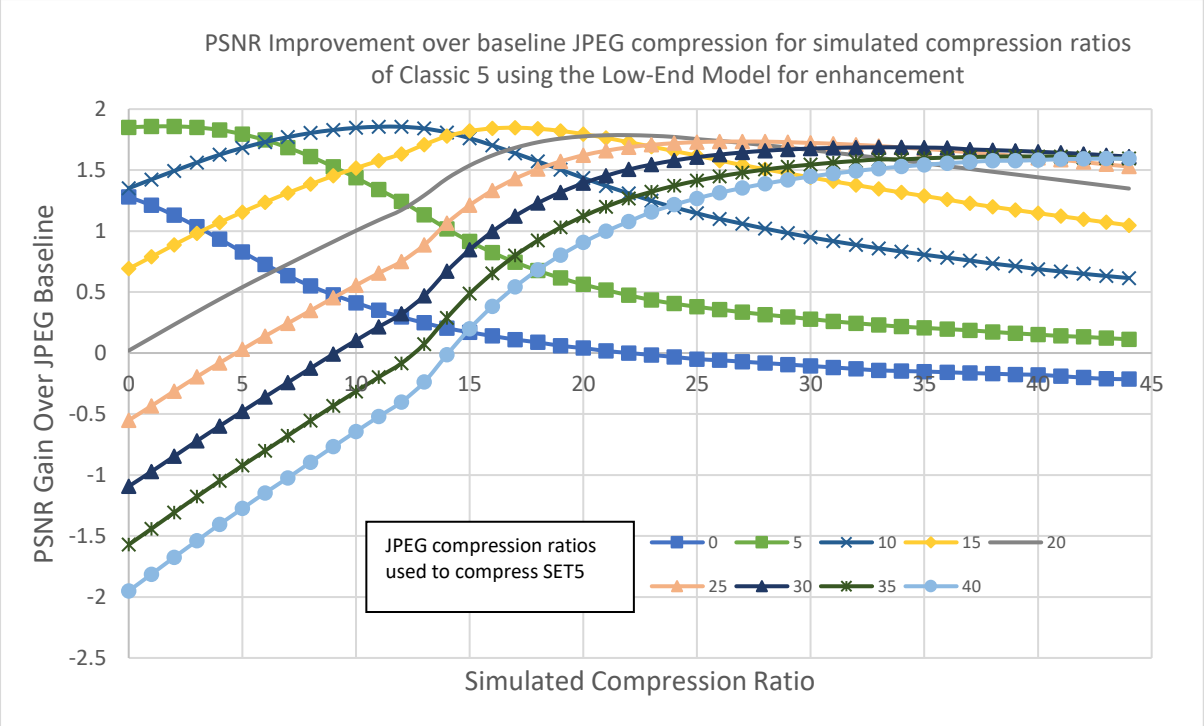


Figure 14: Results of applying the low-end model to reduce compression artefacts of images at compression ratios 0,5,10,15,20,25,30 and 40. (Appendix G shows a similar plot with all compression ratios).

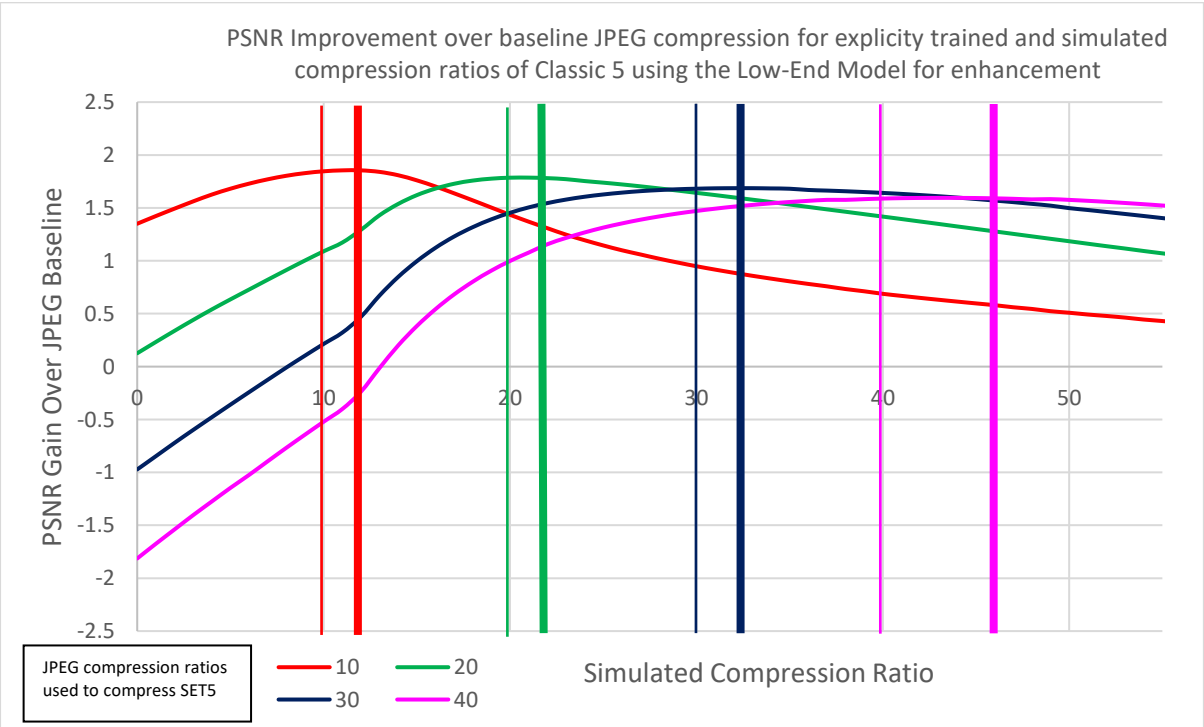


Figure 15: Results of applying the low-end model to reduce compression artefacts of images at commonly used compression ratios

As can be observed from Figure 15 and 14 if an image is decoded at a PSNR ratio that it wasn't encoded at, the neural network can produce worse results than the baseline JPEG compression. This is shown by the negative regions in the graphs. However, this is easily avoidable as all JPEG images have the compression ratio stored in the file structure, and this is easily retrieved and can be input to the machine learning system.

The more perplexing result is Figure 15, this shows that even when the correct compression ratio is chosen (indicated by the thinner lines at compression ratios 10,20,30 and 40), it is not the ideal compression ratio to present to the network for best results. The thicker coloured line for each of the compression ratio shows the point at which the best PSNR results were obtained.

For a compression ratio of 10, this ideal ratio occurs at a simulated ratio of 12, for 20, 21 for 30, 32 and for 40, 46. It is also important to note that this simulated ratio was applied to decode the whole of the set5 test set. However, it can be assumed that each individual image has an ideal compression ratio associated with it, to be best decoded at by the neural network.

Future research could predict the compression level presented to the neural network given an input image. Additionally, there is no reason to assume that the whole image will have the same ideal compression ratio, it is known that JPEG is better at compressing some features in images than others [48]. By training a small network that could learn this ideal value for sections of a whole image, it would not only remove the limitation of having to know the exact compression ratio of the image, but it would also maximise model performance.

This phenomenon is even more pronounced when analysing the broad range results in figure 16, the images compressed at a ratio of 10, have the best result when processed with a simulated ratio of 2. 40 has its best ratio at 33, 60 at 55 and 80 at 74. It's interesting that all the best results are shifted backwards from their actual encoded number instead of forward like the low-end results. It's also interesting that the shift seems to be far more pronounced.

I hypothesise that the greater distance from the actual compression ratio shown by the low-end would indicate that the JPEG algorithm has specific techniques in place that operate in a cut off fashion instead of a slow additive one. This is confirmed by the JPEG standard [44] where the algorithms for the different levels of compression ratio are explained.

It is also important to note that as a biproduct of this, the performance on the commonly tested compression ratios of 10 and 20 is notably worse, however the ability of the model to adjust to have good performance at 30 and 40 is impressive. The fact the model was not explicitly trained for these compression ratios but still has good performance confirms that the model does have the ability to learn the JPEG compression problem in a generic way.

Table 3: Comparison of a model trained on a broader range of image compression ratios compared with a model trained on a more tightly grouped range of image compression ratios.

QF	Method	PSNR
10	Ours (Broad-Range)	29.45
	Ours (Low-End)	29.67
20	Ours (Broad-Range)	31.81
	Ours (Low-End)	31.90
30	Ours (Broad-Range)	33.14
	Ours (Low-End)	33.17
40	Ours (Broad-Range)	34.02
	Ours (Low-End)	34.02

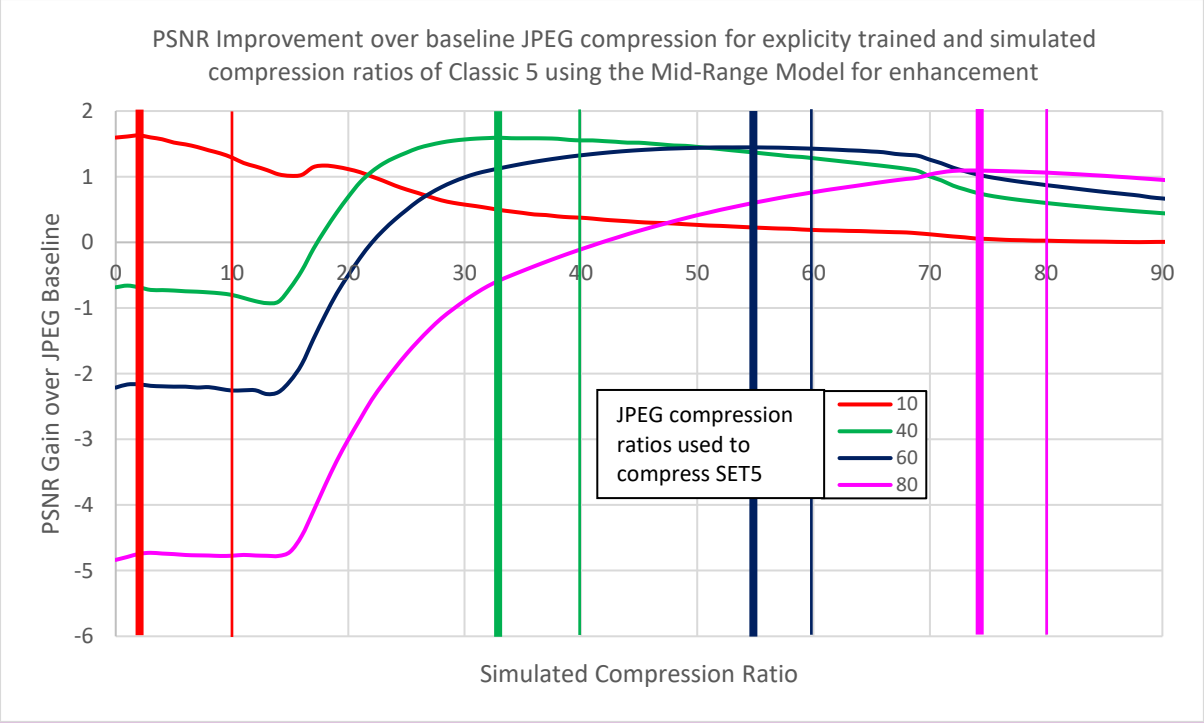


Figure 16: Results of applying the broad range model to reduce compression artefacts using the compression ratios that the model was trained for.

Additionally, the models were tested that were trained for one specific compression ratio. The table below shows the results for two models compared to the low-end training results.

Table 4: Comparing training a model for one specific compression ratio against training a model for multiple compression ratios.

QF	Method	PSNR
10	Ours (Specifically trained)	29.66
	Ours (Low-End)	29.67
40	Ours (Specifically Trained)	33.99
	Ours (Low-End)	34.02

The results for the specifically trained models were nearly identical but slightly worse than the results from the Low-End model. This suggests that the low-end model has learnt a better approach to restoring JPEG artefacts from observing multiple identical inputs at various compression ratios. This result was highly unexpected, it makes an extremely good argument that these kind of restoration networks should be trained on multiple compression ratios distilled into one model, instead of focusing on one compression ratio and training many models.

Research has now moved in the direction of controllable restoration [137], the kind of methodology used in this new research is highly correlated with the research in this section. The fundamental idea of controllable restoration is training a model for multiple types of image degradation, be that noise or compression artefacts. These are all applied at different levels during the training so that the end user can decide the level of restoration they wish to apply [137].

4.4 Interpreting Vision Based Neural Networks with a View to Network Reduction

To test the hypothesis that some neural networks can be underutilised [139], reinforced by the previous tests on MNIST, it was decided that feature maps of the trained JPEG model should be checked to observe their utilisation. Feature maps shown in this section have a range of importance scores, these scores were calculated using MLI.

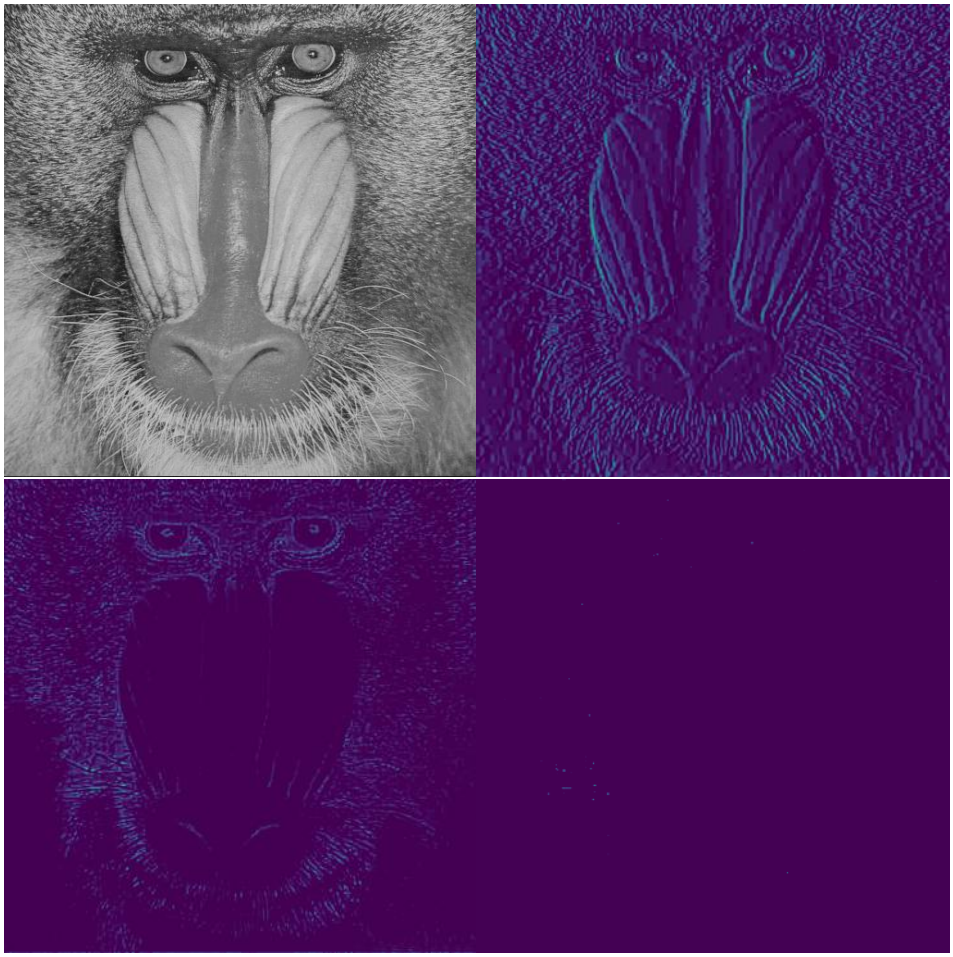


Figure 17: Visualisations of multiple filters from the trained mode given one input image. Input image (Top Left) Filter 43 (Top Right – Importance score 0.7) Filter 25 (Bottom Left – Importance Score 0.3) and Filter 30 (Bottom Right – Importance score 0.015) for conv layer 11

By inspecting the feature maps, we found highly specialised filters at multiple levels in the neural network. In Figure 17 43 looks like a standard edge detector, whilst filter 25 looks like a basic hair detector.

Figure 18 has vastly more specialised filters, filter 84 seems to be turning individual 8x8 blocks on or off based on some metric the network has found. This can be assumed to be because of the 8x8 processing that JEPG applies, and the different approaches the algorithm has utilised depending on the contents of the image. And in Figure 18 filter 33 seems to be a highly specialised 45° hair detector.

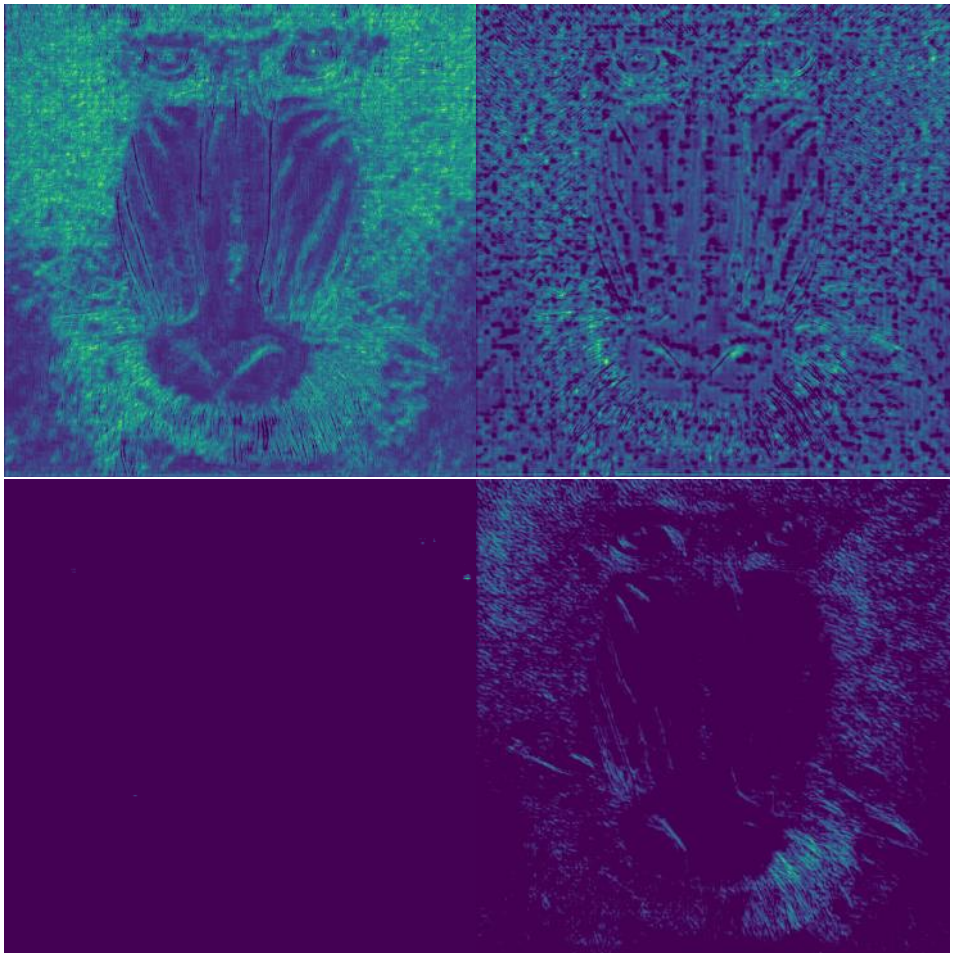


Figure 18: Visualisations of multiple filters from the trained model, located deeper into the neural networks structure. Filter 68 (Top Left – Importance Score 0.95), Filter 84 (Top Right – Importance Score 0.8), Filter 64 (Bottom Left – Importance Score 0.01) and Bottom Right Filter 33 at conv layer 107 – Importance Score 0.3.

Additionally, we observe filters that are barely used, filter 30 in figure 18 and filter 64 in figure 19 both indicate low utilisation by having little to no output.

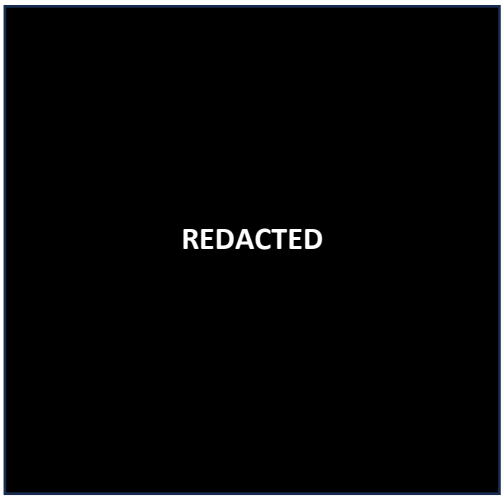


Figure 19: Input image that produces the filters in figure 20.

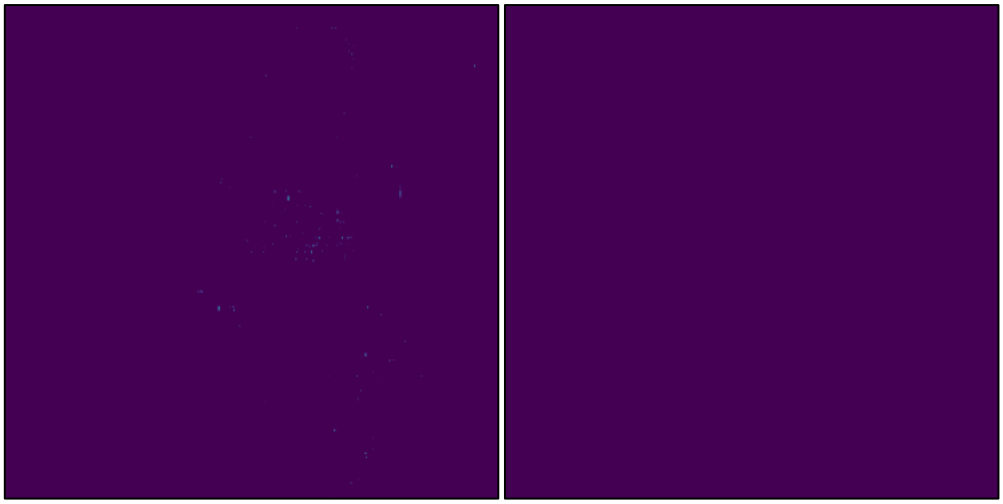


Figure 20: Visualisations of two filters that were determined to be minimally activating from the input shown in figure 17. Filters 30 from layer 11 (bottom Left – Importance Score 0.015) and filter 64 from layer 107 (bottom right – 0.01) are once again shown to have little to no activations present when using the input in figure 19.

The observations in Figure 20, confirm that the observations from the MNIST tests are present in larger vision based neural networks. These observations also show that non-classification networks are also underutilised, and hence redundant components may possibly be removed.

4.5 Conclusion

Primarily in this section we have proven that the network presented in previous works [13], in have an architecture that is transferable and can be used to reduce JPEG image compression artefacts. We achieve results that are comparable to SOTA models of a similar size, however larger SOTA models outperform our solution by a significant degree.

However, whereas in other papers 4 models are trained for each individual compression ratios [46], we only train one network, and guide the level of restoration by using the compression ratio as an input. This means that our one model solution is far more applicable to real world scenarios, we also show that the model achieves a level of generality when trained over a larger range of compression ratios, allowing it to remove artefacts from images it wasn't specifically trained for. Additionally, we show training for one compression ratio at a time as opposed to many has little to no impact on performance.

We could make the claim that because of this our network is 4 times more efficient, however I think there is a more important takeaway. Instead, I would recommend that future networks of this type adopt this method and include the compression ratio in the input data. It has little to no impact on model performance and it shrinks the footprint of the models saved on hard drives by 4 times.

Secondarily we have investigated the utilisation of non-classification computer vision-based models. We have found that these kinds of models are (like classification models) underutilised. This result suggests that pruning should be applied to large computer vision networks to reduce memory usage, inference time and achieve other savings possible through neural network pruning.

5 Pruning a Video Deblocking Neural Network

5.1 Preface

The following body of work aims at replicating the achievements of previous SOTA model ACDNN [13] and then pruning this replicated network to test the viability of a pruning algorithm developed alongside M.L.I. A streamlined version of this work was published in ISCAS2022 [111] and provides a robust method for pruning this type of network.

5.2 Pruning Methodology

In the previous section describing MLI we showed that for certain classifications, specific neurons are not used and can be considered “dead”, this raised the question as to if there are neurons that generally do less work than others.

5.2.1 Visualizing convolutional layers

It is common in visual AI tasks to visualise the convolutional filters in the network to try and infer the structures, textures and ultimately the objects that have been learnt by the neural network.

It was determined that most of the model’s filters were performing very few alterations to the output of the model, shown in Figure 21.

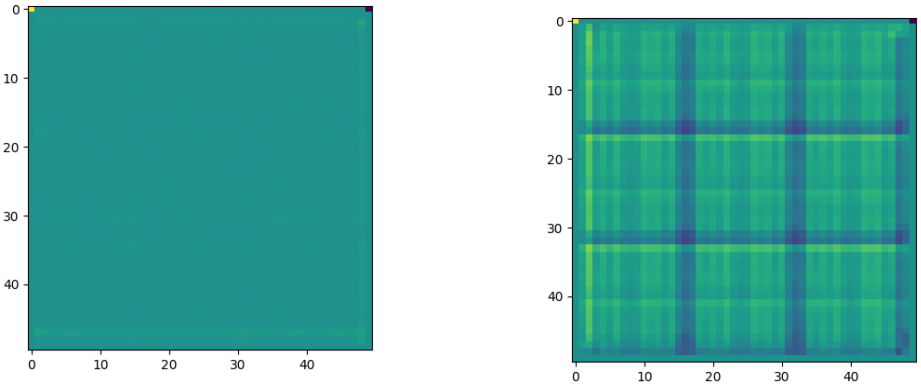


Figure 21: A minimally activating feature map in the first convolutional layer of the model (Left) and a nominally activating feature map (Right).

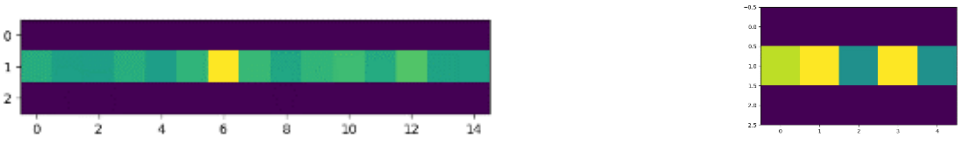


Figure 22: Comparison between 2 fully connected layers, the one on the left shows very small activations over a large output space, whereas the one on the right shows high activity over a smaller output space.

This phenomenon is also found in the dense layers, shown in Figure 22, where some neurons in the layer are used much more than others.

It was found that there was an abundance of filters and neurons that did not seem to produce a large magnitude output. To test the hypothesis that these filters and neurons were redundant they were removed, but this approach was not fruitful. This was because even though the impact of these filters seemed small, they still were fundamental to the overall output of the neural network.

To leverage these "dead" neurons and filters a more refined way to identify and remove filters and other structures in the network need to be developed, in a way that did not impact the performance of the model.

5.2.2 Filter Pruning Methodology for Neural Networks

The method used to aid with pruning in this case was sparsity pruning, for our purposes it made the most sense to use a readily available pruning library, and so the TensorFlow optimization toolkit was used [49]. This toolkit allows sparsity to be applied to any TensorFlow model, this was important because if this method would prove successful in reducing complexity of our neural network, then we wanted a platform that could easily be transferred to other networks. To check if this pruning method was limited to this model or if it was more far reaching.

After the model has been trained to an acceptable accuracy level, standard TensorFlow sparsity pruning is applied. However, sparsity pruning is not applied "model wide", for reasons explained later it may not be possible to apply pruning on certain layers. Additionally, there may be layers that do not make reasonable sense to prune, for example input and output layers. This is because the function of these layers is so directly linked to the data and task of the network. No useful network reduction can be achieved by pruning them unless the data is highly specific and sanitised [158].

As mentioned in the literature review, sparsity pruning does not guarantee all weights around a specific neuron will be zero. However, neurons that have a low importance score after sparsity pruning, can more easily be removed from the network without affecting accuracy.

To identify specific neurons for removal, firstly the network was sliced at multiple stages. The points where the network was sliced was decided before this step was undertaken. This was done manually and was predicated on the structure of this specific network and would have to be decided by the architect of any complex neural network. The reason for slicing the network in this fashion was to easily obtain the activations of the neurons at the layers we were interested in pruning.

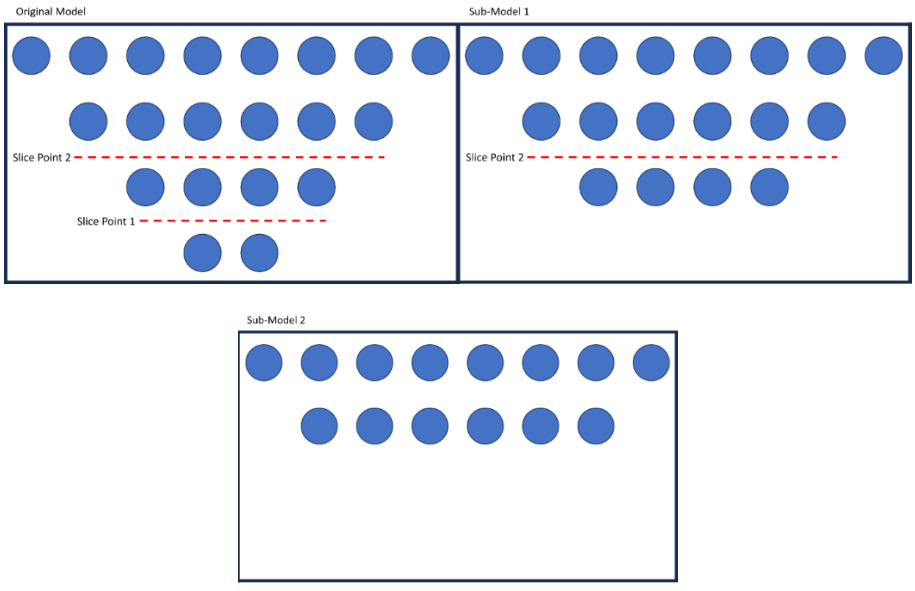


Figure 23: Model slicing applied to aid with gathering neuron activations at predefined "pruning points".

The slicing process produces N sub models, where N is the number of layers where we will attempt to physically remove neurons. Once the sub models have been created, neurons can be identified for removal.

To identify the neurons that can be removed from the network 10% of the test set was passed into the network, the test set has also been patched and split in the same way that the train set has been. Then the filter activations for each channel at each cut point was recorded, these values were then added together and averaged over the total amount of patches and the size of the patch that were input into the model. This produced a patch at each cut point that indicated the amount on average that the filter was used. For fully connected layers this process was the same, however as the number of neurons in the layers do not change depending on patch size, like convolutional filters, the output was simply averaged for each neuron over the number of examples.

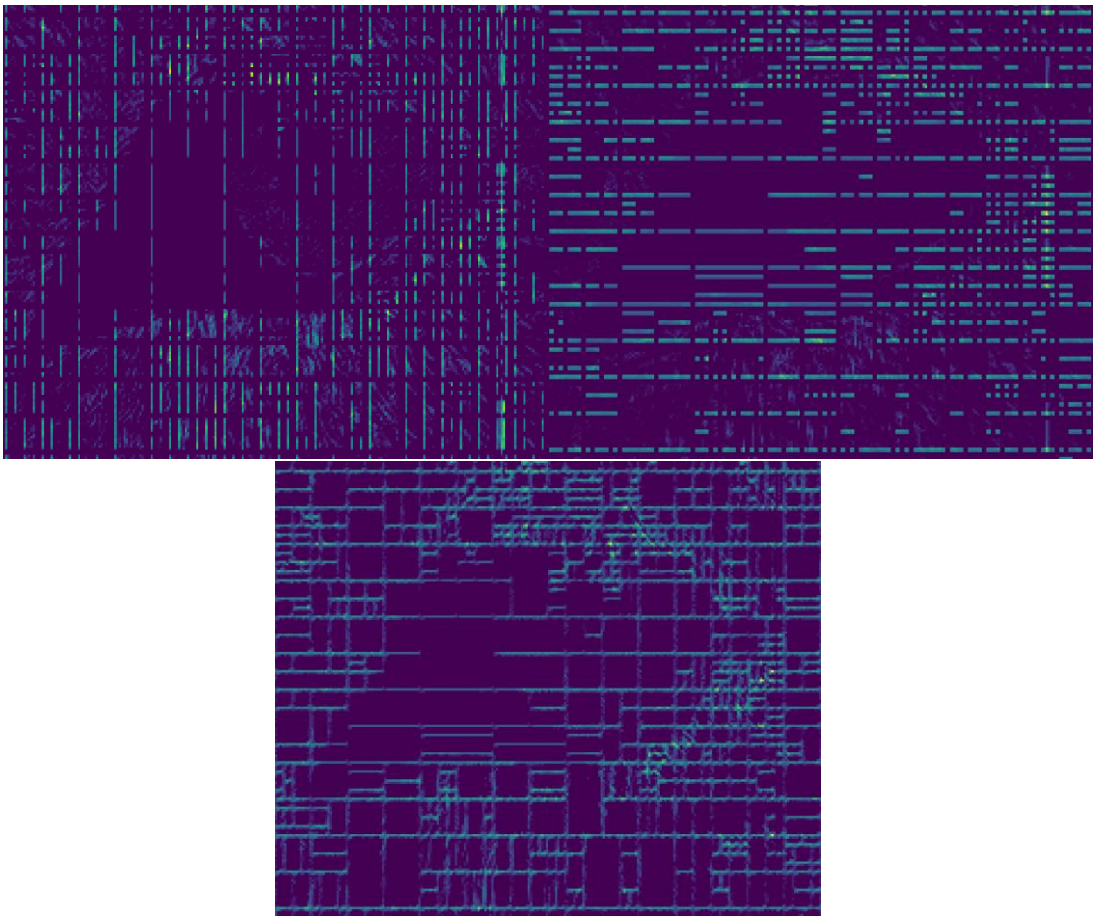


Figure 24: The upper filters are from the 3rd layer of the NN and can be seen identifying the vertical (left) and horizontal (right) CU components. Then the lower filter in the 4th layer combines both with parts of the original image to get a CU map that represents part of the image where deblocking needs to be applied.

Once an average importance value had been obtained for both fully connected neurons and for convolutional filters, some could be selected for removal. For fully connected layers this process was trivial, as the position of these are always static and, in some ways, independent of the input. A simple thresholding value was set for these. If a neuron were to fall below this value, it would be removed from the dense layer. Convolutional filters were more of an issue, at first, L2 Norm was used as a metric, but because the outputs of these filters were so input dependent these approaches did not work.

Additionally, as seen in Figure 24, sometimes a filter can become very focused on an input to the network, in this case, a CU map. This is exactly the sort of behaviour we expected and wanted from a network performing a deblocking task, as often blocking artefacts are worse at CU boarders. We do not want to prune these filters as they seem essential for the task at hand.

Figure 25: Input image use to generate the visualisations in figure 30.

A method had to be created that did not accidently trim these advantageous filters. The realisation that eventually led to the correct method for trimming filters was that the network was dealing directly with pixel information.

The network inputs and outputs pixels at a colour bit rate of 10. However, these have been scaled for the network between values of 1 and 0. Therefore it can be supposed that the lowest meaningful value in the network must be the lowest pixel representation available.

$$Max\ Pixel\ Value = 2^{10} = 1024$$

$$Smallest\ Pixel\ Represenation = \frac{1}{1024} = 0.0009765625$$

Equation 8: Maximum pixel value used to calculate the smallest meaningful pixel representation inside the model.

For simplicity's sake the value was rounded to 0.0001. Further research needs to investigate more refined methods to perform the final identification step, however for now, the averaged filter output is simply checked for values below 0.0001. If all values in the averaged filter are below this threshold, then the filter is removed, if the pruning loop runs and no filters or fully connected neurons are identified for removal then the threshold value is increased by 0.0001, but only if no filters are identified for removal, otherwise the value returns to 0.0001. This is done to not remove filters at an overly aggressive rate.

5.2.3 Network Architectural considerations

Because ideally, we would like this methodology to be transferable to other neural networks, a general case had to be developed of each layer-to-layer connection. Whilst we cannot predict how any amount of machine learning engineers might decide to form their networks, we can predict the most likely kinds of connections required.

5.2.3.1 Non-Learning Layers

These layers are effectively ignored in our pruning process, these are layers like GAP, Gaussian Noise, Max/Min, Multiply and Add. These functions and ones like them represent unlearnable characteristics, they are simply added to models to combine outputs or to apply pre-defined processing. These layers cannot be controlled by the network's learning process and as such it makes little to no sense attempting to interpret and leverage their output. We already know what they are doing, and they will exist in a network for a specific reason that has been identified by the researcher.

5.2.3.2 Learning Layers

These are any layer that the network's learning process has direct control over, Convolutions and Dense layers are the main form of these layers. However, these have many subcategories that must be catered for, i.e., Dimensionality 2D,3D,4D.

5.2.3.2.1 Dense Layers

When removing a neuron from a dense layer it is required to remove all weight and bias references from the previous layer and the subsequent layer. This does change for dense layers, depending on the number of neurons in the subsequent and previous layer, the number of connections that need to be trimmed change.

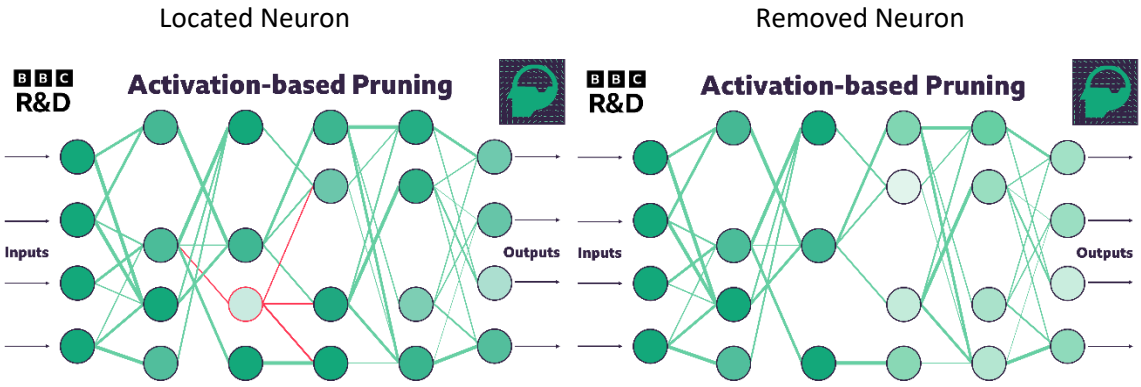


Figure 26: Visualisation of the proposed method to remove a neuron in a dense layer (here the neuron is red) additionally all red weights shown were also removed. (Full animation shown in appendix B.5)

5.2.3.2.2 Convolutional Layers

Convolutional layers are more complicated, in a convolution you have a concept of a kernel. This is a 2D grid of size X by X that has been predetermined by the network's architect, this grid will have a set of values associated with it, and each make up a kernel. Each channel will have n kernels, where n is the amount of input channels of the convolution. Each of these kernels is passed over the input channels and summed together with a bias to give an output channel. This brings up some issues, because not only has the kernel overlapped multiple times with itself it has also been combined with other kernels and a bias value to give an output to the channel. Because of this kernel trimming is nearly impossible, not only because the number of kernels depends on the input to the layer. But also because of the sheer amount of data mixing between the layers and channels that occurs.

Due to this fact we only attempt to trim entire channels from the output of convolutional layers. This simplifies the process immensely and allows for large gains in network calculations.

5.2.3.2.3 Residual Layers

Residual layers add another layer of complexity, this is because of the nature of residual layers. They aim to pass through most of the data that is passed to them. In networks that deal with pixel

augmentation it's not needed to change the input that much; only small augmentations need to be made. However, this kind of architecture creates an issue, in that, the data input to the residual layer must be added to the output of the layer. This means that the dimensionality must match at the input and output. Normally this is not an issue, but when pruning is applied, it is possible that a convolutional or fully connected layer inside the residual layer is selected for pruning, this can cause dimensional mismatches.

There is a very quick, and simple solution, which is to simply trim the same channel that was trimmed by the convolutional pruning from the input, but this isn't without issue. In the case where many residual blocks are chained together, often standard in these sorts of networks, they will no longer be able to access this channel because it has been trimmed away. And whilst one specific residual block might not use one specific channel there is no guarantee that the next will not.

Because of this we opted to retain the channels that were lost by any convolutional pruning and instead these channels were just re-concatenated back into the main branch. This allowed for trimming of these residual layers without heavily impacting the output of the model.

5.3 Pruning Applied To a Deblocking Neural Network

The network used in the SOTA [13], processes all 3 components of an image Y U and V. For the purposes of this work this network was split into 3 individual models. This was done because it was surmised that each component needs a different amount of processing applied to them, and so pruning a network to deal with all components individually would allow each individual network to adapt to the complexity of the data.

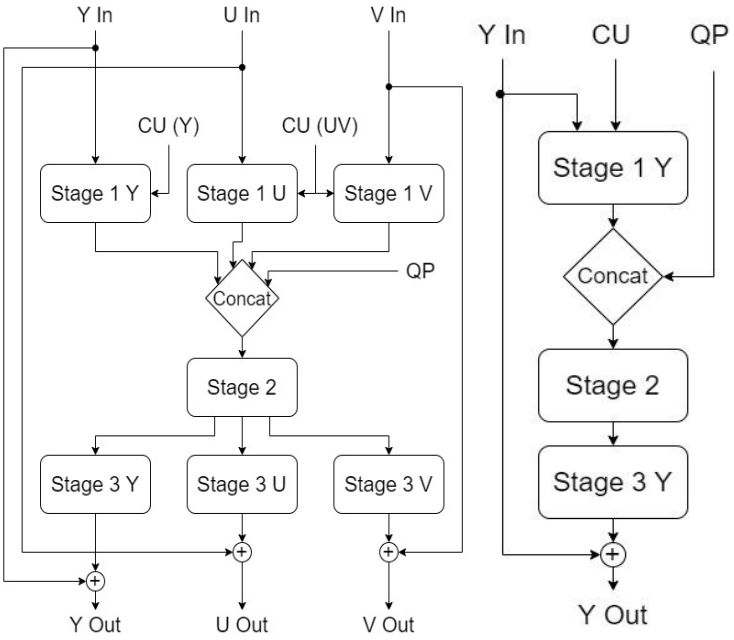


Figure 27: General structure of the ADCNN model [13]. b) Network structure for a channel after separating the Y, U, V network into three UCLF networks.

These three models were named Uni-Component Loop Filter (UCLF), shown in Figure 33 for the Y U and V channels, all of these have the same architecture. These architectures are separated into three stages, each stage consists of generalised residual and non-residual blocks. The residual block structure is displayed in Figure 34 with two convolutional layers (2DConv) followed by two dense layers. Non-residual blocks follow the same structure as residual ones, without the input being

added to the output. Each model was individually pruned using the pruning algorithm that uses a combination of sparsity and structured network pruning.

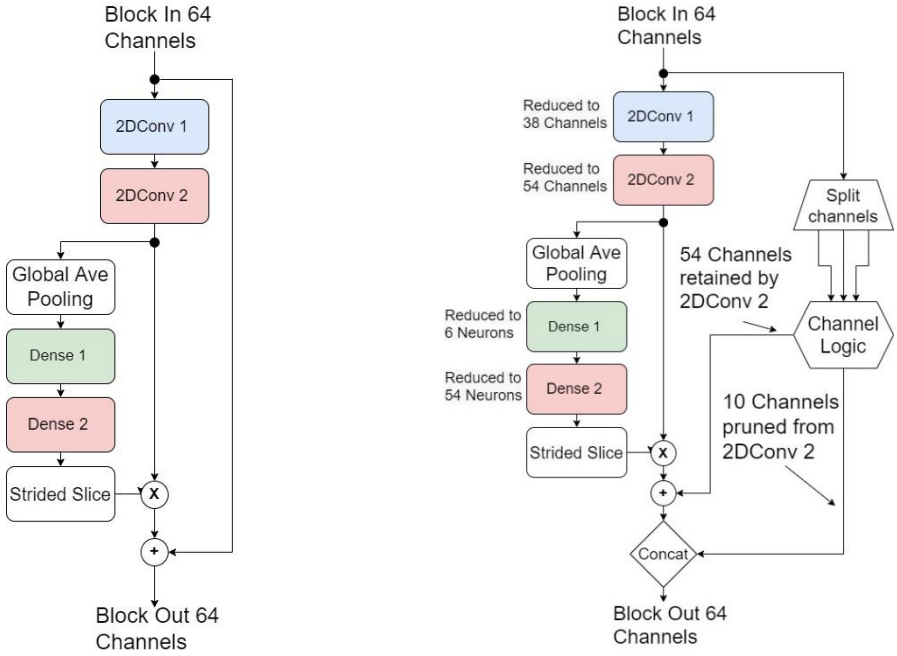


Figure 28: a) General structure of the residual blocks in each stage. b) An example of the proposed structured pruning method applied to a residual block.

5.3.1 Pruning Algorithm

The algorithm takes a pre-trained model as a starting point, it was decided this was a good approach as future researchers would not have to re-train models to fit the mould. Sparsity pruning is applied with the same dataset that the original training used, after which a subsection of the validation set was passed through the model. This part of the validation set is not used for validation anymore and is only used to stimulate the model and identify redundant channels and neurons within the NN.

PROPOSED PRUNING ALGORITHM V.1

Input: Pretrained neural network **T**, number of parameters *num_par*, list of prunable layers *pl*, training samples *x*, validation samples *v*, sparsity_threshold *st* , channel_threshold *ct*, number of optimization epochs *train_epochs* , accuracy_threshold *at* and pruning_threshold *pt*

Output: Pruned neural network **P**

while *True:*

    for *layer* **in T**:

        if *layer* **in** *pl*:

            model = apply_sparsity_pruning(*layer*, **T**, *st*)

    chan_to_remove = identify_redundant_channels(*v*, *model*, *ct)*

    P = apply_structured_pruning(*model*, *chan_to_remove)*

    P.train(*x*, *train_epochs*)

    if P.accuracy < *at* **or P**.*num_par* / **T**.*num_par* < 1 - *pt*:

        P = **T**

        break

    else

        T = **P**

5.4 Pruning Methodology Justification

5.4.1 Sparsity Pruning

Sparsity pruning sets weight values within a layer to zero according to a specified sparsity threshold. For the pretrained UCLF network from Figure 34b, it is applied on both residual and non-residual blocks in all three stages. To retain network performance sparsity was only applied to specific layers within the model. Additionally, the first and last layer of the network are not pruned. The total number of prunable convolutional channels and dense units is presented in Table 5. The values listed in Table 5 indicate the initial numbers of filters and dense units in a UCLF network, these numbers will reduce as pruning progresses.

5.4.2 Insignificant Channel Identification

Sparsity pruning does not guarantee that all weights associated with a channel will be zero. When applied to a certain layer, magnitude-based weight pruning sparsifies the entire layer across channels, rather than within each channel individually. In convolutional layers, filters are considered as channels. In dense layers, neurons are also considered as channels. To identify which channels in a layer can be removed from the UCLF network, a data-driven approach is adopted. The validation set is used as an input to the network. Neuron activation and filter activation maps are stored at each prunable layer. The stored values are then averaged over the smaller validation sub-set. If the average channel value is below the channel threshold, then that channel is marked for removal.

5.4.3 Structured Pruning

Structured pruning removes channels that were identified as insignificant for a specific UCLF network. For channels within convolutional and dense layers, this means removing all associated weight and bias information. An example of a pruned residual block from a UCLF network is presented in Figure 34b. As each stage of the network consists of stacked generalised blocks from Figure 34a. the dimensionality at the input and the output must be retained. In this example, the input is set to 64 channels. The first convolutional layer, marked in blue, and the first dense layer, marked in green, can be pruned without restrictions. The second convolutional layer and the second dense layer need to have an equal number of channels, as they are multiplied together. Therefore, when a channel is removed from the second convolution it must also be removed from the second dense layer. The pruned channels are then added to the corresponding channels from the input to the block. Finally, these are concatenated to the rest of the input channels and constitute the output of the residual block.

From this a pruned model is obtained by repeating this process and removing said redundant components at each iteration. These components are identified by a threshold value described in the previous section and is based around the filters and neurons capability to influence the output of the model, for this work this value was 0.0001. After the redundant channels have been removed the model is retrained for a fraction of the original training time so the model can adjust to the new structure.

Models are complex and not all the connections between layers can be accounted for easily, because of this some layers cannot be pruned, the table below breaks down the prunability of a UCLF.

Table 5: The number of prunable channels in each layer of the UCLF model. Blank entries mean that because of the structure of the network there are no prunable layers at these points.

Stage	Block type	No. of blocks	Prunable channels per block			
			2DConv 1	2DConv 2	Dense 1	Dense 2
1	Non-res.	2	48	--	8	--
	Residual	1	48	32	8	32
2	Residual	5	96	64	16	64
3	Non-res.	2	48	--	8	--
	Residual	1	96	64	16	64

5.5 Training Methodology

5.5.1 Introduction to deblocking neural networks

Replication of previous SOTA [13], will be achieved using TensorFlow [49]. Because VTM is open source to its contributors, a lot of data that is not normally afforded to other approaches is available. In this case the network developed leverages 3 main datapoints, firstly, it processes the Y, U and V components of video both separately and in combination. Additionally, this approach uses the CU block structure to guide the deblocking process. Finally, the quantisation parameter (QP) is used to indicate to the ML algorithm the degree to which the video has been compressed. The reason why this is a clever approach is that the block structure must be calculated by the decoder/encoder whether the AI is using it or not. This also highlights that an approach like this would not be possible unless performed inside or with the aid of the decoder and encoder software, due to the block structure being required. However, it's possible that development of another ML model could be used to generate the block structure too, meaning the process could be completely external from the current software.

5.5.2 Dataset

The dataset used for training was DIV2K [38], this dataset was chosen as it contains a good mix of landscapes, animals, humans, and mechanical objects, additionally, all images are of a high quality. The DIV2K dataset is made up of 900 2K images, 800 were used for training 100 were used for validation.

Each image is converted into 4 versions of itself at 4 different QP values, this was done to allow the AI to train on a range of different quality images. So, in total the actual training dataset is 3200 images, and there are 400 test images. Each image is then split into 48x48 pixel patches, this is done to make sure the data can fit onto the GPU for training. Special effort was made here to make sure the training set contained the patches at the edges of the images. This was essential because at the edges of encoded images there is a lot of complicated block structures that do not exist in the rest of the image. This is a side effect of the image not having a CU size that is wholly divisible by the length or height of the input image, for this reason extra patches are added to the dataset that contain only these overlapping regions seen in red in Figure 21.

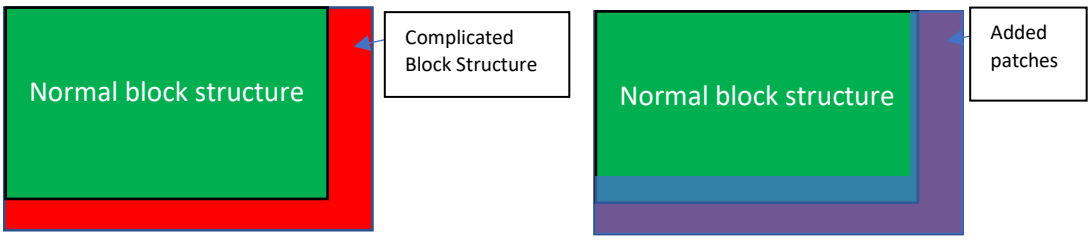


Figure 29: Visualisation of the overlapping sections added to the dataset to ensure the neural network trains on complicated block structures.

This will inevitably cause some overlapping in the dataset, but it is important to include these regions as otherwise when having to deal with such block structures once deployed, the network would not know what to do.

The process of obtaining the corresponding YUV files CU maps and QP values was done by extracting them directly from the VTM software and storing them as Numpy arrays on disk [50]. This was done so the VTM software did not have to be run during training as this would slow down training considerably. Figure 22 was repeated for each QP of 22,27,32 and 37.

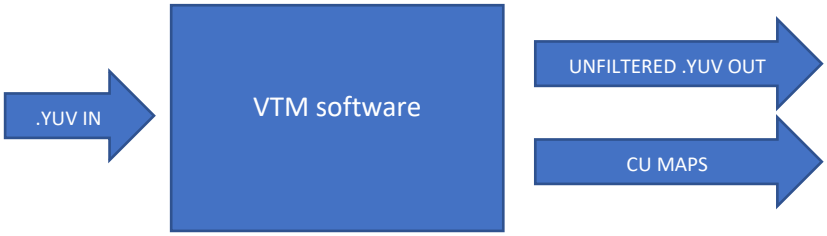


Figure 30: Visualisation of the VTM software handles the inputs and output required for the neural network.

It was found that by using the same error calculations as previous works [13], (MAE) that an error would occur causing training to fail suddenly and inexplicably, Figure 23.

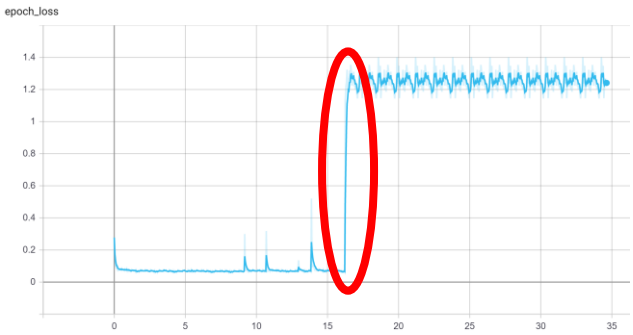


Figure 31: Issues with exploding loss whilst training

This was tested and repeated multiple times. Eventually it was discovered that there were certain images in the dataset that were giving results of infinite PSNR readings, for U and V, when compared to the ground truth. These turned out to be black and white images which makes perfect sense as they have no colour information, meaning the network learnt to predict a value of 0 causing the loss function to tend towards 0. When this was repeated over many batches it would cause the gradients in the models to explode and the model would then not recover back to a normal state, this is well documented in machine learning tasks [140]

This posed an issue as it seems that the training failure was caused by the loss measurement tending towards zero, however this is hard to address directly as it is a core component of this metric. The loss causing the gradients to explode, and by knock on effect the weights and biases too, needed to be addressed. To avoid this issue images 14,230,769 and 800 were replaced with 801-4 from the test set, these images were also removed from the test set.

This does set a precedent for considering what will happen to your network when a perfect solution is found. In this case the perfect solution that was found and caused the network to irreversibly break was to just ignore the U and V components, this highlights the need to be more cognisant when picking the loss function for your machine learning model.

5.5.3 Neural Network Structure

The same structure used in previous works [13], has been used for the experiments so far. A visual representation in Figure 24 shows how the network originally starts as separated parts (Y/U/V) before it is combined in the middle and then split apart again at the end of the network to produce a YUV output.

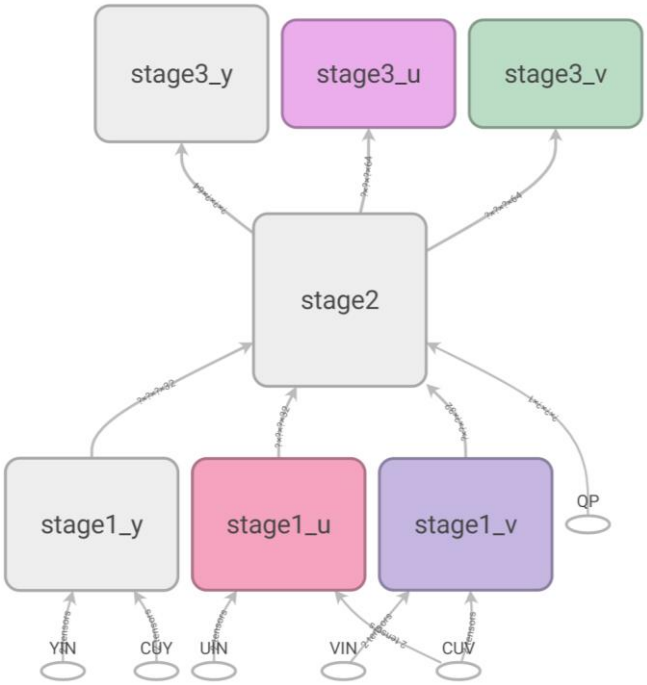


Figure 32: Network structure implemented to replicate SOTA [13], shown using TensorFlow.

5.5.4 Training

When training the NN takes an unfiltered output of the VTM software, in a YUV format, the block structure, and the QP value, it uses all this information to generate a new output YUV file. This YUV file is then compared to a ground truth image that was used to generate the unfiltered output.

A Batch size of 128 is used, 48x48 patches were also used, which reflects previous approaches [13]. To avoid small batch numbers when near the end of the dataset the last few overflow patches are thrown away. This is done because when a NN deals with small batches, the gradients heavily favour the data in that small batch, making the network optimise for an edge case that does not exist [141]. This process does result in the loss of some training data but the effects of having full batches far outweigh the few extra patches that would be gained otherwise.

The ADAM optimiser [142] was used with TensorFlow's [49] default settings, the Network was trained for 100 epochs and compared to the performance of the SOTA [13] Figure 25.

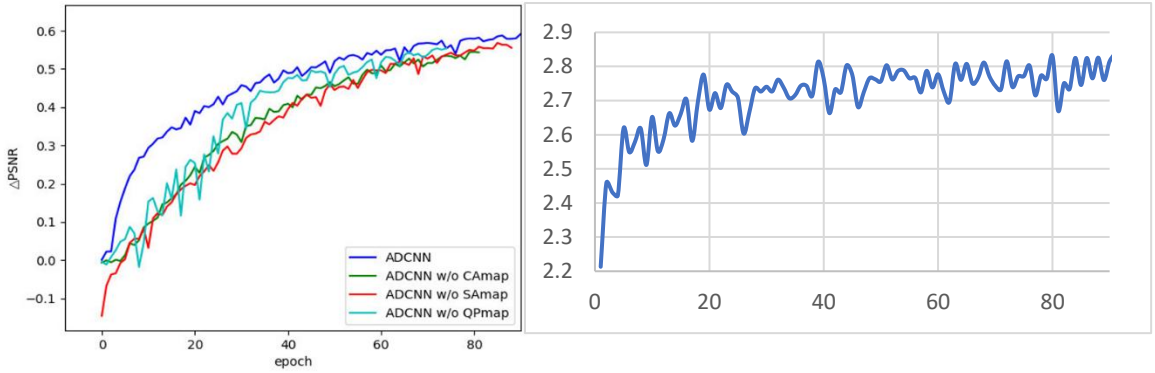


Figure 33: Left shows SOTA results [13], and the right shows my results from training. It is important to note the y axis, ours ranges from 2.2-3 SOTA from -0.1 to 0.6. This is because they are reporting the average PSNR per image whilst my results are cumulative over the test set.

Whilst training this network some special considerations had to be made, when uncompressed on disk the dataset takes up a total of 300-400GB. Some of this data is redundant for ease of loading the files, but an estimated 250GB of RAM at least would be required to hold the full dataset in memory. In addition, TensorFlow seems to duplicate the data in memory, (this is probably due to TensorFlow changing the data's format to a tensor), hence the RAM requirements for this experiment grow quickly.

To circumvent this issue the data is loaded and trained on in sections, 100 images at a time. This cuts the RAM requirements to 50-70GB Max and makes the training process possible for most well-equipped systems. For speed 2 RTX 2080 Ti's were used for training.

5.6 Interpreting a Video Coding Algorithm

To interpret computer vision neural networks tools had to be developed focusing on models which had available code and datasets for testing [13]. This network was developed to reduce the compression artefacts present when compressing video with the H265 codec.

5.6.1 Visualizing the NN filter

Figure 26 shows how the NN used by the SOTA [13], differs from the normal and coded non-ML approach used in H265. It can be seen in the output produced by the NN that there are large black boxes, these are related to a check that the ML software does. This check makes sure that the image produced by the ML image is of better quality than the one that is produced normally. If it is not, then the ML image is not applied, nothing is changed and this manifests as the black box.

The image shows the difference in pixel intensity of the Y components when the NN filters the image and when normal VTM filters the image. Blue indicates a gain in intensity and red indicates a loss.

The histogram shows the changes made over the whole image, overall the standard approach mostly changes values by -1, 0 or 1 magnitude. The NN changes less values in this range and instead prefers to change values more than 4 and less than -4. This is reflected in the output images as the NN output seems more "heavy handed", having thicker brighter lines indicating it is applying different filtering approaches than that of standard VTM.

There are many other aspects of these images that can be analysed but the problem is this kind of analysis is rather subjective. Further mathematical analysis must be done on these images to truly get an objective view of the difference between these approaches.

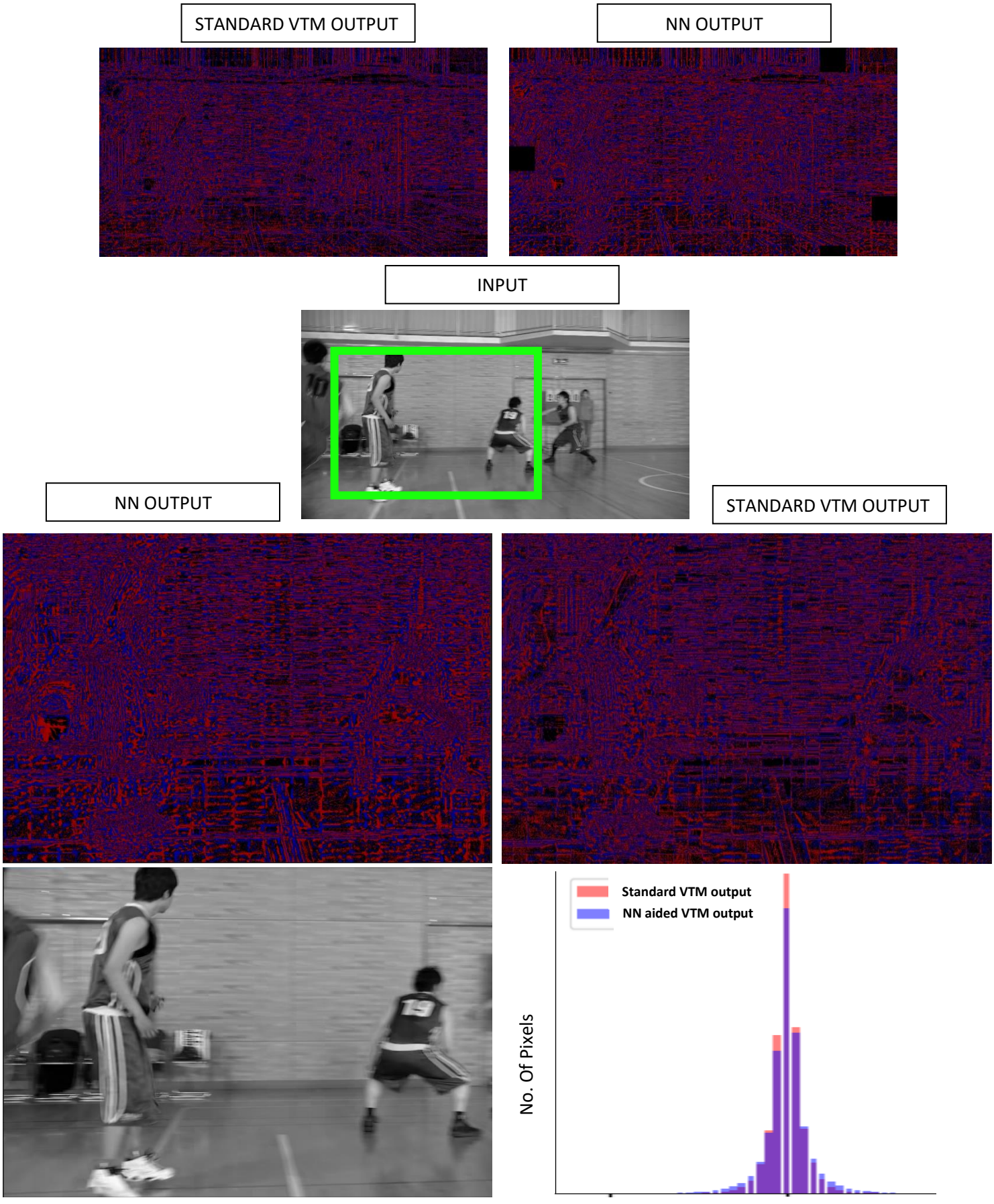
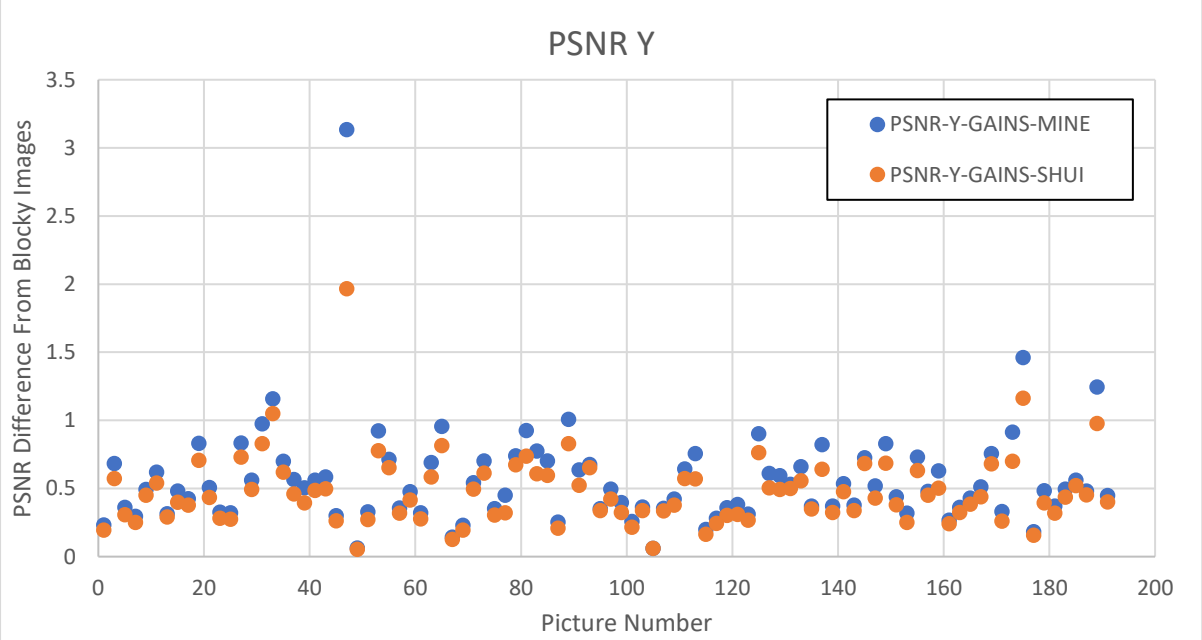
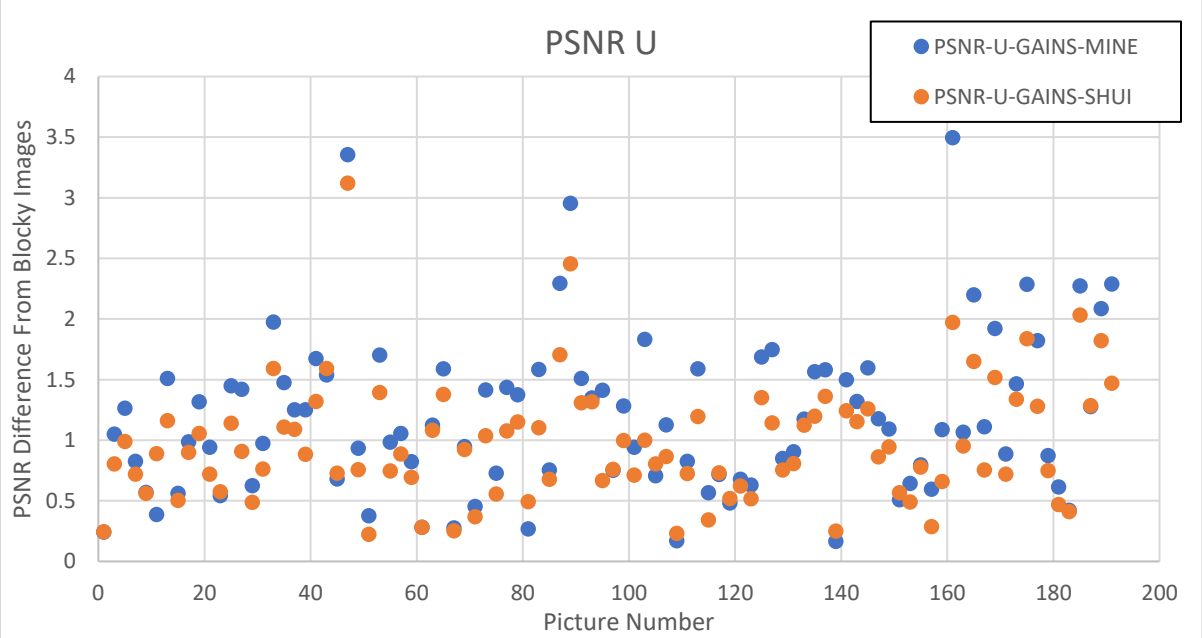


Figure 34: Detailed visualisations of the output of the edited version of VTM [13] compared to vanilla version of the VTM software. (Full animation shown in appendix B.4)

5.7 Replicated Results

To test the SOTA against the network developed in this work it was important to develop a test that was a fair comparison of the two. As discussed in the literature review the software has ways to detect particularly badly performing blocks that the neural network has filtered and turn them off, so they do not negatively impact the results. Because of this the network was removed from inside the software completely and tested both networks in TensorFlow 2. They had Identical data inserted into them and this data was the DIV2K Test set, there is no benefit to either network here as both used this network for training. The results can be seen below.





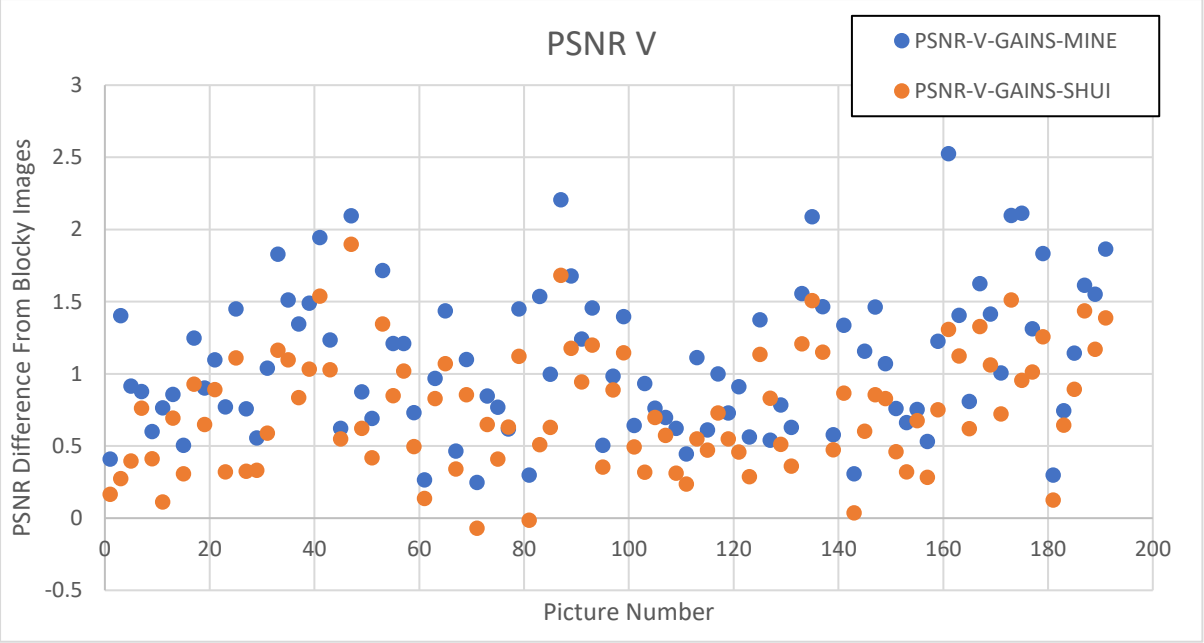


Figure 35: The graphs above show how the network trains consistently and performs the same as if not better than the SOTA [13]. Each graph deals with one component of the image be that Y, U or V. Each point indicates an image in the test set.

Each image of a test set of 97 images has been fed into both networks, the PSNR of the output is then calculated and the amount that the PSNR has changed from the input image has been plotted. The results from this work's network are in blue and results from the SOTA [13], are in orange.

This result shows that the network developed for this work has been trained to a similar level to the SOTA and is performing as expected. Additionally, when using SOTA models [13], the PSNR-V results in some neural network filtered images have worse PSNR values than the blocky image that was fed into the network (images 71 and 81). This is not true for the network we developed, and our results (blue dots) never go below 0 for Y, U and V results.

Table 6: This shows the average PSNR gains across the whole DIV2k validation dataset when compared against the SOTA [13].

Network	PSNR Y Gain	PSNR U Gain	PSNR V Gain
SOTA [13]	0.475	0.967	0.747
NN Trained for this research	0.564	1.186	1.081

5.7.1 Visual Examples

Below shows how the network has smoothed out the blocky structures present in the input images.

Figure 36: Visual examples of deblocking and smoothing of frames when the machine learning model has been applied.

These results show that blocky artefacts are being removed from the input image. However, on close inspection of the eyebrow, some of the filters used to create the blocks in the previous step to this are not being successfully removed (circled in red). This leaves weird texturing on the image.

These results indicate that the SOTA has been successfully replicated and the next stages will focus on interpreting and pruning this network.

5.8 Pruning Results

5.8.1 Metric justification

Before viewing the results, a metric had to be decided on, this was so models could be fairly compared that achieved different accuracy metrics and varied in compute requirements. For this reason, two metrics were combined.

5.8.2 Timing Test

Initially models were loaded and tested on the entire test set, the time taken for the test to complete was recorded for comparison later. Because consistency was important in this test, the test set was patched into 48x48 sections, this was done to maximize the number of samples presented to the network. A large amount of test patches ensured a true representation of the time taken for a specific iteration of the network to produce an output. The test set was stored in memory and no reading or writing to disc occurred, this was done so that timing metrics were a representation of the GPU compute time only. 2400 was then divided by time in seconds taken to compute these tests, this was because the test set contained 2400 patched images. This gave us a number that represented the number of images per second that that specific network could process.

5.8.3 PSNR Test

The second test was also conducted on the same test set, but this time the images being tested were not patched. Each input image was blocky, and the output was saved to disk, then PSNR was calculated for the blocky image with the ground truth unprocessed image as reference. The saved image was later reloaded from disk and an identical PSNR measurement was made. The values of the blocky image and the processed image's PSNR were then subtracted from each other and hence a PSNR gain was obtained. The tests were split in this way because this test often produced wildly different timing metrics for the same model. This was attributed to time taken saving values to disk, and the variance in image dimensions passed to the model, causing it to have to re-deploy CUDA code to the GPU.

This was not compared against the standard output of VTM at this stage because that was not the comparison we wanted to make in this part of the research. This metric was developed to track the model's ability to be pruned and retain its abilities of deblocking.

5.8.4 Combined Metric (PSNR/Sec)

The PSNR gains over the blocky image were then multiplied with the value obtained by the timing tests, this gave a final metric of PSNR/sec.

$$Processing\ Rate = \frac{2400}{timing\ test\ duration\ (s)}$$

$$\frac{PSNR}{Sec} = Processing\ Rate\ \times PSNR$$

Equation 9: This shows how the PSNR/Sec metric was derived, as both PSNR and speed of inference is important for this kind of ML model.

5.8.5 PSNR Results

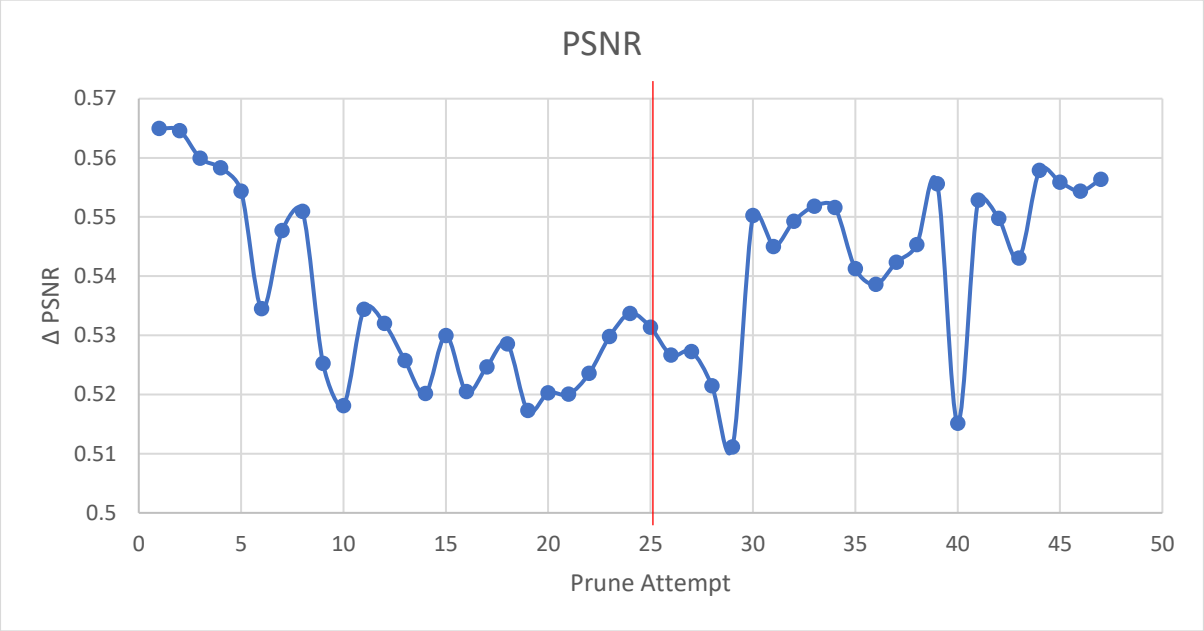


Figure 37: PSNR performance of the network was affected over each prune attempt. The red line indicates a change in the learning rate from 1e-4 to 5e-4. PSNR does vary by at least 0.06 over the whole model set it does recover towards the end of the pruning attempts.

5.8.6 Timing Results

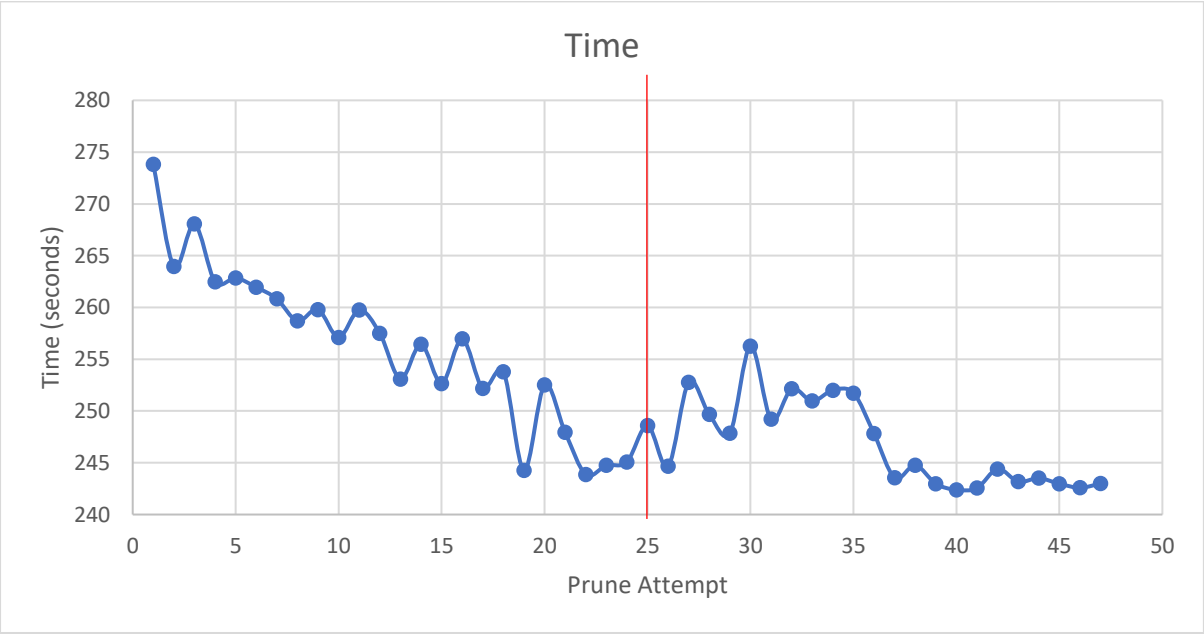


Figure 38: The network slowly reduced the amount of time it took to process images. The red line indicates a change in the learning rate from 1e-4 to 5e-4. This timing reduction is dramatic at first but becomes less and less impressive as the network becomes harder to prune.

5.8.7 PSNR/Sec

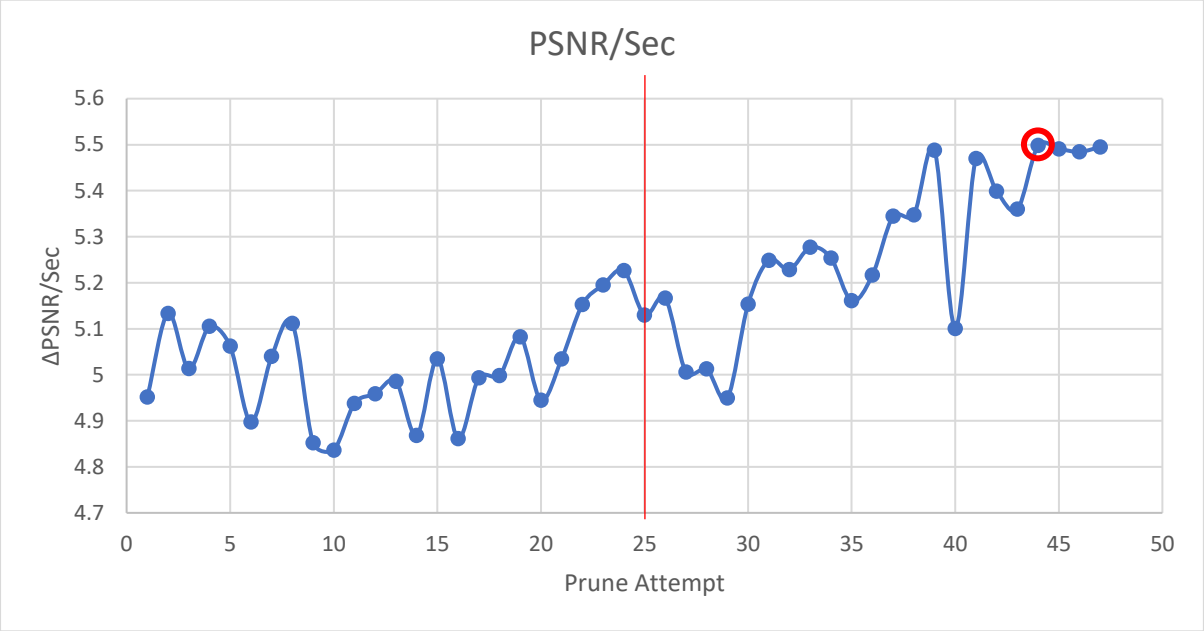


Figure 39: The metric we developed to compare the speed of processing images against the performance of the network. The red line indicates a change in the learning rate from 1e-4 to 5e-4. The upward trend indicates that the network is performing its task quicker with comparable PSNR values to the unpruned network. The red circle indicates the model that scored highest with this metric.

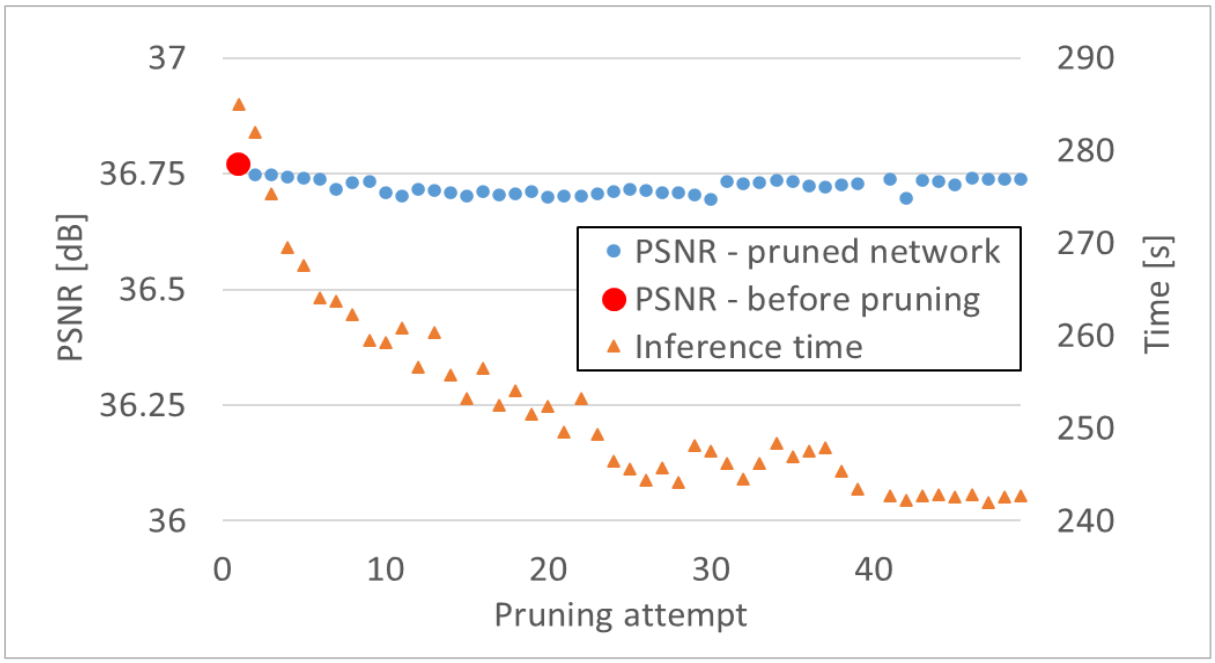


Figure 40: Pruning of UCLF Y component network. Each pruning attempt represents one pruning loop of Algorithm 1. Results are displayed as average PSNR and total inference time for the validation dataset.

5.8.8 Network Performance Comparison

Whilst parameters of a model are not really a valid comparison of network performance when it comes to NNs it is still of interest. The initial Y model had 879,681 trainable parameters; the pruned network contained 667,265 trainable parameters. This is a reduction in parameters of 25%.

The network unpruned would take 285 seconds to process the test set, we choose to compare this against prune attempt 44 as our fully pruned model. This is because at a score of 5.49 PSNR/Sec, this model performs best on our developed metric, circled in Figure 39. Prune attempt 44 processed the same test set in 243 seconds. This is a reduction of 42 seconds or is a 15% reduction in processing time.

Prune attempt 44 performed with a PSNR performance of 0.557, the base model before pruning performed at 0.57. Therefore, the pruned model performed at a deficit of 0.013 PSNR over the whole test set, we view this as an acceptable impact for the performance gains.

5.8.9 Ablation Study

When reducing the number of residual blocks manually so that the network exhibited a similar number of parameters to that automatically obtained through our pruning method. The network could never reach the same level of PSNR performance as the original network.

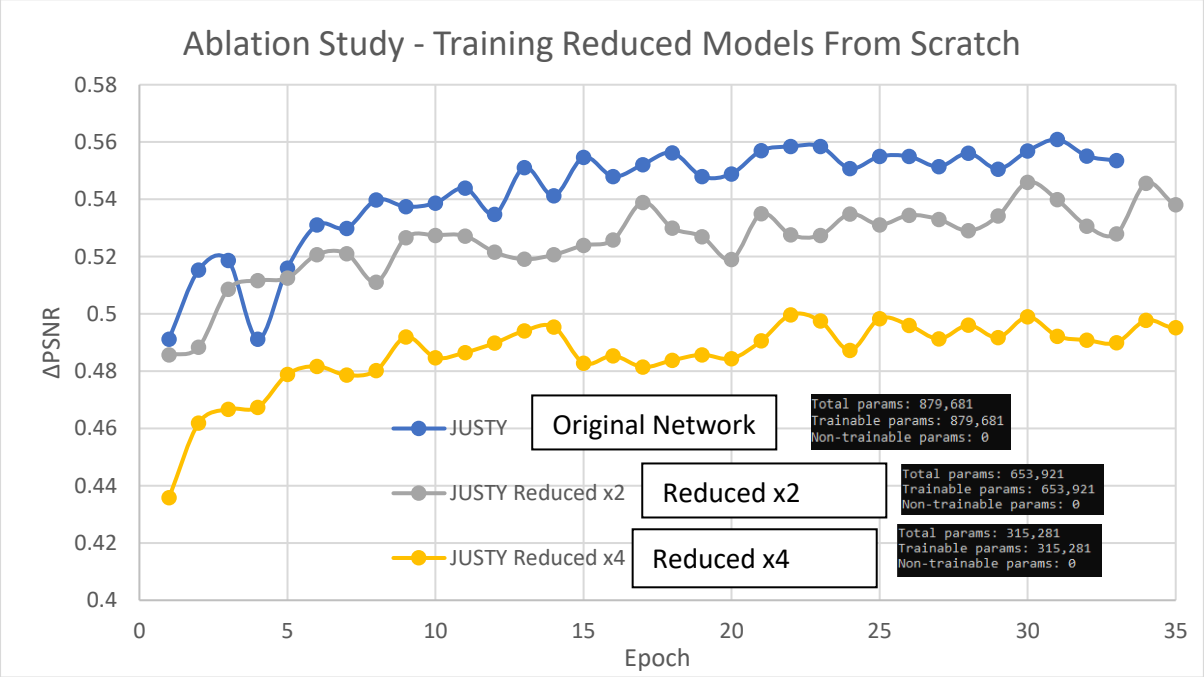


Figure 41: Ablation study comparing a manually reduced network. These results show that this kind of manual pruning never achieves the same PSNR performance as the original network.

The grey line in Figure 41 shows the performance of a network that was manually pruned before training to reflect the same number of parameters as the automatically trimmed network. These results indicate that simply reducing the number of parameters of a neural network to reduce the computational complexity of the model is not a viable approach, and the method we developed is more applicable.

5.8.10 Network Architecture Over Time

From prune attempt to prune attempt the nodes and channels that were removed were recorded, showing how pruning progresses over time. Interestingly pruning seems to be focused on removing filters in layers that contained more filters. Additionally, layers that were closer to the end of the network were pruned for more. This could be due to the final layers changing the data minimally, as they are the last components before output, they may only fine tune the image and hence be easily removed. This same logic also applies to layers with many filters. This visualisation can be seen in appendix A.

5.8.11 Comparison to VTM baseline and original research

So far, the comparisons of pruning have been against an altered version of the SOTA model [13]. This was done to compare two "like networks" as fairly as possible, however it is also important to compare the fully pruned network to the original network, these results can be seen below.

Table 7: Improvement of inference time and U/V PSNR when tested on VVC common test conditions. The columns on the left show the models before pruning and the columns on the right show the pruned models.

Class	UCLF before pruning						UCLF after pruning						Time Reduction (50 frames)
	BD-rate [%]			BD-PSNR [dB]			BD-rate [%]			BD-PSNR [dB]			
	Y	U	V	Y	U	V	Y	U	V	Y	U	V	
B	-3.53	-3.82	-2.58	0.13	0.07	0.06	-3.07	-4.75	-3.90	0.12	0.09	0.09	31%
C	-4.68	-5.16	-2.37	0.29	0.19	0.21	-4.52	-5.91	-5.30	0.28	0.22	0.21	36%
D	-6.57	-7.58	-8.68	0.47	0.30	0.35	-6.43	-7.93	-9.03	0.46	0.32	0.37	44%
E	-5.13	-2.35	-2.12	0.25	0.08	0.06	-5.09	-3.89	-3.49	0.25	0.14	0.11	59%
Average	-4.98	-4.73	-3.94	0.29	0.16	0.17	-4.78	-5.62	-5.43	0.28	0.19	0.20	42%
#Par	Y: 879,681; U: 879,681; V: 879,681						Y: 667,265; U: 293,811; V: 116,972						
Time [s]	Y: 285; U: 120; V: 120						Y: 243; U: 84; V: 76						

In VTM encoder performance is compared with a suite of standard test conditions, above is the summary. The percentages show the BD gains of the two models when compared against the VTM anchor, which is a standard version of the VTM encoder. As it can been seen, the pruned model performs better if not very similarly to the original model. However, the big difference is in the time components shown below.

Table 7 shows the number of parameters has been reduced significantly for every UCLF network, with the U and V networks being pruned the most. The U and V networks both increase in their performance when compared to their unpruned models, we believe this is due to the retraining steps in the pruning loop. The Y network decreases in performance slightly and reduces to a much lower degree than the other networks. We believe this is because the Y components of video contain 4 times the information of U and V and therefore this network is more highly saturated with information and subsequently harder to prune.

Table 8: Speed up in inference time due to pruning on various classes in the common test conditions.

Class	Time taken pruned model to render 50 frames (s)	Time taken original model to render 50 frames (s)	speedup
Class B	27.85	40.40	31%
Class C	13.33	20.94	36%
Class E	6.72	11.98	44%
Class D	3.01	7.27	59%

5.9 CONCLUSION

An initial approach to reduce complexity for learned in-loop filters was developed, by combining sparsity pruning and structured pruning, redundant parts of a neural network can be identified and removed without impacting its performance.

Results show that this method can reduce the number of parameters by as much as 87 % and improve inference by up to 59 %.

The presented method has the potential to reduce the size of neural networks used in other pixel augmentation tasks making all these types of networks more applicable for practical applications.

6 Pruning an Image Translation GAN Neural Network

6.1 Preface

After the success of applying pruning to a deblocking model it was important to test the capability of the method on another model and multiple datasets.

To achieve this the Pix2Pix model [51] was chosen, this model was chosen because it is a GAN based model which is very different in operation to the residual model tested earlier.

Additionally, Pix2Pix is an image translation task, and many datasets can be processed by the same network. In total there are 6 datasets that are readily available, 3 of which were selected for testing.

6.2 Pruning Methodology

Using a GAN model adds an extra level of complexity when applying the pruning algorithm, this is because a GAN network is made-up of a discriminator and a generator [148]. In GAN networks these work in tandem, however it makes no sense to prune the discriminator in these networks because the discriminator Is not used when the model is deployed.

When deployed GAN models are split and the generator is the only part of the model used, because of this pruning was only applied to the generator portion of the networks to save time. Additionally, due to restrictions of the GAN tools in the TensorFlow framework, the generator had to be pruned without the aid of the discriminator. However, this was only true for the sparsity pruning step. After pruning away entire channels, the generator and discriminator could be recombined and trained in tandem again.

Additionally, the pruning method was changed slightly, the previous method relied on identifying “dead” neurons in the neural network. Although this was very effective it was slow, and it did not give complete control over the number of parameters removed from the original model. To address this the process was changed so that each pruning loop had to remove a minimum ratio of the overall filters within the generator model. For these tests this was set to 0.01, this is the decay rate of the pruning loop. Although this percentage may seem low, it’s important to consider that these filters will be removed for each iteration of the pruning loop, so, in total over 200 pruning loops, the model would reduce in size by over 80%.

$$F_F = F_I\left(1 - \frac{D_r}{1}\right)^x$$

$$F_F = Final\ Filter\ amount \quad F_I = Inital\ Filter\ amount \quad D_r = Decay\ rate\ x = Number\ of\ pruning\ loops$$

Equation 9: The following equation was derived from the formula for compound interest [58] to calculate the percentage of the model reduced at each pruning loop.

$$\lim_{x\to\infty} F_I(1 - D_r)^x = 0\ for\ F_I \in \mathbb{R} \wedge \log(1 - D_r) < 0$$

Equation 10: Exploring the limit of the equation above shows that as the number of pruning loops tends towards infinity, the number of filters left in a model tends towards 0 for any given pruning rate (as Dr has to be a positive real number).

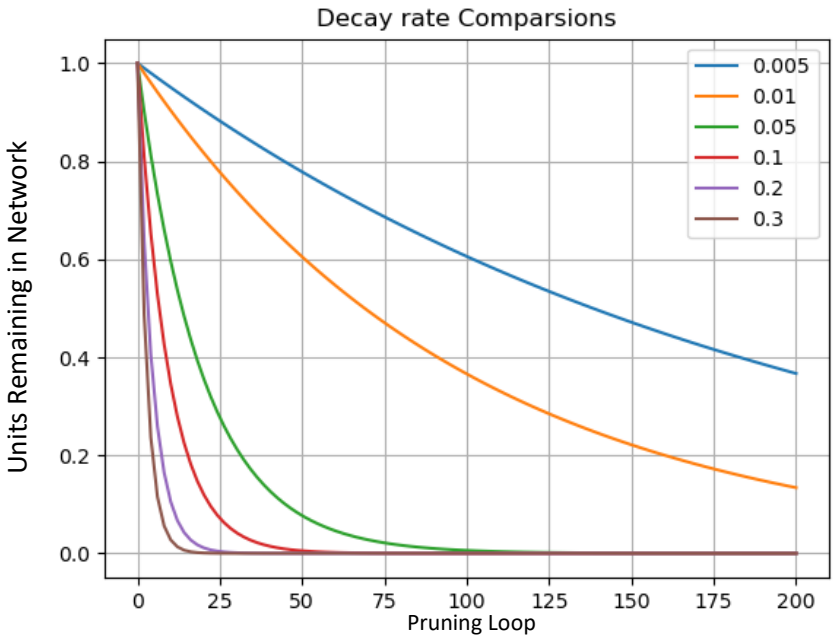


Figure 42: The effect of changing the decay rate when running the pruning loop 200 times.

These parameters give finer control over the pruning of the network, one can choose to have a very high prune percentage, and low pruning loop iterations or vice versa. Logically the higher the number of pruning loops, the greater the possibility that the GAN model will fail. By analysing equation 9 in its limit, we find that if prune iterations are infinite, an infinite number of filters will be pruned, shown in equation 10. This is impossible as no neural network has an infinite number of filters, and instead the result would be no neural network remaining. It is ensured this outcome is never reached by taking a snapshot of the model at every pruning iteration. By taking snapshots of the model at every iteration, we ensure that when the model does fail, we can return to a point before the failure occurred.

Another change was made that was crucial to make when identifying "dead" neurons. The data for the network in chapter 1 was scaled between 0 and 1, additionally the network in chapter 1 only contained ReLU activations. These two aspects meant that mathematically a value below 0 could not exist in the network (except for the last layer which was not pruned). However, the Pix2Pix data is scaled between -1 and 1, whilst ReLU is used in the decoder blocks of the network leaky ReLU is used in the encoder parts of the network.

The blue line in Figure 47 shows the ability for leaky ReLU to pass negative values at the output of a layer, because of this, the check for important neurons had to consider negative values to be just as impactful as positive values.

$$Old\ Neuron\ importance = \frac{\sum Act_{Val}}{N_{Val}} \qquad New\ Neuron\ importance = \frac{\sum |Act_{Val}|}{N_{Val}}$$

$$Act_{Val} = activation\ on\ a\ subset\ of\ the\ validation\ data \qquad N_{Val} = number\ of\ samples\ in\ the\ validation\ split$$

Equation 10: New method of identifying Neuron importance that considers negative and positive values.

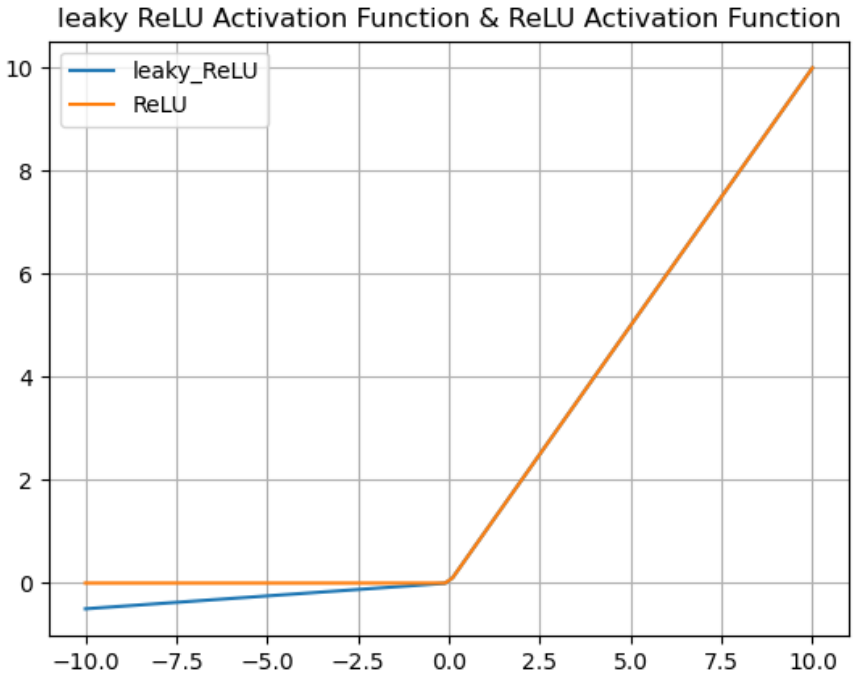


Figure 43: ReLU Vs leaky ReLU activation functions

Leveraging what was learned from pruning the model in the previous chapter, scores were calculated for each pruned model trained on the cityscape dataset from the pruning process. These metrics will be considered along with the reduction in FLOPs and Parameters for a given model. The metrics will be FCN, PSNR, SSIM and Inception scores where available. FCN segmentation score uses a model designed for sematic segmentation [59] trained on the cityscape's dataset, trained by the original authors of Pix2Pix [51], where the output of the segmentation task is directly compared to the ground truth.

The remaining datasets are difficult to mathematically quantify, as often with GAN models the output is subjective, and subjective testing is out of scope for this project as this time. The 'Maps' dataset is still difficult to perform analysis on due to it lacking a well-defined metric for the validity of a map image, for this analysis PSNR and SSIM was used. PSNR and SSIM were also used as metrics across other experiments to understand how they interact with pruning.

6.2.1 Pruning Algorithm

For this body of work, it was important that results were replicable and simple to obtain, because the source code for Pix2Pix was readily available and well documented it was decided that all models would be trained from scratch, datasets would be downloaded and processed automatically, and pruning would be performed on these models automatically.

This work has been released in an open-source repository [60], this should provide anyone wishing to replicate the results of this body of work a solid starting point.

The functions work the same way as described in the previous section with the changes to the identification of redundant neurons described in 6.3.

PROPOSED PRUNING ALGORITHM V.2

Input: Pretrained neural network **T**, list of prunable layers *pl*, training samples *x*, validation samples *v*, sparsity_threshold, number of optimization epochs *opti_epochs*, dataset to process *python_args*, dataset path location *path_for_datasets* , the amount of initial epochs *initial_epochs*, the number of times to run the pruning loop *prune_loops*

Output: Pruned neural networks $\mathbf{P_n}$

```
dataset_selection = select_dataset(python_args)
download_dataset(dataset_selection, path_for_datasets)
process_dataset(dataset_selection, path_for_datasets)
x,v = load_dataset(dataset_selection, path_for_datasets)
UP = initalise_model()
UP. train(x, initial_epochs)
for n in prune_loops:
    P,D= split_GAN_model (UP)
    for layer in P:
        if layer in pl:
            model = apply_sparsity_pruning(layer, T, st)
    chan_to_remove = identify_redundant_channels(v, model)
    P = apply_structured_pruning(model, chan_to_remove)
    UP = recombine_GAN_model(P,D)
    UP.train(x, opti_epochs)
    UP.save_model()
```

Additionally, two new functions were added, split GAN, and recombine GAN, these are true to their name sakes, one separates the GAN into its two constituent parts, the generator and discriminator, the other recombines them.

6.3 Training Methodology

Introduction to the Pix2Pix models

The Pix2Pix model is a GAN model that performs image translation tasks, this means that the neural network takes an image of a given object and translates it to another version of itself. The Pix2Pix framework was developed to deal with multiple versions of these tasks without any change to the underlying architecture of the model.

These tests aim to also provide information on the complexity of a task, related to the ability of a network to be pruned. The intuition would be that more complex tasks (datasets) will create models that contain more information and therefore will be harder to prune.

6.3.1 Datasets

To test the capability of the pruning algorithm to adapt to multiple datasets 3 were selected from the 6 available, the 6 datasets available for Pix2Pix are.

- Cityscapes [52] ✓
- Edges2Handbags [53] ✗
- Edges2Shoes [54] ✗
- Facades [55] ✓
- Maps [56] ✓
- Night2Day [57] ✗

Edges to handbags Edges to shoes and Night to Day was excluded due to training inconsistencies and the inability to mathematically quantify the results.

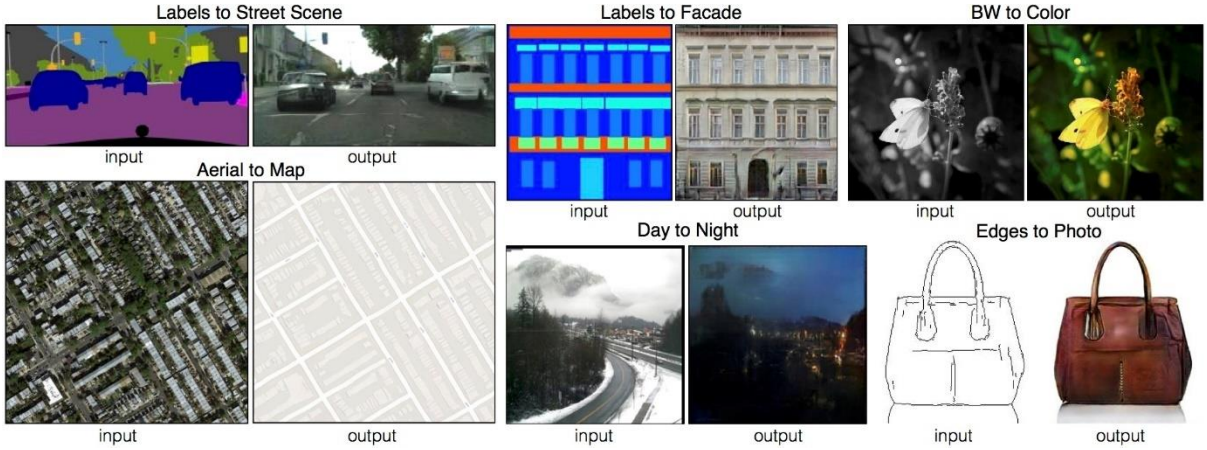


Figure 44: Pix2Pix dataset example pairs [56]

Cityscapes is a pair of city images taken from a car dashcam, and a manually created segmentation map. The segmentation map identifies cars, vans, trees, and other objects commonly located on or around roads. The Facades dataset is a picture pair, one being the front of buildings in Paris, and the other a manually created segmentation map. This map Identifies doors windows balconies and other architectural features. Finally, the maps dataset is a satellite image taken from google maps, and the google map navigational map pair.

The genius of the original Pix2Pix paper is that because the model architecture is so adaptable the datasets can be input into the network in any order. It's possible, by reversing the datasets, to generate a satellite image from a map image, or to generate a map image from a satellite image.

Figure 45: Each dataset can be setup in 2 ways, here is an example of maps generating satellite imagery or satellite imagery generating maps.

Because of this we can use each dataset twice, this gives us a total of 6 experiments to run, made up of each dataset with an A/B, B/A configuration, demonstrated in Figure 43.

This will allow for intricate analysis of the pruning process, as discussed in the subsequent section Pix2Pix is a GAN, and the generative part of the network is a U-Net [144]. This structure has a decoder side and an encoder side. When selecting filters to prune, the pruning algorithm only identifies “importance”, meaning the model could be pruned at any point. Hopefully by reversing the datasets, and hence the tasks, we can identify the most essential layers for this kind of model.

After each image pair was separated the RGB data was converted from values of 0 to 255 to -1 and 1, then each image was then stored as an A/B tag. These were stored separately to allow for swapping of the dataset pairs in the future.

6.3.2 Neural Network Structure

A GAN has a generator and a discriminator, in Pix2Pix the generator is a U-NET with skip connections. The discriminator is a Convolutional PatchGAN Classifier [51], the discriminator, in this case, performs analysis on patches and not the entire output.

GANs work by periodically presenting real and generated images to the discriminator, the discriminator tries to identify if the image is real or generated. The result of the discriminator can be combined with other metrics, like SSIM and PSNR, to create a loss fucntion to train the generator.

Pix2Pix is a GAN network, specifically a conditional GAN [145], conditional GANs work in much the same way as conventional GANs, however, the loss is learned instead of hard programmed. They also have the addition of random noise introduced to the model, in the form of dropout layers. As such the target and the output can be better compared and errors corrected for.

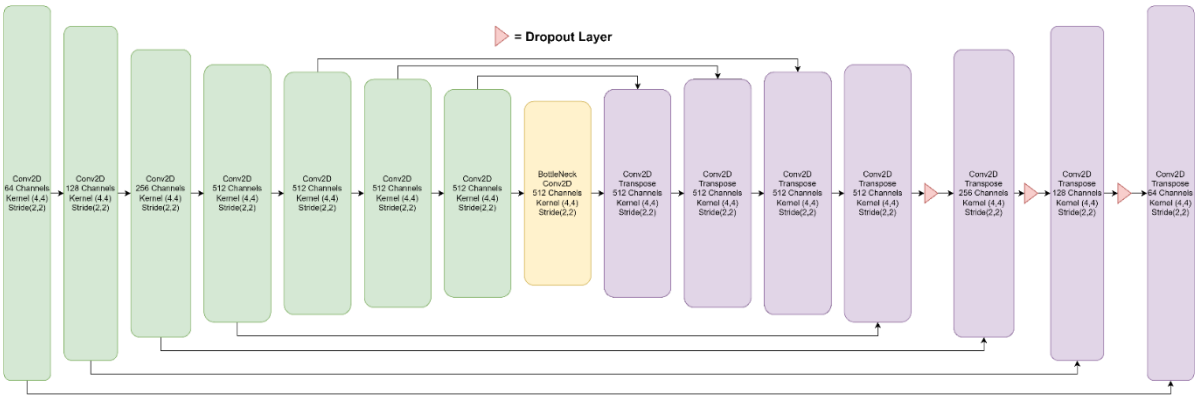


Figure 46: Architecture of the U-Net used in Pix2Pix [51]

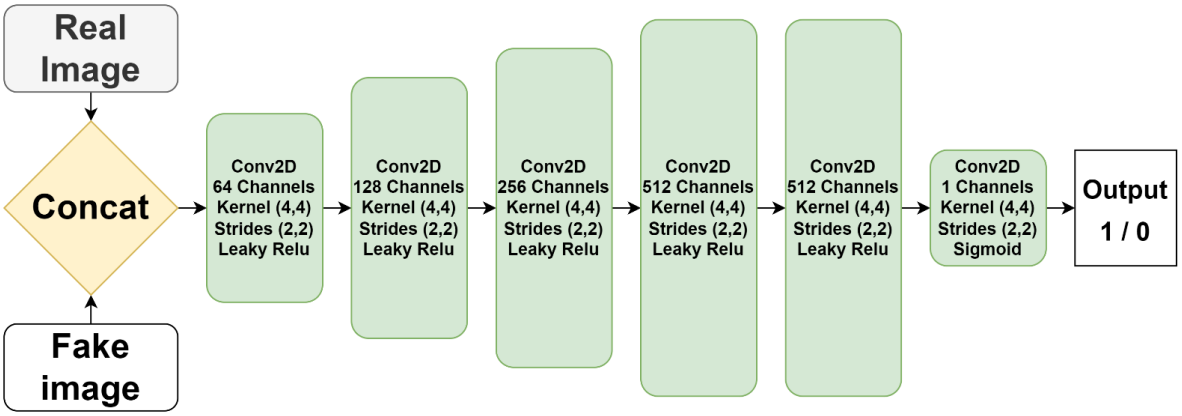


Figure 47: Architecture of the PatchGAN classifier [51]

The discriminator is not pruned when using our pruning method, figure 45 shows the structure of the discriminator.

6.3.3 Training

Each GAN was trained using the following hyper parameters.

- Initial Training Epochs = 25
- Initial Pruning Epochs = 25
- Retraining Non-Pruning Epochs = 10
- Retraining Pruning Epochs = 10
- Optimiser = Adam
- Learning rate = $2x10^{-4}$
- Beta = 0.5
- Batch Size = 32
- Pruning Loops = 100
- Decay Rate = 0.005

When performing sparsity pruning, an initial sparsity target of 50% was chosen (this was the percentage of zeros the network had to achieve by the end of the first epoch). Sparsity pruning was set to have a final sparsity of 80%, which had to be achieved by epoch 25. Polynomial decay was used to control this over the pruning epochs.

There were some considerable hurdles found when pruning the Pix2Pix model, mainly because of its size. The previous models tested were 800,000 parameters before pruning, the Pix2Pix model starts with 61,397,572 parameters. Because of this the method developed for deploying the model to memory had to be adjusted, previously all models were loaded into VRAM, whilst wasteful this never caused issues as the GPU's available had ample space to store the redundant models.

When dealing with models of this size it is imperative to load and unload all model information from the GPU, unfortunately the only way to do this currently is to either use separately launched python scripts or to use the python multiprocessing library.

By using the pseudo code below, it was possible to contain models within their own environment, when their training and pruning loop had finished, the model could be completely dropped from the GPU's VRAM. Each model was pruned using the algorithm in section 6.3.1 for 50 epochs total.

parent_conn, child_conn = multiprocessing.Pipe()

reader_process = multiprocessing.Process(target=Train_Model, args=(Training_Args))

reader_process.start()

vars_returned_from_training = parent_conn.recv()

reader_process.join()

This approach is not without issues, multiprocessing does not allow python to pass "pickle" objects [147]. These are large variable types in python used for dealing with complex datatypes, unfortunately both TensorFlow models and TensorFlow dataset objects are pickle objects. This means that within the training and pruning loops the model and datasets must be reinitialised every time the loop is run, luckily this adds little to no overhead when training the models.

6.4 Inception Score

Inception score had its inception in a paper analysing the output of a GAN network trained to replicate the MNIST, CIFAR and ImageNet datasets. The metric found inception score closely aligned with the subjective tests relating to the 'realistic' nature of the generated images [149].

This metric has been adopted by the machine learning community as a measure of how real a set of images look, the higher the inception score obtained, the more "real world" the output is considered. The metric considers both the variety of the images and the amount that each image looks like an object, this ensures both a varied range of outputs, and the ability of the GAN to form coherent images.

Inception score varies a lot depending on the image dataset used, each dataset contains different objects and variations in these objects. Therefore, it is important to obtain a baseline Inception score, this is used as a known score of "realism" for the GAN to aim for, for the given dataset. For our purposes the inception score of the network at prune loop 0 will be considered as the inception score to aim for.

6.5 FCN Score

FCN score is a metric designed to measure the accuracy of a segmentation map. This metric was selected as the original Pix2Pix used it as a metric to analyse their results, specifically of the cityscapes dataset. It cannot be used as a metric for the other 2 datasets, as it requires a deep learning model specific to the dataset. Only one was produced from the original paper, and this was for the cityscape's dataset.

This score is calculated by first training a separate fully connected network to perform the task of image segmentation on the original cityscape's dataset. Once trained the model can perform the task of producing a segmentation map given an image from a car windscreen.

Figure 48: Segmentation map generated by the FCN model.

This model can then be used on the generated output from the GAN to check if the same kind of segmentation map is produced when the original input is also presented to the FCN.

Figure 49: Segmentation map generated by the Pix2Pix model

This technique allows for many metrics to be produced, but for this work we will focus on mean pixel accuracy, mean class accuracy, and mean class IOU.

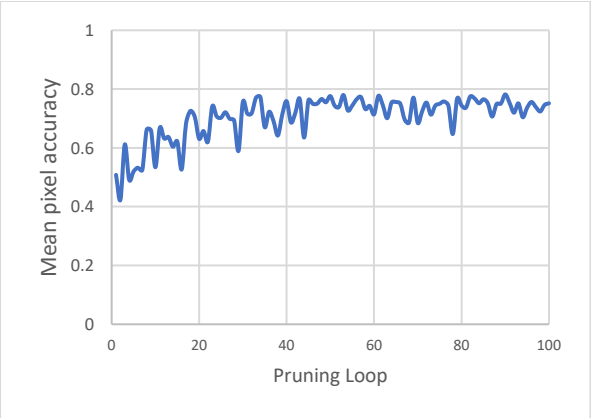
Mean pixel accuracy is the average amount that a pixel in the generated image was segmented as the correct class. Per class accuracy is the number of times the correct number of classes were identified in the generated image. And mean class IOU is a number that quantifies the degree of overlap between the segmentation masks.

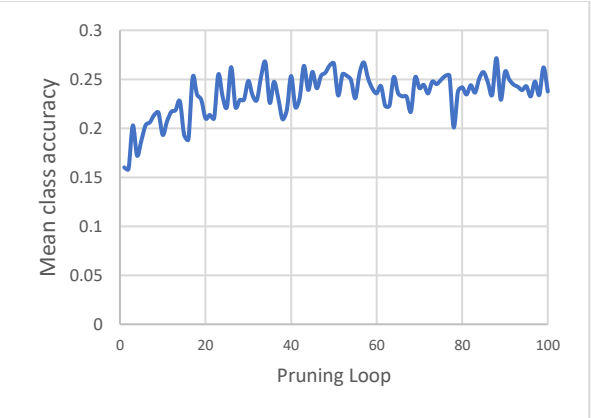
Whilst this metric can only be applied to one of the datasets, it was considered important to replicate the tests seen across academic publications [51], the model used for segmentation [51] and scoring method were obtained from the relevant GitHub repositories.

6.6 CITYSCAPES A -> B RESULTS

Figure 50: Ground truth example of the task undertaken, here the task is converting a segmentation map to an image for the A->B Cityscapes dataset.

Figure 51: Final pruned model's generated result for the A->B Cityscapes dataset.





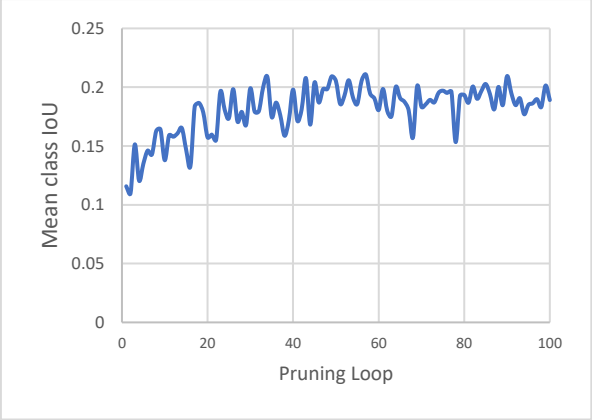


Figure 52: These results compare the results for mean pixel accuracy, mean class accuracy and mean class IOU of the Pix2Pix model against the FCN model over multiple pruning loops for the A->B Cityscapes dataset.

These results show that with consecutive pruning loops all three metrics improve from an initial low value, there isn't any obvious accuracy fall off or disruption to the model performance.

The initial low performance can be contributed to the original model, which on closer inspection, had experienced a mode collapse [150] close to the end of the training loop. The recovery of the performance suggests that this kind of pruning can perhaps avoid and correct for mode collapse, as although mode collapse occurred many times whilst training the original models, it doesn't ever occur when pruning.

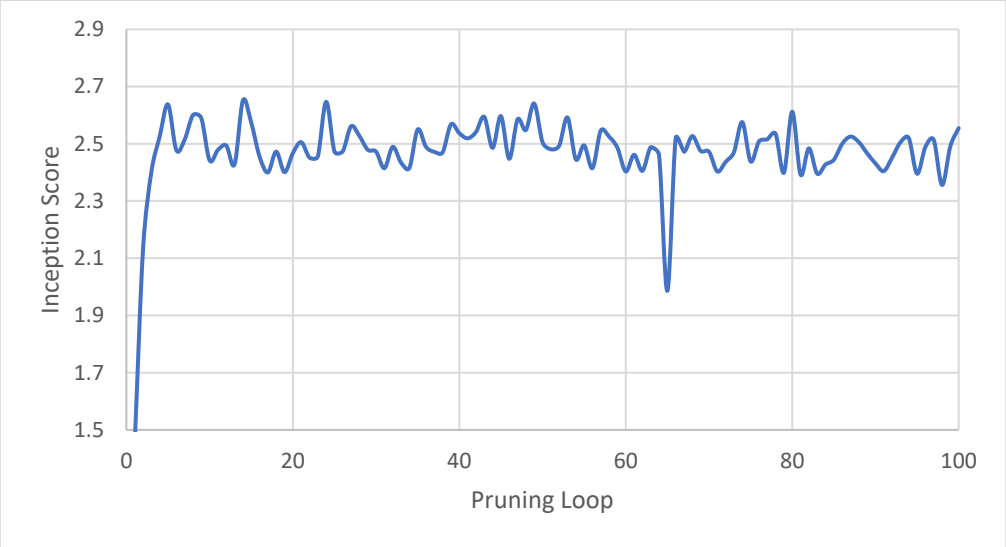


Figure 53: Inception score of the pruned Pix2Pix model at different stages of the pruning loop for the A->B Cityscapes dataset.

Once again, the poor results at pruning loop 0 can be attributed to the mode collapse of the model before pruning was applied, this additionally shows the greater refinement the FCN measurements provide. You can see the slower recovery in Figure 52, whereas the results in Figure 53 suggest the recovery is instant, but both metrics generally track in a similar manner.

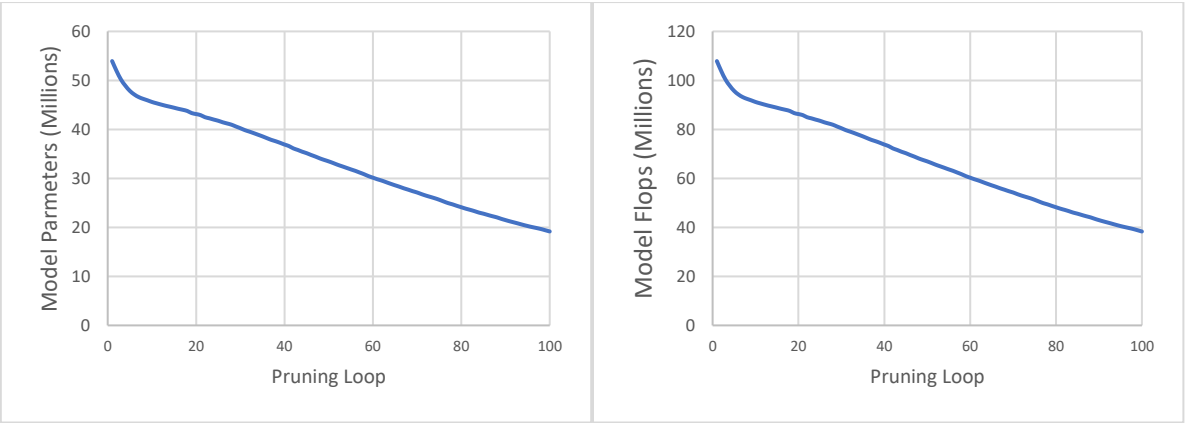


Figure 54: FLOPs and Parameter metrics for the pruned model at different stages of the pruning loop for the A->B Cityscapes dataset.

The decrease in the model's parameters and flops generally mirror each other, this makes sense as the U-Net only contains convolutional layers, the model's structure is shown in Figure 44. When models contain different layer types there can be a greater difference between these two metrics, this is because each layer type has a different parameter to flops ratio.

Additionally, these plots don't mirror the predicted decay rate of 0.005 in figure 46, this is because the decay rate is applied to the number of layers and not the number of parameters. Hence, we get a similar plot but not an exact match.

The model that was selected as the final pruned model was pruning loop 100, this model was selected as the inception score does not seem to be impacted negatively by the pruning. All the FCN metrics also indicate that pruning isn't negatively impacting the model performance and the model is at its most pruned at the end of the process.

Table 9: Final pruned parameters and FLOPs and final metrics for the pruned model of the A->B Cityscapes dataset (Positive values are good negative values are bad)

Pruned Parameters %	Pruned Flops %	Inception Score Δ	Mean Pixel Accuracy Δ	Mean class accuracy Δ	Mean class IoU Δ
64	64	-0.1	+ 0.0483	+ 0.0035	+ 0.0082

Figure 55: Top left is the ground truth image, top right shows the initial trained model before pruning has been applied, bottom left shows the model at pruning step 33, and bottom right shows the model output at pruning step 99.

The results in Figure 55 show what we would expect from our metrics, the images are becoming more detailed over time, the model at pruning loop 33 looks nearly identical to the model at pruning loop 99. It's also interesting to analyse the layers in the model selected for pruning.

Table 10 shows the encoder is pruned much more than the decoder side of the network. The encoder has a total of 1,042 channels when pruned, the decoder has 1,920 channels, it is interesting the pruning is asymmetric as encoder decoder models are traditionally symmetric. This suggests that there is potential for a more efficient model architecture based around a smaller encoder.

Another interesting layer of note is the bottleneck, this is often referenced as the most important layer in a encoder-decoder architecture, however it has been pruned away to a layer size of 1. In fact, the layers either side have also been reduced significantly to 14 and 2, and if the pruning software had the capability the layers of size 1 would have been removed from the model entirely. Assuming these layers are no longer contributing to the model we can recreate the architecture from pruning as follows.

Table 10: Comparison of the number of filters in each layer of the Pix2Pix model before and after pruning for the A->B Cityscapes dataset.

Layer	Starting Number of Filters	Final Number of Filters
Encoder Conv2D	64	47
Encoder Conv2D	128	58
Encoder Conv2D	256	143
Encoder Conv2D	512	253
Encoder Conv2D	512	317
Encoder Conv2D	512	210
Encoder Conv2D	512	14
Bottleneck Conv2D	512	1
Decoder Conv2D	512	2
Decoder Conv2D	512	460
Decoder Conv2D	512	501
Decoder Conv2D	512	509
Decoder Conv2D	256	256
Decoder Conv2D	128	128
Decoder Conv2D	64	64

Table 11: Final configuration of the pruned model with layers containing 1 filter removed for the A->B Cityscapes dataset.

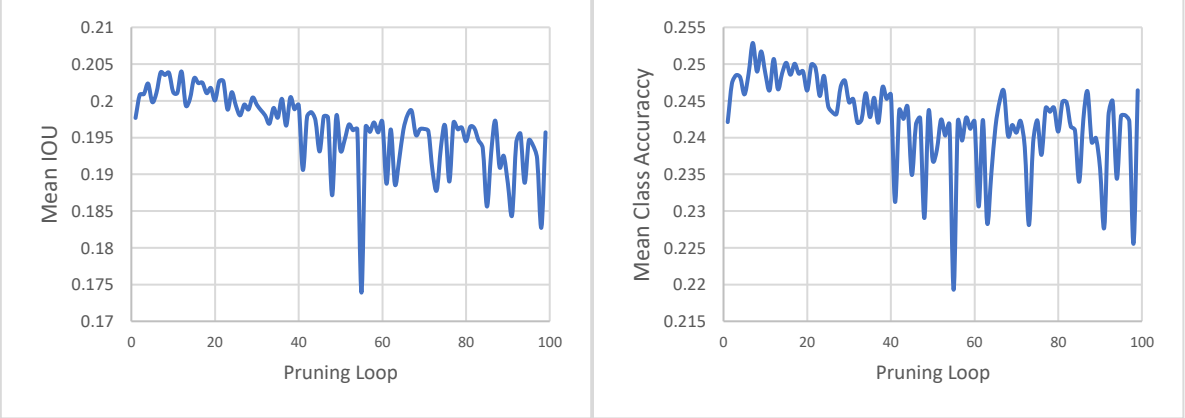
Layer	End Number of Filters
Encoder Conv2D	47
Encoder Conv2D	58
Encoder Conv2D	143
Encoder Conv2D	253
Encoder Conv2D	317
Encoder Conv2D	210
Encoder Conv2D	14
Encoder Conv2D	2
Decoder Conv2D	460
Decoder Conv2D	501
Decoder Conv2D	509
Decoder Conv2D	256
Decoder Conv2D	128
Decoder Conv2D	64

This does seem to mimic the hourglass design of a traditional U-Net network, but unlike the original architecture which creates “compression” of data by reducing the height and width at each layer. The pruned network adds to this compression by using an hourglass approach to the filter numbers. It also seems like a new bottleneck has also emerged with a much-reduced filter number of 2.

6.7 CITYSCAPES B->A RESULTS

Figure 56: Ground truth examples for the B->A Cityscapes dataset.

Figure 57: Selected pruned model example for the B->A Cityscapes dataset.



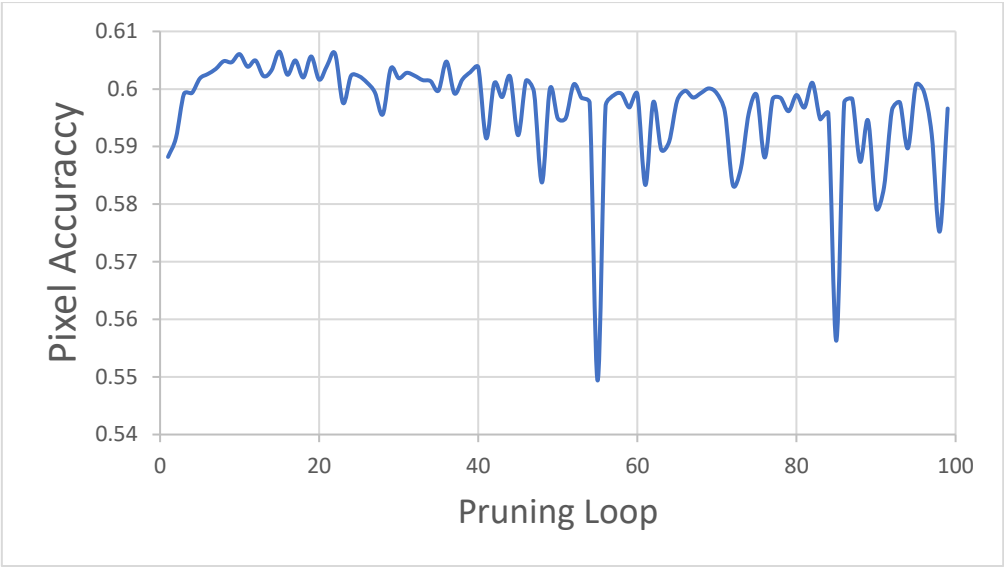


Figure 58: These results compare the results for mean pixel accuracy, mean class accuracy and mean class IOU of the Pix2Pix model against the ground truth over multiple pruning loops for the B->A Cityscapes dataset.

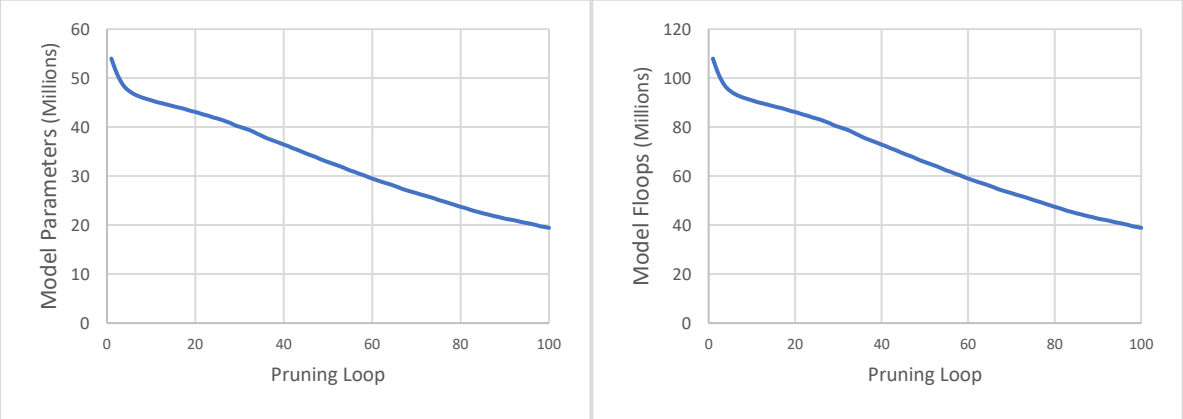


Figure 59: FLOPs and Parameter metrics for the pruned model at different stages of the pruning loop for the B->A Cityscapes dataset.

These results are very different to the A->B results. The class and pixel accuracies were compared to the ground truth this time and not processed by the FCN segmentation model. This is because a reverse FCN model did not exist for this dataset.

There is some initial poor performance, once again this can be contributed to mode collapse, and as it can be seen the pruned model recovers. However, the decrease in these metrics is significant, when compared as a percentage of the pre-pruned model's performance the shift in model performance can be more easily observed.

Table 12: Final pruned parameters and FLOPs and final metrics for the pruned model train on the B->A Cityscapes dataset (Positive values are good negtive values are bad)

Pruned Parameters %	Pruned Flops %	Mean Pixel Accuracy Δ	Mean class accuracy Δ	Mean Pixel Accuracy Δ	Mean class accuracy Δ	Mean class IoU Δ	Mean class IoU Δ
64	64	- 0.0358	- 0.0204	- 5.67 %	- 9.44 %	- 10.84%	- 0.0299

The decrease in performance can be directly observed in the output in figure 60.

Figure 60: Top left is the ground truth segmentation mask, top right shows the initial trained model before pruning has been applied, bottom left shows the model at pruning step 33, and bottom right shows the model output at pruning step 99.

Figure 60 shows that the pruned models are gradually becoming worse at the task of segmentation. It seems that in the A->B task the slow increase of details in the image benefited the metrics, however in the segmentation task this is not true. The segmentation outputs show that the model is trying to add more detail to the segmentation map (this can be seen in the red circled region). Unfortunately, by learning the complexity of these regions, the uniformity needed from a segmentation map has been lost, hence the decrease in the metrics matches the output we're observing here.

Table 13 shows that pruning left 1,105 filters in the encoder and 1,912 filters in the decoder, these are very similar to the pruning results shown in the A->B test, the only difference is the A->B network retains the use of one of the encoder convolutional layers by reducing the filters to 2 and not 1.

These initial results indicate a limitation in the algorithm used to prune these neural networks, this limitation lies in the requirement to prune a set percentage of the network's parameters every pruning loop. This approach may not allow for the neural network to fully adjust to its new architecture before it is pruned again, this will be investigated in the next iteration of the pruning algorithm.

Table 13: Comparison of the number of filters in each layer of the Pix2Pix model before and after pruning for the B->A Cityscapes dataset.

Layer	Starting Number of Filters	Final Number of Filters
Encoder Conv2D	64	60
Encoder Conv2D	128	101
Encoder Conv2D	256	219
Encoder Conv2D	512	389
Encoder Conv2D	512	252
Encoder Conv2D	512	78
Encoder Conv2D	512	6
Bottleneck Conv2D	512	1
Decoder Conv2D	512	1
Decoder Conv2D	512	453
Decoder Conv2D	512	502
Decoder Conv2D	512	509
Decoder Conv2D	256	256
Decoder Conv2D	128	128
Decoder Conv2D	64	64

Table 14: Final configuration of the pruned model with layers containing 1 filter removed for the B->A Cityscapes dataset.

Layer	Final Number of Filters B->A	Final Number of Filters A->B
Encoder Conv2D	60	47
Encoder Conv2D	101	58
Encoder Conv2D	219	143
Encoder Conv2D	389	253
Encoder Conv2D	252	317
Encoder Conv2D	78	210
Encoder Conv2D	6	14
Decoder Conv2D	-	2
Decoder Conv2D	453	460
Decoder Conv2D	502	501
Decoder Conv2D	509	509
Decoder Conv2D	256	256
Decoder Conv2D	128	128
Decoder Conv2D	64	64

6.8 Facades A->B Results

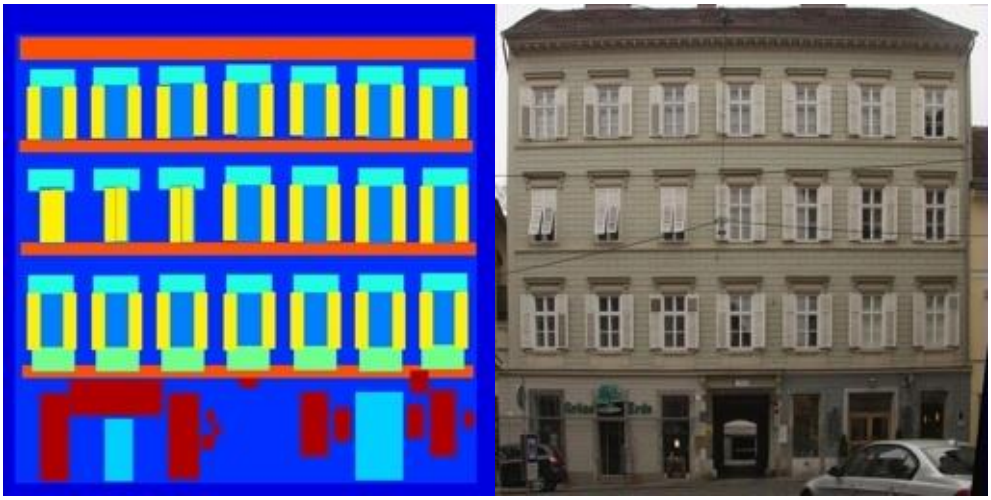


Figure 61: Ground truth example for the A->B Facades dataset.

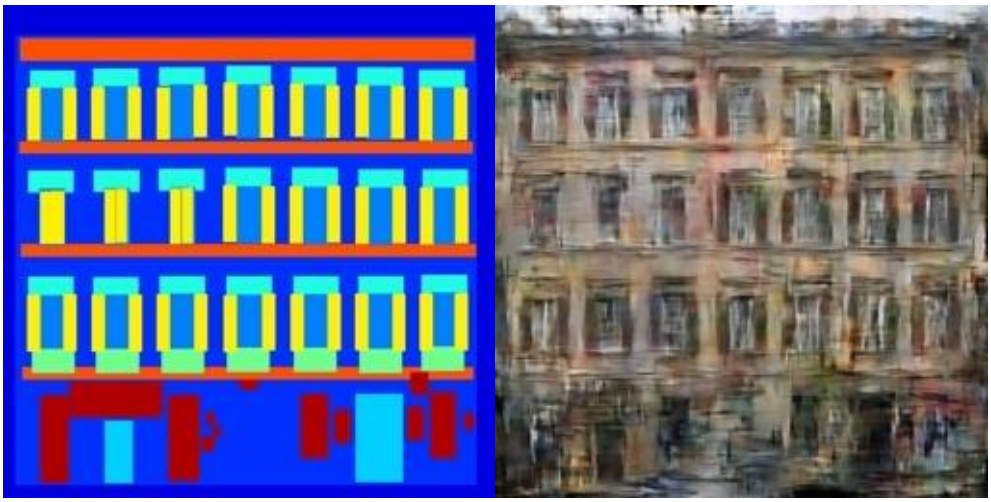
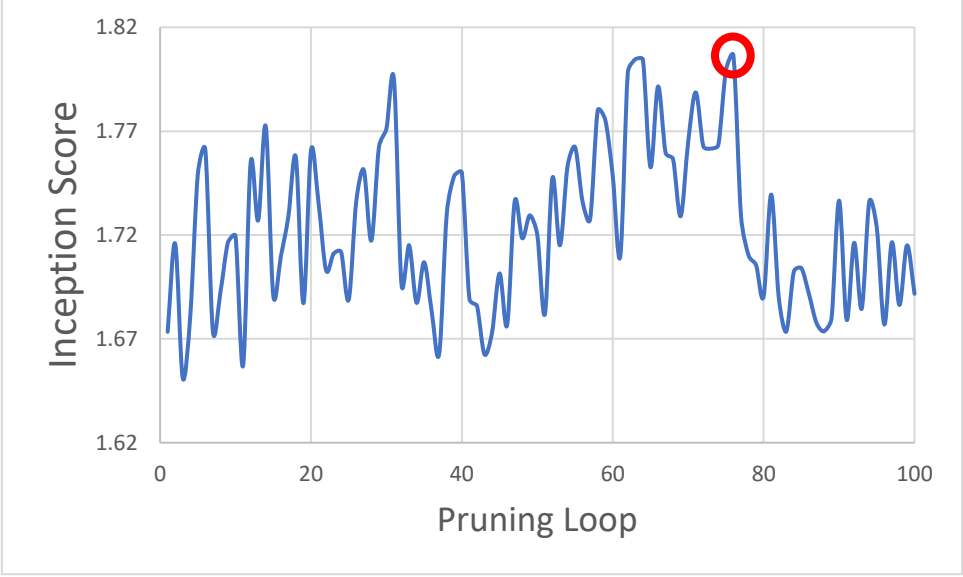


Figure 62: Selected pruned model example for the A->B Facades data.



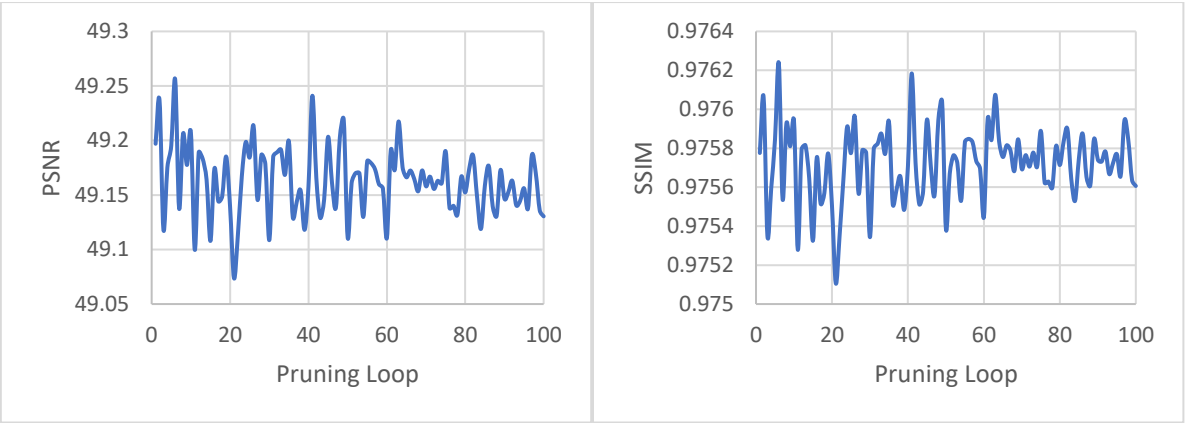


Figure 63: For this dataset an FCN model to provide IOU and segmentation map scores is not available, because of this PSNR and SSIM are included as extra metrics over multiple pruning loops for the A->B Facades dataset.

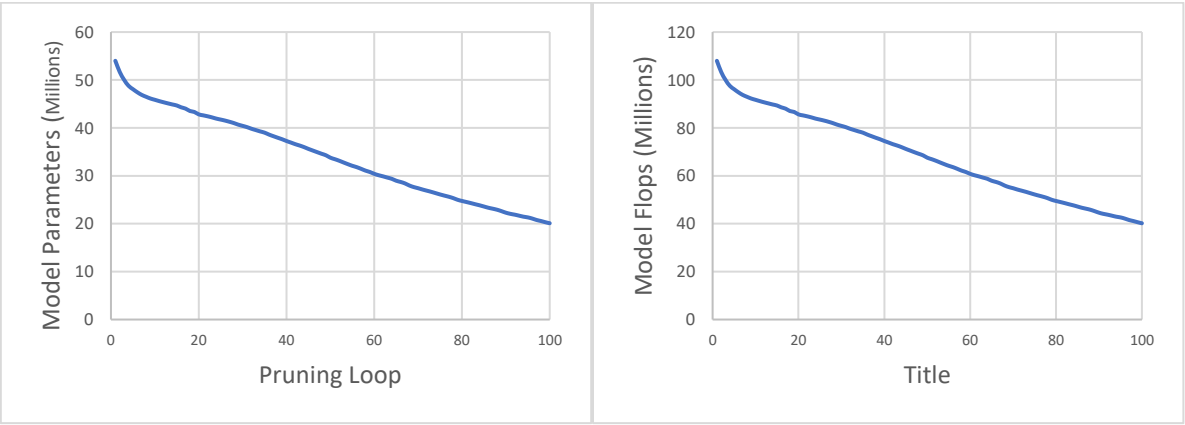


Figure 64: FLOPs and Parameter metrics for the pruned model at different stages of the pruning loop for the A->B Facades dataset.

For the results in table 15 an earlier model was selected for the final model. This was because the inception score of the model falls off after loop 76 (circled in red). Whilst PSNR and SSIM don't see the same fall off, inception score is the most important metric for these generated images [151], and so it takes precedent. (An animation of the facades A->B test can be found in appendix B.3).

Table 15: Final pruned parameters and FLOPs and final metrics for the pruned model (Positive values are good negative values are bad) for the A->B Facades dataset.

Pruned Parameters %	Pruned Flops %	Inception Score Δ	PSNR Δ	Inception Score Δ	PSNR Δ	SSIM Δ	SSIM Δ
52.47	52.47	-0.209	+ 0.384	- 10.38 %	+ 0.78 %	+ 0.27 %	+ 0.002

The pruned model has added a lot of detail from the initial trained model; however, this detail gets muddy. Shown in Figure 66, some of the façade (specifically the area towards the bottom of the image) becomes warped and does not resemble the form of a building anymore. At pruning loop 100 this becomes even more of an issue shown in Figure 66.

Figure 65: Output of the model at pruning loop 100 for the A->B Facades dataset

Figure 66: Top left is the ground truth facade, top right shows the initial trained model before pruning has been applied, bottom left shows the model at pruning step 40, and bottom right shows the model output at pruning step 76 (the selected model) for the A->B Facades dataset.

Table 16: Comparison of the number of filters in each layer of the Pix2Pix model before and after pruning for the A->B Facades dataset.

Layer	Starting Number of Filters	Final Number of Filters
Encoder Conv2D	64	39
Encoder Conv2D	128	75
Encoder Conv2D	256	164
Encoder Conv2D	512	356
Encoder Conv2D	512	372
Encoder Conv2D	512	365
Encoder Conv2D	512	116
Bottleneck Conv2D	512	1
Decoder Conv2D	512	1
Decoder Conv2D	512	490
Decoder Conv2D	512	508
Decoder Conv2D	512	512
Decoder Conv2D	256	256
Decoder Conv2D	128	128
Decoder Conv2D	64	63

Table 17: Final configuration of the pruned model with layers containing 1 filter removed for the A->B Facades dataset.

Layer	End Number of Filters
Encoder Conv2D	39
Encoder Conv2D	75
Encoder Conv2D	164
Encoder Conv2D	356
Encoder Conv2D	372
Encoder Conv2D	365
Encoder Conv2D	116
Decoder Conv2D	490
Decoder Conv2D	508
Decoder Conv2D	512
Decoder Conv2D	256
Decoder Conv2D	128
Decoder Conv2D	63

Table 16 shows that pruning leaves 1,487 filters in the encoder and 1,957 in the decoder, this is more than all the previous loops as these results are taken 25 pruning loops earlier than the previous two experiments. Interestingly there are many more layers still in the encoder which would show that each training loop does not necessarily prune the same ratio from the decoder and encoder with each iteration.

6.9 Facades B->A Results

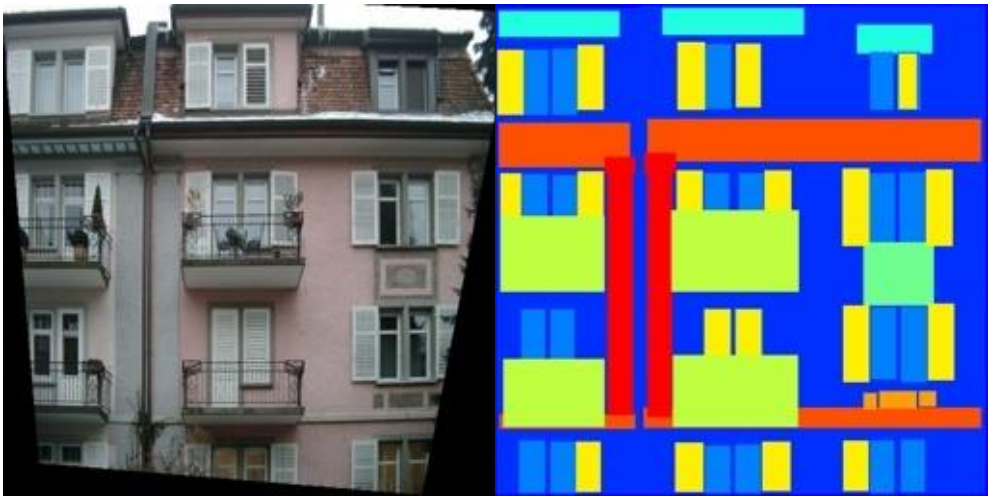


Figure 67: Ground truth example for the Facades B->A dataset.

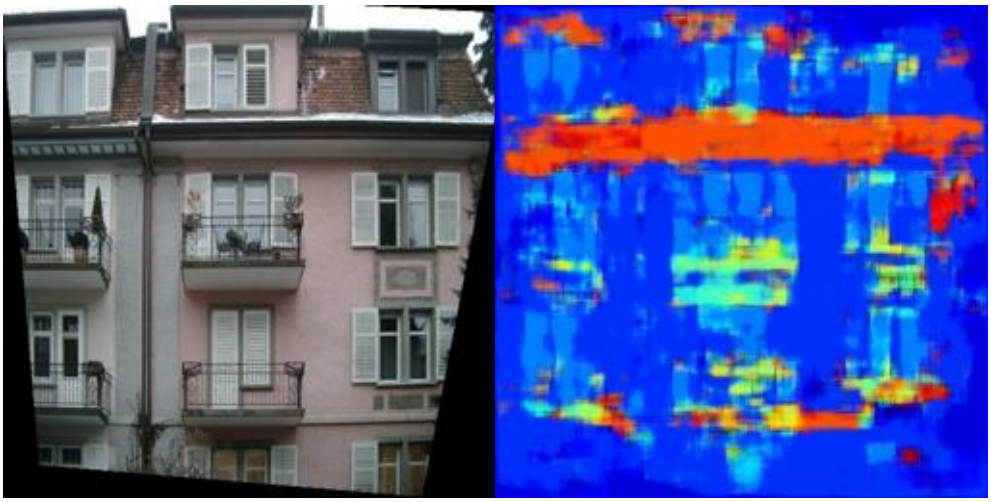
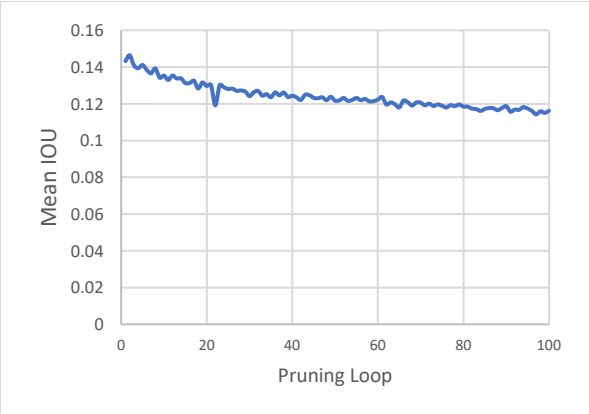
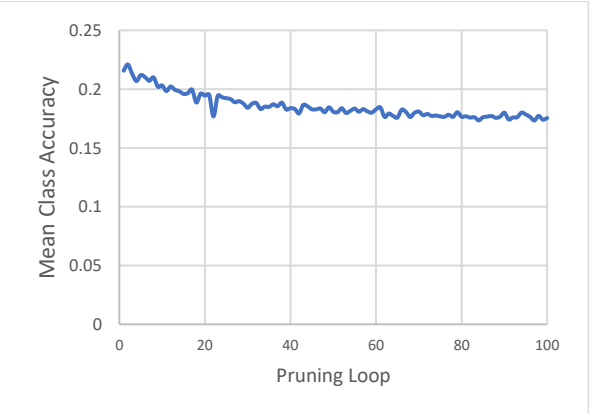


Figure 68: Selected pruned model example for the Facades B->A dataset.





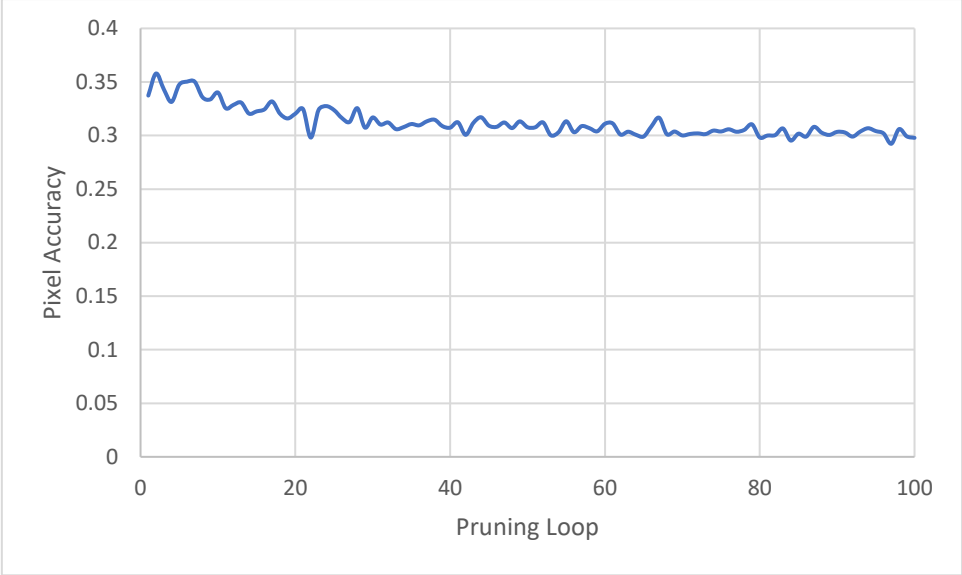


Figure 69: These results compare the results for mean pixel accuracy, mean class accuracy and mean class IOU of the Pix2Pix model against the ground truth over multiple pruning loops for the B->A Facades dataset.

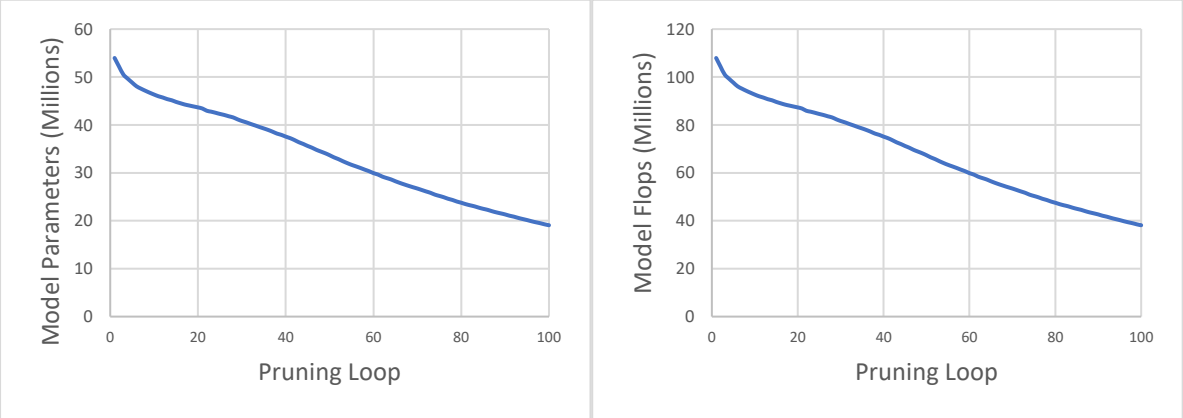


Figure 70: FLOPs and Parameter metrics for the pruned model at different stages of the pruning loop for the B->A Facades dataset.

Table 18: Final pruned parameters and FLOPs and final metrics for the pruned model (Positive values are good negative values are bad) for the B->A Facades dataset.

Pruned Parameters %	Pruned Flops %	Mean Pixel Accuracy Δ	Mean class accuracy Δ	Mean Pixel Accuracy Δ	Mean class accuracy Δ	Mean class IoU Δ	Mean class IoU Δ
64	64	- 0.06	- 0.03	- 16.78 %	- 20.66 %	- 20.62 %	- 0.04

This pruning has the biggest impact on performance observed yet, this is reflected in the visual results too. These results are using the model at pruning loop 100, as although the performance decreases dramatically, there is no sudden drop off, the performance decrease is gradual. This may suggest that this is a harder task than the others tested so far.

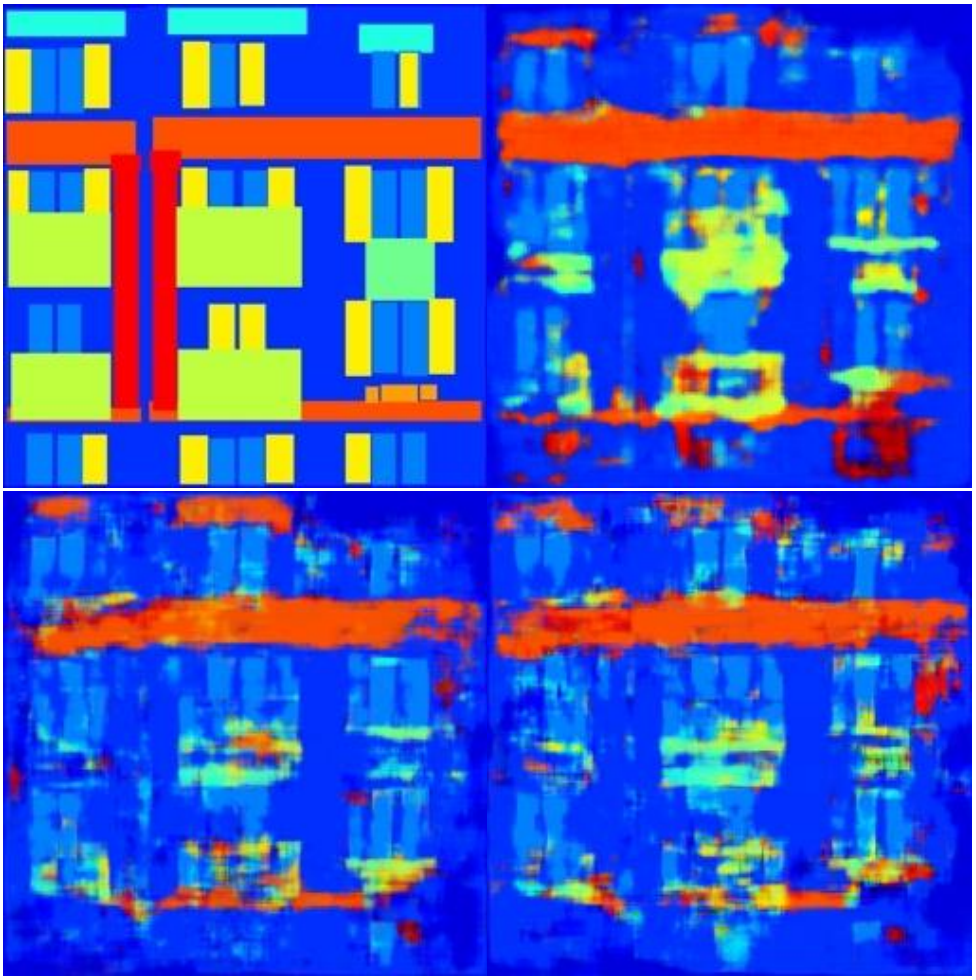


Figure 71: Top left is the ground truth segmentation mask, top right shows the initial trained model before pruning has been applied, bottom left shows the model at pruning step 33, and bottom right shows the model output at pruning step 99 for the B->A Facades dataset.

Indeed, the visual outputs suggest that the initial model never sufficiently represented the segmentation maps in the first place, furthering the idea that this task is more complex.

Table 19: Comparison of the number of filters in each layer of the Pix2Pix model before and after pruning for the B->A Facades dataset.

Layer	Starting Number of Filters	Final Number of Filters
Encoder Conv2D	64	60
Encoder Conv2D	128	101
Encoder Conv2D	256	219
Encoder Conv2D	512	389
Encoder Conv2D	512	252
Encoder Conv2D	512	78
Encoder Conv2D	512	6
Bottleneck Conv2D	512	1
Decoder Conv2D	512	1
Decoder Conv2D	512	453
Decoder Conv2D	512	502
Decoder Conv2D	512	509
Decoder Conv2D	256	256
Decoder Conv2D	128	128
Decoder Conv2D	64	64

Table 20: Final configuration of the pruned model with layers containing 1 filter removed for the B->A Facades dataset.

Layer	End Number of Filters
Encoder Conv2D	60
Encoder Conv2D	101
Encoder Conv2D	219
Encoder Conv2D	389
Encoder Conv2D	252
Encoder Conv2D	78
Encoder Conv2D	6
Decoder Conv2D	453
Decoder Conv2D	502
Decoder Conv2D	509
Decoder Conv2D	256
Decoder Conv2D	128
Decoder Conv2D	64

Table 21: Comparison of the number of filters in the final pruned models for the facades dataset.

Layer	Final Number of Filters B->A	Final Number of Filters A->B
Encoder Conv2D	60	39
Encoder Conv2D	101	75
Encoder Conv2D	219	164
Encoder Conv2D	389	356
Encoder Conv2D	252	372
Encoder Conv2D	78	365
Encoder Conv2D	6	116
Decoder Conv2D	453	490
Decoder Conv2D	502	508
Decoder Conv2D	509	512
Decoder Conv2D	256	256
Decoder Conv2D	128	128
Decoder Conv2D	64	63

Table 20 shows pruning left 1,105 filters in the encoder and 1,912 in the decoder. This is the poorest result in terms of channels removed so far and shows that this pruning method is dependent somewhat on the dataset used to train the model.

6.10 Maps A->B Results

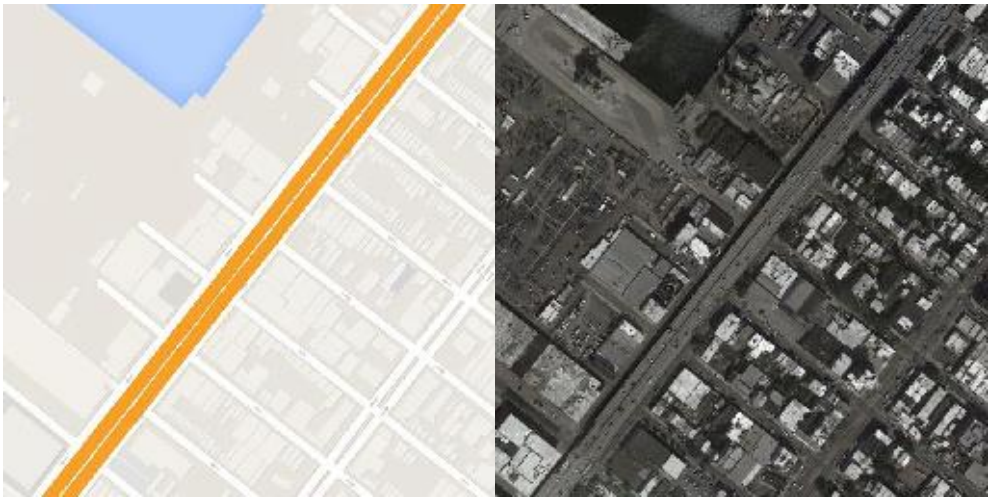


Figure 72: Ground truth example for the Maps A->B dataset.

Selected model output

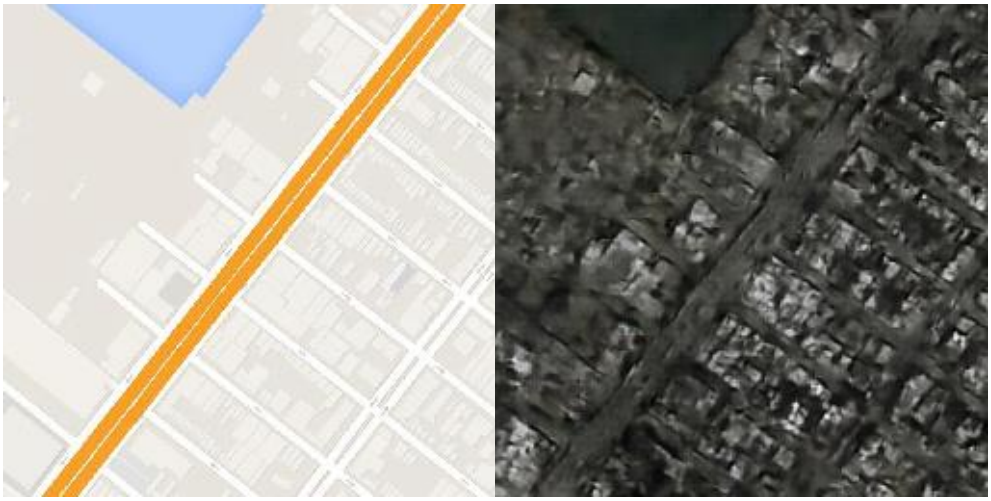
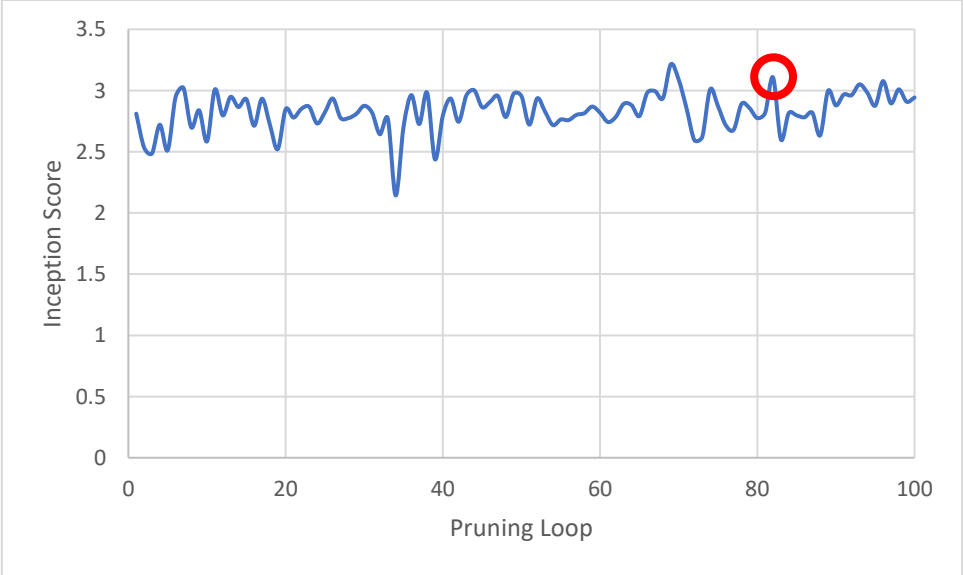


Figure 73: Selected pruned model example for the Maps A->B dataset.



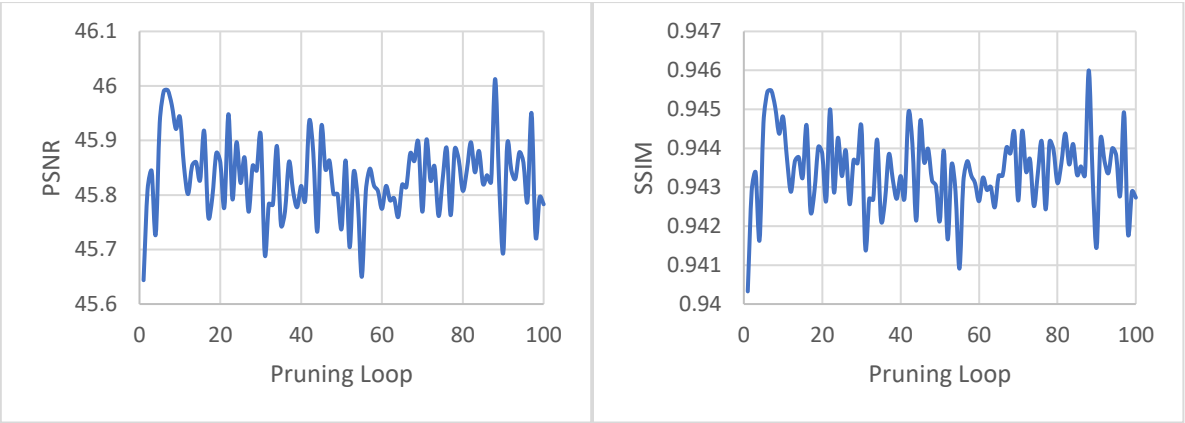


Figure 74: For this dataset an FCN model to provide IOU and segmentation map scores is not available, because of this PSNR and SSIM are included as extra metrics over multiple pruning loops for the A->B Maps dataset.

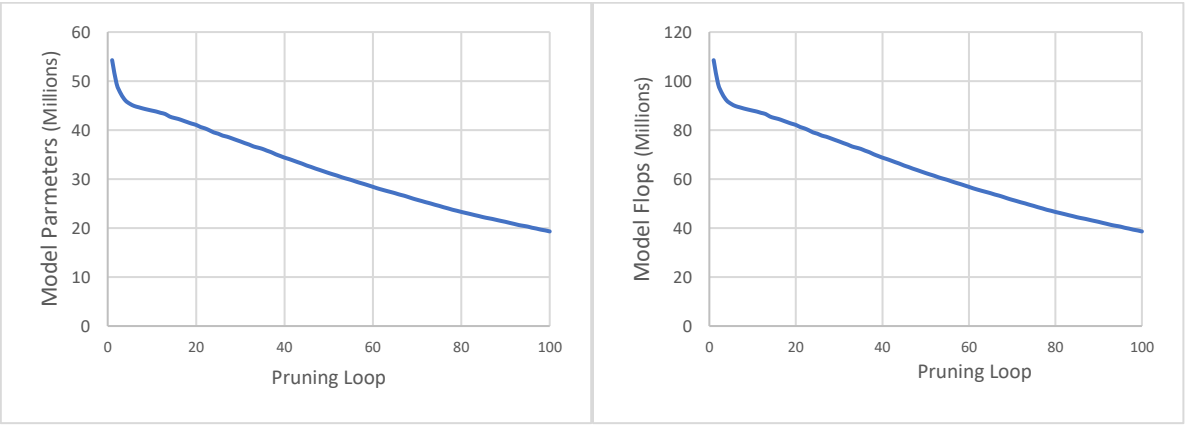


Figure 75: FLOPs and Parameter metrics for the pruned model at different stages of the pruning loop for the A->B Maps dataset.

Table 22 shows the pruning results of model 81, this model was selected because it exhibited the best inception score performance whilst still being a highly pruned network. The decrease in the inception score is comparable to the other real world generative tasks in this section, pruning was stopped prematurely because of the drop off in performance, this reflects the similar result in the A->B results for facades.

Table 22: Final pruned parameters and FLOPs and final metrics for the pruned model (Positive values are good negative values are bad)

Pruned Parameters %	Pruned Flops %	Inception Score Δ	PSNR Δ	SSIM Δ
58	58	- 0.132	+ 0.252	+ 0.004

It seems that all the real-world generative examples have a similar decrease in inception score and can be effectively pruned using this method. However due to the complexity of the task changing with each experiment the point at which an optimal model is achieved also changes.

This makes perfect sense, as that models undertaking a more complex task require more computation to achieve their task effectively, I believe this is the phenomena being observed with these results.

Figure 76: Top left is the ground truth satellite image, top right shows the initial trained model before pruning has been applied, bottom left shows the model at pruning step 33, and bottom right shows the model output at pruning step 99. For the A->B Maps dataset.

Once again, the trade-off between generality and detail is easily observed in Figure 76, the metrics reflect this as inception score does decrease and arguably the results at step 99 are not a good representation of a satellite image shown in Figure 76.

Table 23: Comparison of the number of filters in each layer of the Pix2Pix model before and after pruning for the A->B Maps dataset.

Layer	Starting Number of Filters	Final Number of Filters
Encoder Conv2D	64	16
Encoder Conv2D	128	53
Encoder Conv2D	256	109
Encoder Conv2D	512	266
Encoder Conv2D	512	268
Encoder Conv2D	512	365
Encoder Conv2D	512	179
Bottleneck Conv2D	512	1
Decoder Conv2D	512	3
Decoder Conv2D	512	468
Decoder Conv2D	512	506
Decoder Conv2D	512	510
Decoder Conv2D	256	256
Decoder Conv2D	128	128
Decoder Conv2D	64	63

Table 24: Final configuration of the pruned model with layers containing 1 filter removed for the A->B Maps dataset.

Layer	Final Number of Filters
Encoder Conv2D	16
Encoder Conv2D	53
Encoder Conv2D	109
Encoder Conv2D	266
Encoder Conv2D	268
Encoder Conv2D	365
Encoder Conv2D	179
Decoder Conv2D	3
Decoder Conv2D	468
Decoder Conv2D	506
Decoder Conv2D	510
Decoder Conv2D	256
Decoder Conv2D	128
Decoder Conv2D	63

By inspecting the structure, we once again see that the bottle neck layer has disappeared, but the general hourglass shape of the encoders has arisen again.

One thing of note is the level to which the first Conv2D has been pruned, this has been pruned away to 16 filters, the lowest of all the tests so far. This might be due to the sparsity of the map information, the images used to generate the satellite images have by far the smallest amount of information so far. There are only 4 colours used and all the shapes are blocky or lines, because of this lack in detail the first layers of the network don't have much information to extract, this results in them being pruned away much more aggressively.

6.11 Maps B->A Results

Ground Truth Examples

Figure 77: Ground truth example for the B->A Maps dataset.

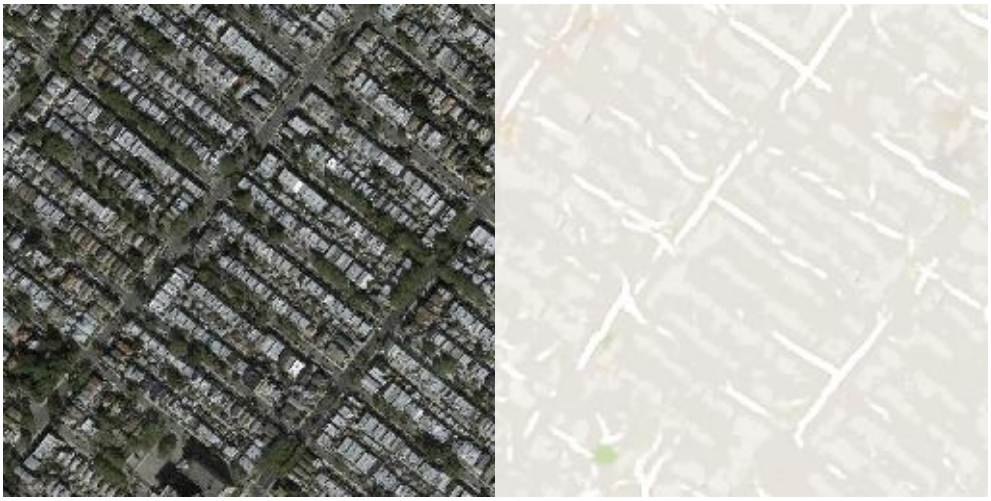


Figure 78: Selected pruned model example for the B->A Maps dataset.

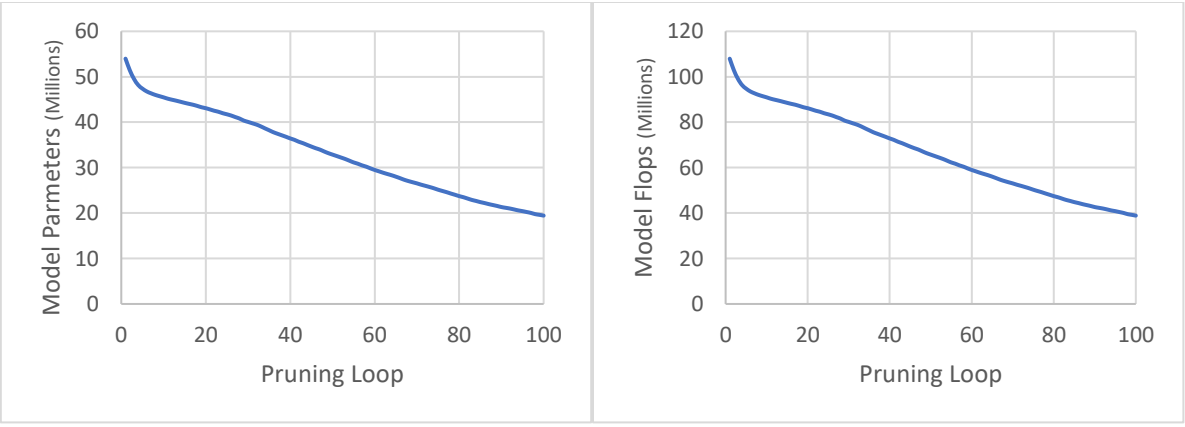


Figure 79: FLOPs and Parameter metrics for the pruned model at different stages of the pruning loop for the B->A Maps dataset.

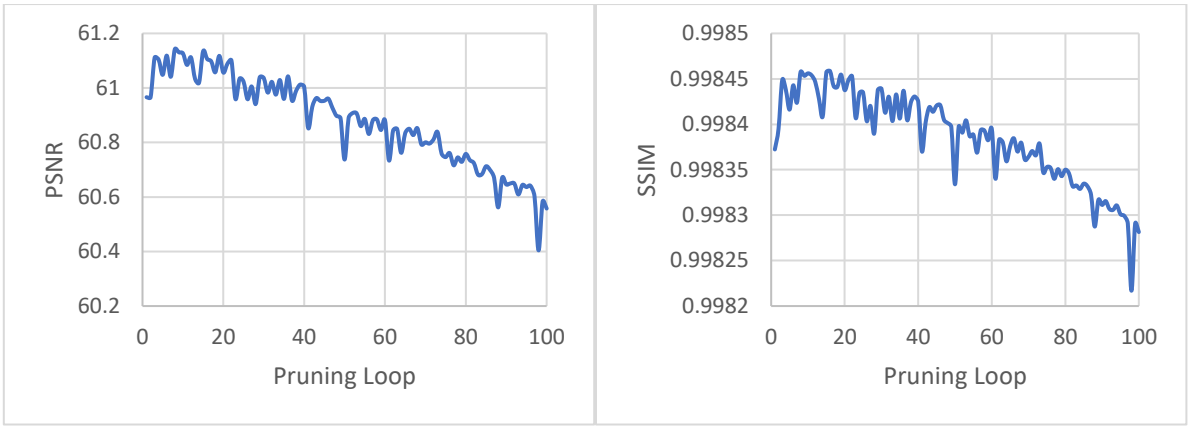


Figure 80: PSNR and SSIM metrics over multiple pruning loops for the A->B Maps dataset.

Table 25: Final pruned parameters and FLOPs and final metrics for the pruned model (Positive values are good negative values are bad) for the A->B Maps dataset.

Pruned Parameters %	Pruned Flops %	PSNR Δ	SSIM Δ
64	64	- 0.79	- 0.0001

This experiment proved extremely hard to evaluate, there is not a mature measurement for satellite images to road map accuracy. However, PSNR and SSIM seem to be a good indicator of the model's performance and so are used to gauge accuracy over time.

Figure 81: Top left is the ground truth map image, top right shows the initial trained model before pruning was applied, bottom left shows the model at pruning step 33, and bottom right shows the model output at pruning step 99. for the A->B Maps dataset.

These results in Figure 81 clearly show the degradation of the model's performance the further through the pruning process. But it also highlights the mediocre performance before pruning had occurred, this once again shows the differences in the complexity of the datasets and how that affects pruning.

Table 26: Comparison of the number of filters in each layer of the Pix2Pix model before and after pruning for the A->B Maps dataset.

Layer	Starting Number of Filters	Final Number of Filters
Encoder Conv2D	64	6
Encoder Conv2D	128	27
Encoder Conv2D	256	46
Encoder Conv2D	512	211
Encoder Conv2D	512	277
Encoder Conv2D	512	293
Encoder Conv2D	512	134
Bottleneck Conv2D	512	1
Decoder Conv2D	512	1
Decoder Conv2D	512	473
Decoder Conv2D	512	508
Decoder Conv2D	512	510
Decoder Conv2D	256	255
Decoder Conv2D	128	128
Decoder Conv2D	64	63

Table 27: Final configuration of the pruned model with layers containing 1 filter removed for the A->B Maps dataset.

Layer	Final Number of Filters
Encoder Conv2D	6
Encoder Conv2D	27
Encoder Conv2D	46
Encoder Conv2D	211
Encoder Conv2D	277
Encoder Conv2D	293
Encoder Conv2D	134
Decoder Conv2D	473
Decoder Conv2D	508
Decoder Conv2D	510
Decoder Conv2D	255
Decoder Conv2D	128
Decoder Conv2D	63

This final model also exhibits the hourglass shape of the previous pruning; however, it has by far the lowest number of filters in the first Conv2D encoder block. This is a confusing result, as the input image is far more complex than the previous experiment, but this layer has been pruned more.

This could be due to the model's architecture, the skip connection is not additive, it is concatenated, meaning the filter at the beginning of the network is presented unchanged at the end of the network. Because this dataset has a large discrepancy between the information of the two sets of images, this could simply be a side effect of the model attempting to stop the propagation of the input image to the output.

By limiting the number of filters, the output images has less of the input propagated to it, however this has the side effect of limiting the number of extractable features. This explains why the performance of this model is so poor after pruning and explains the low filter numbers.

Table 28: Comparison of the number of filters in the final pruned models for the Maps dataset.

Layer	Final Number of Filters B->A	Final Number of Filters A->B
Encoder Conv2D	6	16
Encoder Conv2D	27	53
Encoder Conv2D	46	109
Encoder Conv2D	211	266
Encoder Conv2D	277	268
Encoder Conv2D	293	365
Encoder Conv2D	134	179
Decoder Conv2D	-	3
Decoder Conv2D	473	468
Decoder Conv2D	508	506
Decoder Conv2D	510	510
Decoder Conv2D	255	256
Decoder Conv2D	128	128
Decoder Conv2D	63	63

6.12 Conclusions

This work highlights the importance of the dataset when pruning neural networks, better performance when pruning GAN based neural networks can be achieved [152]. However, this chapter was not intended to create the best GAN pruning mechanism, it was intended to provide comparisons using one network architecture that can do many tasks, this allowed for the comparison of dataset complexity, and reveals the role that the dataset plays when pruning neural networks.

All these trained networks used identical hyperparameters, this was to remove any discrepancies between the level to which any one network was trained. This was done because a network trained for 100 epochs and a network trained for 1000 epochs will be at very different states in terms of "useful" network utilisation. Training these networks to a pre-defined performance level that didn't take epochs into account was considered but was ultimately not used. For this body of work the initial performance of the network is not massively important, the affect pruning has on performance is of much more interest.

This is why the full set of results up to loop 100 are shown, as commented in the results, models should be selected considering a balance between performance and hardware usage.

The cityscapes A->B test shows excellent results, maximum level of pruning was achieved and a minimal reduction in performance. However, the B->A tests show 10% reduction in performance metrics, this is also reflected visually in the results. This effect is noticeable from the beginning of pruning and each step gives a decrease in performance.

This is the first indication that the dataset or task is important when it comes to pruning. Although this is the same dataset and the same number of examples, this result suggests that the A->B task is easier for an ML system to learn than the B->A task.

Intuitively this does not make much sense, as generating a segmentation map seems like a more "machine friendly" task then generating real world images. But in fact, this may reflect that the metrics used to gauge the "realism" of an image are not sufficiently mature [151].

The tests on the Façade dataset shows poor performance on both A->B and B->A tests but once again the segmentation task performs worse.

The tests on the map's dataset are also interesting, once again the model performs better on the task of generating real world images from a map. Whilst a metric to measure a map's "correctness" does not exist, it can be seen from the visual examples that the model performs poorer the further through the pruning loop we get.

The other thing to consider is that although the Pix2Pix methodology is reversible, as proved in this work, it may not be optimal to do so. The general poorer performance on all tasks that generate a representation of the real world (segmentation maps), suggests that the Pix2Pix learning method may be more optimised for generation of real-world images. This could be an optimisation in the model architecture, or the datasets or the learning metrics used.

The pruning algorithm seems to have heavily suffered from pruning a set % of the parameters of a model regardless of how essential they are for the performance of the model. Additionally training GAN networks is difficult, and the discriminator was not pruned. Meaning that there is a high chance that pruning has created an imbalance between the capacity of the generator and the discriminator, cancelling out any gains in training from using these types of networks.

Lack of specific comparisons in literature make it imperative that this pruning method must be tested on generic models and datasets, used to test the efficacy of neural network pruning. This will be the focus of the next chapter of the thesis.

7 PRUNING CLASSIFICATION NETWORKS

7.1 PREFACE

This thesis has investigated a method to prune vision based neural networks, and whilst successful when pruning deblocking networks, the results when pruning Pix2Pix were inconclusive. This was due to a lack in similar works and issues with metric stability. This has made our method extremely hard to evaluate against other methods and hence the efficacy of the pruning method, universally, cannot yet be conclusively proved.

To directly address this issue the pruning method will be updated, and applied to classification tasks, the pruning algorithm will be referred to as the Weight Action Pruning (WAP) from here on in. This is an essential set of experiments because leading pruning methods [61,62,63] are all tested on common models and datasets. By directly comparing these methods and ours, it is possible to achieve the quantitative tests to gauge the effectiveness of WAP.

7.2 PRUNING METHODOLOGY

7.2.1 Network Arch Considerations

Many of these networks are simple in terms of their connections between each layer in the network, LeNet-300, LeNet-5 and VGG-16 all have directly connected layers. If you ignore the task of locating poorly utilized neurons in these networks, then structurally pruning these networks is trivial. This is because in these networks, layer A connects to layer B, layer B connects to layer C and so on, this makes it easy to identify weights that need to be removed when given a neuron that needs to be removed. Additionally, the impact of removing a single neuron is much easier to infer when dealing with single layer-to-layer connections.

The main issues arise when pruning ResNet-50, Res-Net is a highly residual network, this means it has many connections that “skip” through the network and propagate both the training gradients and the activation maps at the early stages of the networks towards the output of the network.

This allows for better training and accuracy; however, these networks create an issue when structurally pruned. Res-Net’s require the channel dimensionality to match at every residual connection in the network, and in ResNet-50 there are 16 residual connections throughout the model. Much like in the work published in ISCAS2021, these kinds of models can be carefully pruned to preserve the dimensionality and for this body of work we wanted to take what we had done there and increase its scope.

Shown in Figure 92, the right-hand block, called the identity block, is repeated a specific number of times depending on the location in the model. Then the dimensionality is reset by the block on the right, called the conv block.

The colours of the boxes, of the identity block, indicate the level to which that layer is prune-able. The green block, in the middle, can be pruned at its input, and at its output. This is because the dimensionality of the data at this layer does not need to match any other connection in the model. The blue block can have its output pruned, but its input dimensionality must remain identical to the skip connection. In the opposite fashion, the red block can have its input pruned freely, but its output must match the dimensionality of the skip connection.

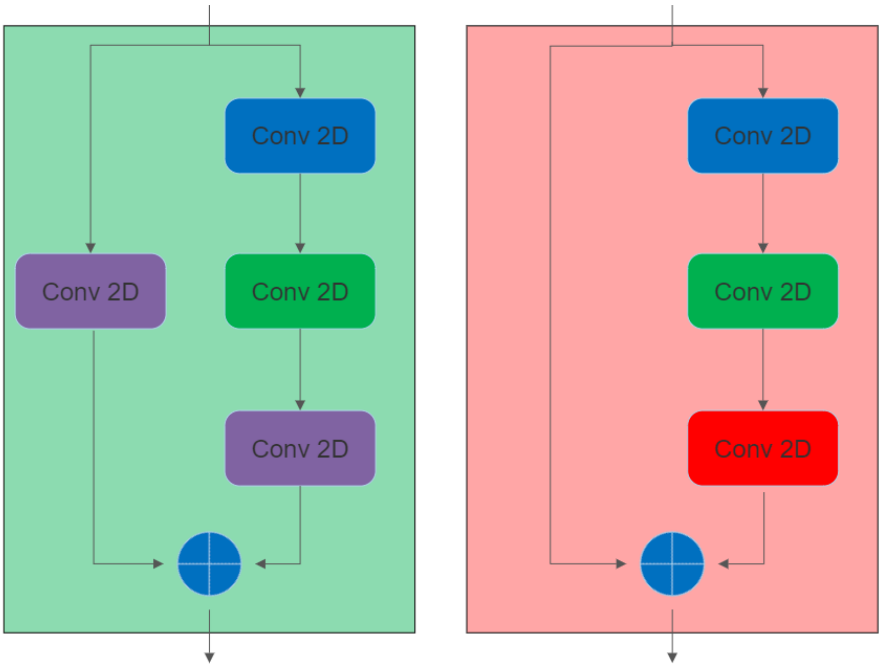


Figure 82: Visual representation of the structural difference between convolutional blocks (left) and identity blocks (right).

The conv block has the effect of resetting this dimensionality, as the purple blocks in Figure 92 sets the channel dimensionality from all subsequent identity blocks. This means, that if the network was to be structurally pruned, without any extra layers being added to the model, the output of these two purple blocks, and the output of the red blocks, in all subsequent identity layers would all have to be identically pruned.

In our previous work we had success with splitting channel data, where dimensionality did not match, and weaving this data back into the main branch, using concatenate and add functions. This method was initially assumed as a good approach to the same issue, however there were some severe complications with this method when applied to ResNet-50.

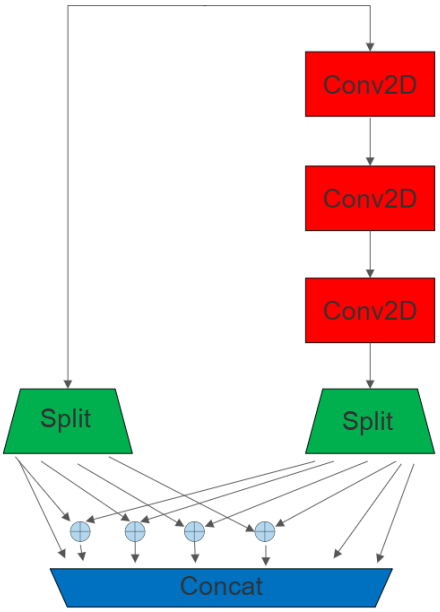


Figure 83: Recombination required for identity blocks due to mismatched filter numbers.

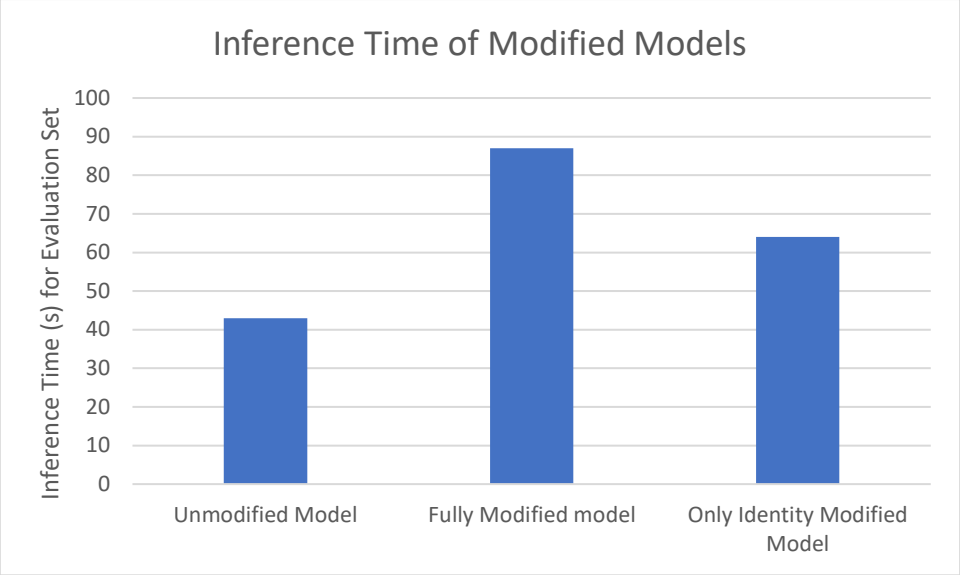


Figure 84: Impact of adding recombination logic to ResNet50.

It was decided to allow for the network to be pruned in its entirety, and not some select channels like in the previous work. To achieve this, a more complex method was implemented, joining outputs which had different dimensionalities. In the example in figure 93, we split not only the output of the side branch, but also the output of the main branch. This allowed for finer pruning of all convolutional layers in the network. However, this had unexpected effect on the inference time of the network.

It was found the addition of the extra components (four concatenation layers and three split layers and two adds) made for a nearly 2x increase in inference time, shown in Figure 94. This is contradictory to the previous set of work, and so to test this we tried splitting the channels identically to the original work [60]. However, this still resulted in a significant inference time increase shown in Figure 94. This is also contradictory to the FLOPs measurements made on the network, as the additional layers showed no increase in the number of parameters or FLOPs. We believe this is down to two main differences, namely, the input data size, and the network structure.

The network this method was first applied to had residual blocks, like ResNet-50. However, the channel size within these blocks was a lot lower, the maximum channel size for the previous network [60] was 96. In contrast, the minimum channel size in a residual block for ResNet-50 is 64, however this is only true in the first identity block. All subsequent blocks (that make up more than 90% of the network) have at least 128 channels, with the majority having 256 and finally topping out at 2048.

This means that adding the split and concatenate layers was adding orders of magnitude more complexity than with the previous model. Instead of splitting 96 channels, now 2048 channels had to be split. Meaning that the last layer alone of ResNet-50, had more computation dedicated to this step than all the residual blocks in the ISCAS2021 work.

However, this was not the end of the issues, ResNet-50 and most other neural networks used for image recognition tasks compress data. Meaning they start with a large input space, in this case 150,528 pixels and reduce it down to a smaller output space, in this case 1000. This is in stark contrast to the work done on filter-based networks [60]. These networks keep the same input and output dimensionality. In image translation networks, the computation time is mostly spent on convolutional layers, processing these images in full resolution. This meant that when the overhead

from the extra layers was added, it was so miniscule in terms of computation compared to the overall model, they had no measurable impact on inference time. Additionally, image filtering networks process 1080p inputs, meaning on average they must process approximately 3,110,400 pixels, this is vastly more than classification networks are expected to process.

ResNet-50 models can process upwards of 100 images per second on modern hardware. When compared with the models used to perform image filtering, that can only process one 1080p frame every 0.5 seconds.

This fact is the easiest way to understand the issue, if you were to add 0.01 seconds of overhead to processing a single filtered frame then this would have a negligible effect on the inference overall [60]. However, if you add 0.01 seconds of overhead to each classified image then 100 images per second turns into 50 images per second, essentially doubling inference time.

Because of this we decided to prune the network where possible, meaning no removal of any of the output channels of the purple and red blocks, the only blocks pruned were the internal green blocks. Whilst this might at first seem to remove a huge amount of the network that can be pruned, there are still 52 blocks that can be pruned, leaving only 20 that can't.

7.2.2 Global Vs Local

This work also explored the differences in pruning globally or pruning locally. This means that the importance of a filter within a layer was compared with importance of the filters in the rest of the network (global). Or the filter's importance was evaluated by comparing filters in the same layer (local).

This became essential when dealing with networks that had both convolutional and fully connected layers, as the magnitude of the importance values was vastly different between the two. This difference could result in either the convolutional layers, or the fully connected layers not being pruned.

7.2.3 Stimulation Dataset

Additionally, a review of how the data was selected to stimulate a response from the filters was undertaken. This was essential as the biggest critiques of the work so far was that "using a section of the validation/test set to identify importance values in filters means the pruning is biased" [60].

This comment was taken from one of the reviewers of the work completed on filter-based networks [60], and whilst the comment is not completely valid as there was never any mixing of datasets, it is described ambiguously in the paper, and so, to be clear.

For all the following experiments there were 3 distinct dataset splits.

- training data
- stimulation data
- validation data

The training and validation data remained identical to the original experiments; stimulation data was obtained by sampling a set % per class of the training dataset. For our experiments this value changed depending on the dataset used. However, any value that represented more than 0.1% of each class was deemed valid. This percentage is proposed as it adequately represents the dataset without biasing the pruning towards the training set [110]. This 0.1% per class simulation data was used to generate the filter importance values, indicating which sections of the network needed to be removed.

Additionally, an experiment was conducted to identify if passing Gaussian white noise at the input of the neural network would identify filter importance. The logic being that this white noise will simply highlight the most highly connected sections of the network. This also had a nice side effect of removing the possibility of bias in the simulation dataset.

7.2.4 Pruning Algorithm V3

This version of the pruning algorithm was far more complex than all other methods up to this point and aimed to combine and test many of the methods developed so far.

7.2.4.1 Inputs in Detail

Global Pruning (GP) – This is a Boolean value to indicate whether global pruning is required or not.

Sparsity Pruning Percentage (SPP) – This is a percentage value that indicates the number of weights that must be zero by the end of the sparsity pruning step.

Layer Percentage Consideration (LPC) – This is a percentage value that indicates the percentage of neurons within a given layer that will be considered for pruning. Ie an LPC of 10% on a layer with 100 neurons will mean 10 are considered for pruning, this is used to reduce computational requirements.

Maximum Layer Deviation (MLD) – This is a float value that is used to control which neurons are pruned, a deviation within MLD from the minimally activating neuron in each layer will select it for pruning.

Re-Training Epochs (ReTEp) – This is an integer value that indicates the number of epochs the network is retrained for when accuracy falls below a given value.

Re-Pruning Epochs (RePEp) – This is an integer value that indicates the number of pruning epochs the network is retrained for when accuracy falls below a given value.

Accuracy Drop before Retraining (ADR) – This is a percentage value that indicates the amount of accuracy the pruned model is allowed to lose before retraining is applied.

Accuracy Drop before Stopping (ADS) – This is a percentage value indicating the accuracy the pruned model is allowed to lose before the pruning process stops due to poor model performance.

Pruned With Signal or Noise (PrSig) – This is a Boolean value indicating whether noise-based or signal-based simulation data should be used.

Dataset Percentage for Stimulated Signals (DatPerSig) – This is a percentage value identifying how much of the dataset should be used for data-based stimulation data.

Initial Training Epochs (ITrEp) – This is an integer value indicating the initial training epochs.

Initial Pruning Epochs(IPrEp) – This is an integer value indicating the initial pruning epochs.

Pretrained model location(PrMLoc) – This is a string that points to the pretrained model on disk.

7.2.4.2 Functions in Detail

apply_sparsity_pruning(model,sparsity_percentage) – This function uses the TensorFlow backend to apply magnitude-based sparsity pruning to the model and a sparsity_percentage for the model to achieve during training. This was setup for our purposes in such a way that sparsity pruning would only be applied to specific layers that had been pre-selected for pruning. This was done by adding "sparse_prune" to the layer name when defining the model, the function returns a model that is ready to be pruned using the standard model.fit functionality in TensorFlow.

get_sim_data(signal,percentage) – This function creates a dataset used for stimulation data, this data is either signal based (from real data) or noise based. In the case of real data, the percentage is used to identify the number of samples from each class to add to the dataset. This happens for each class in the dataset and is returned as the stimulation data. In the case of noise data, the standard deviation and mean is gathered from the real stimulation dataset and white noise is generated with these values, this dataset is the same size and shape as the one that would have been generated from real examples.

find_neurons_2_prune(Pruned_model, Stimulation_Dataset, MLD, LPC) – This function firstly splits the pruned model into lots of smaller models, these splits happen at layers that were defined containing the string "structure_prune" in the layer name. This was done prior to pruning by the user of the software, this created lots of sub models, each having an output corresponding to its filters.

For each of these sub models the stimulation data was passed through the models and the outputs of the filters were recorded and the importance value was obtained in the same fashion as the V2 algorithm. These values were then shortlisted using the LPC, this was used to select the lowest activating filters in any given layer, the exact amount selected depends on the value of LPC. These values are then compared against the MLD and checks if they are within a deviation of the MLD value, further reducing the number of neurons for pruning, but crucially leaving at least one neuron per layer to prune.

These filters and layers are then returned as the output of the function and used to guide the removal of neurons.

globally_trim_neurons(MLD, Trim_Locs) – This function checks if the user has requested the network to be globally pruned or not, if it has, each neuron returned by find_neurons_to_prune is compared in importance regardless of the layer it is contained in. Only if the neuron importance value is within one deviation of the MLD value does it remain in the pool for pruning. This returns the smaller list of neurons to prune.

remove_neurons_n_eval(Trim_Locs) – This function takes the list of neurons marked for removal and removes them from the current model. The model is then tested against the evaluation set and the model is returned to be pruned further.

compare_and_store_models() – This function simply stores the current model and records the number of parameters, FLOPS and VRAM use. It also records accuracy, loss and inference time, these are they key variables to observe when pruning models and deciding on the best trade-off for the task.

7.2.4.3 General pruning loop process

The algorithm starts by identifying if the process is starting from a pre-trained model, like ResNet or VGG. If the model is not pre-trained an initial training is performed, then the model is sparsely pruned, the performance of these models is recorded for reference later.

The stimulation dataset is then generated considering the percentage of the dataset required by the user, and if the user has selected noise-based simulation or data-based stimulation.

Then the pruning loop begins, each loop starts by identifying neurons to prune, this is done using the stimulation dataset, the current sparsely pruned model, and finally the input variables layer percentage consideration and maximum layer deviation. This gives a list of filter numbers that need to be removed from the network, at a given layer.

Next a function checks if the user has requested to globally prune the network or not, if so then all values are compared and only the lowest returned, otherwise nothing is changed.

These filter locations are used by the following function to remove the neurons, and, additionally evaluate the performance of the pruned model.

Next an if statement checks if the model's performance has fallen below the threshold set by the ADR. If it has then the model is retrained and re-pruned. If the model is still performing well nothing happens and the loop restarts.

The second if statement checks if the model's performance has fallen below the ADS threshold. If this block is triggered it means, even with retraining, the model hasn't recovered enough performance, and so the pruning process stops.

The block checking the regen_sim_data simply checks if it's time to regenerate the stimulation data presented to the network. This can be done on a set schedule pre-defined by the user. When the pruning loop ends the final model is stored and evaluated.

PROPOSED PRUNING ALGORITHM V.3

Inputs: Global Pruning (GP), Sparsity Pruning Percentage (SPP), Layer Percentage Consideration (LPC), Maximum Layer Deviation (MLD), Re-Training Epochs (ReTEp), Re-Pruning Epochs (RePEp), Accuracy Drop before Retraining (ADR), Accuracy Drop before Stopping (ADS), Pruned With Signal or Noise (PrSig), Dataset Percentage for Simulated Signals (DatPerSig), Initial Training Epochs (ITrEp), Initial Pruning Epochs(IPrEp), Pretrained model location(PrMLoc).

Outputs: Pruned Networks (Pn)

```
If model is not pretrained:
        Initial_model = model.fit(ITrEp)
Else:
        Initial_model = load_pretrained_model(PrMLoc)
Pruned_model = apply_sparsity_pruning(Initial_model, SPP)
Pruned_model = Pruned_model.fit(IPrEp)
Initial_performance = Initial_model.eval()
Sparsity_Pruned_Performance = Pruned_model.eval()
Stimulation_Dataset = get_sim_data(PrSig, DatPerSig)
Pruning = True
While Pruning:
        Trim_Locs = find_neurons_2_prune(Pruned_model, Stimulation_Dataset, MLD, LPC)
        Trim_Locs  = globally_trim_neurons(MLD, Trim_Locs)
        pruned_model, pruned_model_accuracy = remove_neurons_n_eval(Trim_Locs)
        if pruned_model_accuracy < Initial_performance – ADR:
                pruned_model_rt = pruned_model.fit(RetEp)
                pruned_model_accuracy = pruned_model_rt.eval()
                pruned_model = apply_sparsity_pruning(pruned_model_rt, SPP)
        if pruned_model_accuracy < Initial_performance – ADS:
                break
        compare_and_store_models(pruned_model,pruned_model_rt)
        if regen_sim_data:
                Stimulation_Dataset = Get_Sim_Data(PrSig, DatPerSig)
store_final_model(pruned_model_rt)
```

7.3 Intro to Classification Networks

Classification neural networks are responsible for some of the first widespread successful applications of machine learning [64]. These networks have a visual input, normally a small image, images often go under heavy pre-processing, even before reaching the input neurons of the neural network. Once processed by the neural network the image is classified as a single class indicated by the output of the network, e.g., Dog or Cat.

This pre-processing can be essential in reducing the amount of information needed to successfully classify an image, pre-processing is also used to normalise the inputs so that networks can later handle inputs of varying size and colour spaces [154]. Many different leading methods use vastly different pre-processing techniques [65]. For this body of work, we are going to focus on 4 networks and 5 datasets, the models are below, when possible, the image pre-processing from the original model's implementation will be used.

- LeNet300 [64]
- LeNet5 [64]
- VGG16 [66]
- Resnet50 [67]

And the following datasets

- MNIST [68]
- Fashion-MNIST [69]
- CIFAR 10 [70]
- Birds [71]
- ImageNet [72]

These networks were chosen specifically as they are represented far more frequently than other networks and datasets in the field, as shown in this review paper [73].

These datasets range in the number of classes from 10 to 1000 and the size of the image from 28x28 to 224x224. Additionally, the models range in processing requirements from 266.2k FLOPs to 15.3b FLOPs. For this reason, some specific tests were performed to highlight the differences in dataset size and complexity, to see how datasets impacts pruning neural networks. The decision behind these pairings is fully explained in their specific sections, the final model/dataset pairings are below.

- LeNet300 on MNIST
- LeNet5 on MNIST and Fashion-MNIST
- VGG16 on CIFAR-10 and Birds 300 Kaggle
- Resnet50 on ImageNet

7.4 DATASETS

7.4.1 MNIST

The Modified National Institute of Standards and Technology (MNIST) database is a collection of 70,000 handwritten digits [68]. The images are single channel i.e., black, and white and contain the digits 0-9, no image contains more than one digit. Each image is 28x28 pixels, a sample of the dataset is shown below. This dataset was selected because it was used in 5 out of 10 prominent pruning papers since its creation in 1994 [73].

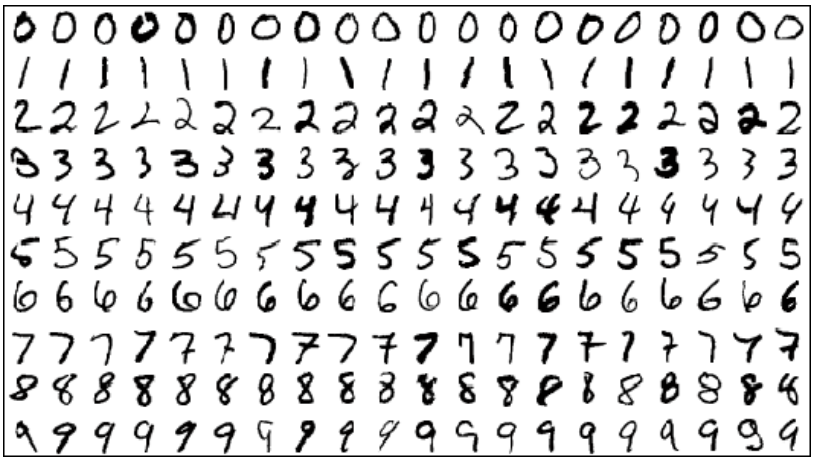


Figure 85: Example of handwritten MNSIT digits [68]

The dataset is perfectly balanced and has 7,000 examples of each class.

7.4.2 Fashion-MNIST

Fashion MNIST was designed as a drop-in replacement for the standard MNIST dataset. The characteristics of dataset are identical to MNIST, i.e., single channel, 28x28 images, 70,000 examples, 10 classes and perfectly class balanced at 7,000 examples per class [69]. This dataset was created because MINST was considered “too easy” for convolutional neural networks to learn [74].

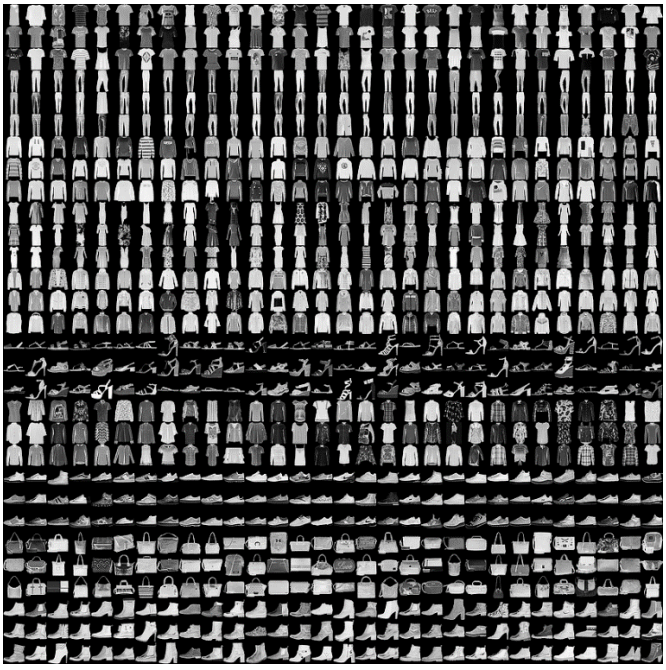


Figure 86: Examples of all 10 Fashion MNIST classes [69].

These are considered harder to learn than digits, which share similar shapes and structures to each other, an example of the dataset is shown below.

This dataset was selected because it is also featured in pruning papers, but additionally to explore the impact of "difficulty" on neural network pruning. Intuitively a simpler task should be easier to prune… by directly testing MNIST digits against MNIST fashion it is possible to draw some initial conclusions on the difficulties of pruning neural networks that are already optimal for the task.

7.4.3 CIFAR-10

Canadian Institute For Advanced Research (CIFAR–10) is a collection of 60,000 images of size 32x32 pixels, the images are colour and contain 3 channels. There are 10 class types, and they are Airplanes, Cars, Birds, Cats, Deer, Dogs, Frogs, Horses, Ships and Trucks. Again, the classes are equally balanced at 6,000 examples per class and an example of the dataset is shown below.

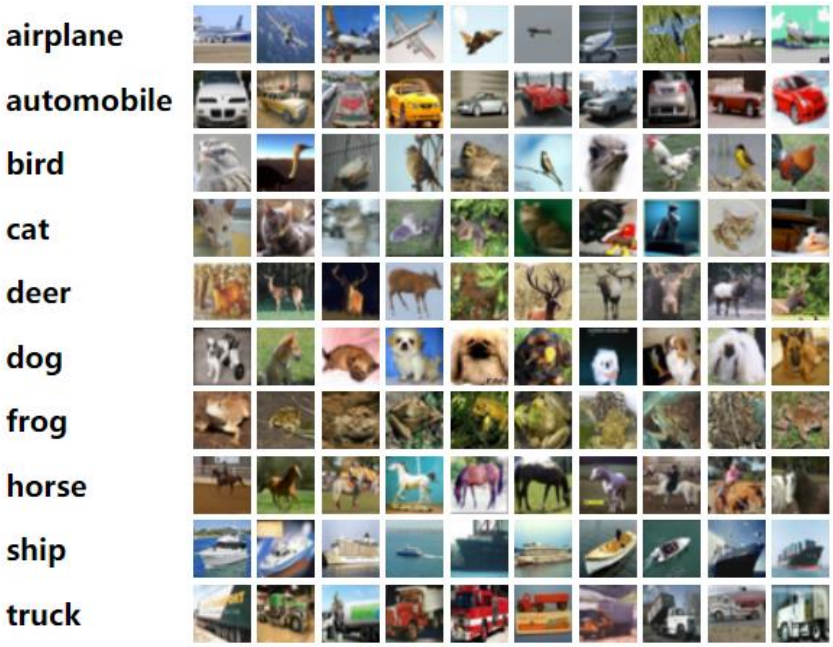


Figure 87: Examples of the 10 classes present in CIFAR-10 [70]

This dataset is once again considered a step up in complexity of the classification task. MINST – Fashion contained objects of 3 main groups, namely, worn clothing, footwear, and bags. CIFAR - 10 contains Animals, Road vehicles, Ships and Planes, a whole extra group to learn.

7.4.4 Birds 300 Kaggle

Birds 300 Kaggle is a subset of the larger Birds 510 Kaggle dataset [71] this dataset contains 300 species of birds from many areas of the globe. Each species is its own class, this includes such bird species as Ostrich, Crow, Wood Duck and more obscure species like Bobolink, Kagu and Sora.

Each image is of size 224x224 pixels, which is the standard input size for VGG16 and RESNET-50. These images have been pre-processed to this size, each bird has been centred in the image, and the image is in colour format. Figure 85 shows some examples from the dataset.

Figure 88: Example of 6 selected classes from the Birds Kaggle dataset [71].

7.4.5 ImageNet

ImageNet is an "image database organised according to the WordNet [75] hierarchy" [72], it contains over 14 million images in total and more than 20,000 classes. Because ImageNet uses a hierarchy to organise the data within, the dataset can be organised in many ways. This is done by selecting different nodes in the hierarchy to define as a class. For instance, a cat is an animal but so is a dog and a frog, in the case that "animal" was chosen as a class, every example from these nodes would be contained within the class. However, if mammal was chosen instead then frog would be excluded, and if corgi was selected both cat and frog would be excluded.

For the purposes of this research the configuration used for the competition in 2012 called ILSVRC2012[72] will be used, this contains 1000 classes and contains 1,281,167 images total. The dataset is unbalanced as shown in Figure 87 there are 732–1300 training images available per class.

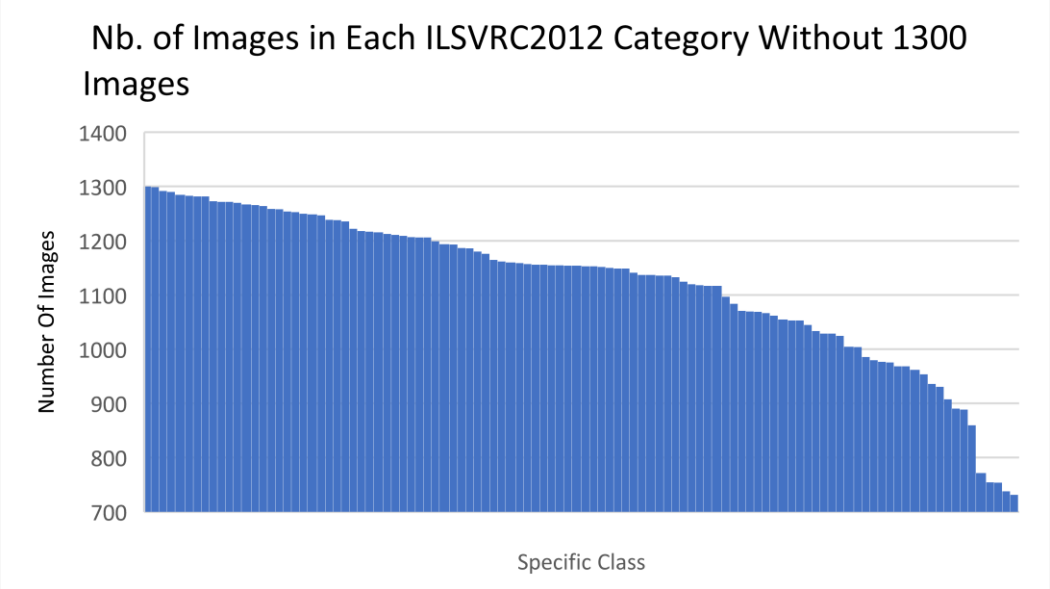


Figure 89: Distribution of the number of examples in the training dataset for ImageNet that contain less than 1300 examples.

7.4.6 Dataset Summaries

Table 29: Summaries of all the datasets used for training.

Dataset	Resolution	Colour	Classes	Nb. Examples	Balanced
MNIST-Digits	28x28	No	10	70,000	Yes
MNIST-Fashion	28x28	No	10	70,000	Yes
CIFAR-10	32x32	Yes	10	60,000	Yes
Birds-300	224x224	Yes	300	50,000	No
ImageNet	Varied	Yes	21,841	14,197,122	No
ILSVRC2012	Varied	Yes	1,000	1,281,167	No

7.5 Neural Network Structures

7.5.1 LeNet-300

LeNet300 [64] is a fully connected network with one hidden layer. The first layer is made from 300 neurons and the second layer 100, each layer has a ReLU activation function applied after each layer, the input is of size 28x28.

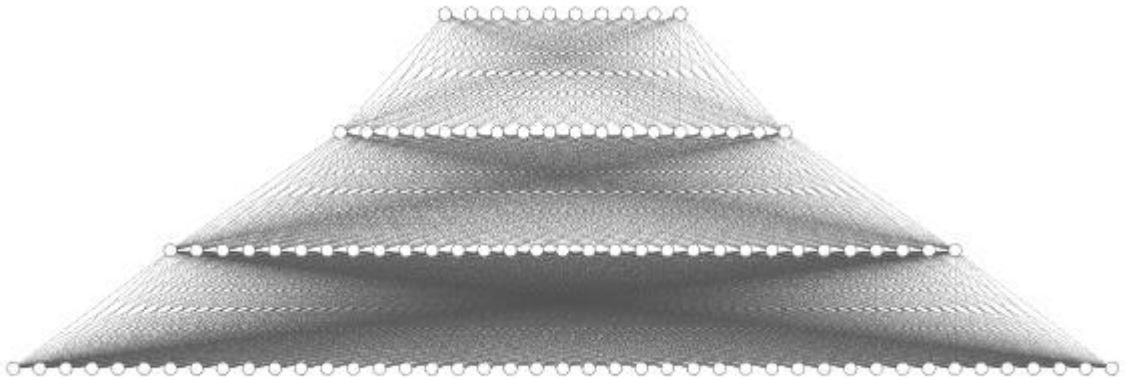


Figure 90: A simplified representation of LE-NET 300

The network was designed to be computationally efficient in processing data when compared to LeNet-5. This network was originally designed to detect hand-written MNIST digits. Images were flattened so that the network has an input shape of 784 float values. Digits were normalised to make sure the mean input was zero and variance was one. This is a well-established approach to aid with the learning process. The network has 265,400 FLOPs and 265,600 parameters, this is broken down by layer in Table 30.

This network was found to be able to achieve a 3.05% top 1 error when tested on MNIST digits, some papers since have claimed higher accuracy for this same architecture [76]. This is a pervasive issue in the field of neural network pruning, it seems sometimes pruning methods are comparing the efficacy of re-training neural networks, rather than the pruning capabilities of the method they have developed.

Table 30: The number of FLOPs and parameters in each layer of LE-NET 300

Layer	Flops	Parameters
Dense 1	235350	235500
Dense 2	30050	30100

Because of the accuracy issue laid out above, where possible pruned neural networks will be compared using an accuracy difference from the original un-pruned model. This will hopefully show the effectiveness of the pruning method and not the improvements through any new training techniques.

7.5.2 LeNet-5

LeNet-5 is a convolutional neural network comprised of seven layers not including the input layer which is of size 32x32 pixels [77]. The network was used to achieve an error rate of 0.95% on the MNIST digits dataset, the input size is obviously more than the 28x28 pixels of standard MNIST, this was done to centre the digit and to achieve this the digit was padded with zeros on all sides. Digits are also normalised so that the mean input is zero and the variance is one, this is used once again to aid with the speed of learning.

The other four layers are listed in Table 31 with their flop and parameter amount, tanh is used as an activation function at every layer, and a sigmoid activation function is used after every pooling layer.

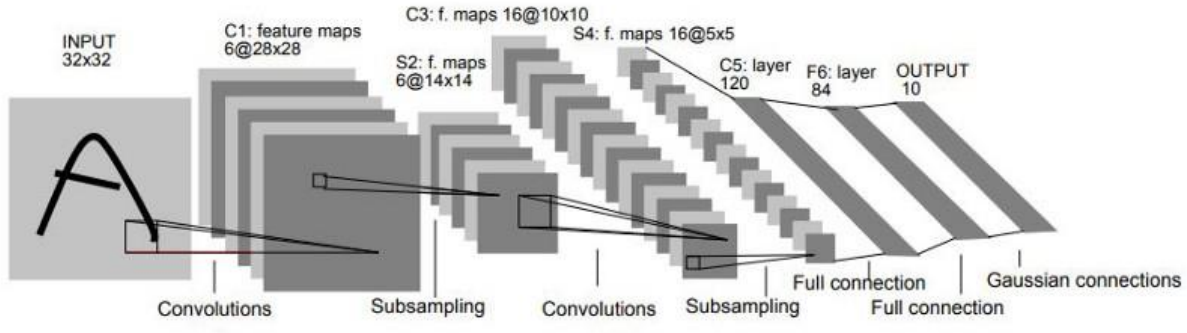


Figure 91: A simplified representation of LE-NET 5 [77]

Table 31: The number of FLOPs and parameters in each layer of LE-NET 5

Layer	Flops	Parameters
Conv1	307200	208
Conv2	1228800	3216
Dense1	138240	69240
Dense2	20160	10164

These measurements are with the channel configuration of 6,16,120,84. Other methods use a starting channel configuration of 20, 50, 800, 500. This configuration is now considered overkill for a problem such as MNIST.

7.5.3 VGG-16

VGG-16 is a configuration of a neural network "ConvNet" developed for the ImageNet 2014 challenge [66] the 16 indicates that the model has 16 weight layers. This model was originally trained on a subset of ImageNet containing 1000 classes and an input size of 224x224 coloured images.

For this work we will focus on two versions of the VGG-16 model. One that was fine-tuned from this original VGG-16 model. This fine-tuned model has been trained on CIFAR-10 meaning that the input size has been reduced to 32x32 pixels and the number of output classes has been reduced to 10. The other is the model that was trained on ImageNet, however the final, classification layers will have been removed and replaced with new layers to tune the network to a new dataset.

Whilst this is a commonly used test in the field of neural network pruning, I struggle to understand why it is so widely adopted. This model was designed to classify 1000 classes, from a much more complex dataset. By reducing the number of classes by 100 times it can be expected that the network can be reduced in complexity through pruning easily. This might be to show how a large network can distil its knowledge into a smaller model for a smaller task, however the idea that you would begin with a model that is 100 times too complex for the task is questionable at best. In any case, it is a widely used test amongst pruning papers and so I have included it.

Table 32 shows the breakdown of VGG-16's FLOPs and parameters, and Figure 90 shows a simplified representation of the structure.

Table 32:The number of FLOPs and parameters in each layer of VGG-16

Layer	Flops	Parameters
Conv1	3538944	1792
Conv2	75497472	36928
Conv3	37748736	73856
Conv4	75497472	147584
Conv5	37748736	295168
Conv6	75497472	590080
Conv7	75497472	590080
Conv8	37748736	1180160
Conv9	75497472	2359808
Conv10	75497472	2359808
Conv11	18874368	2359808
Conv12	18874368	2359808
Conv13	18874368	2359808
Dense1	524288	262656

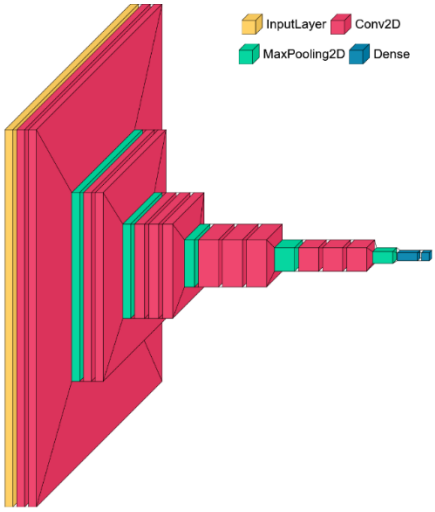


Figure 92: A simplified representation of VGG-16 generated using [78].

7.5.4 ResNet-50

ResNet-50 is a configuration of the Res-Net family of neural networks which attempts to overcome the issue of vanishing gradients [155]. This is done in deep neural networks by adding many residual connections to the network [67]. Res-Net was trained on the ImageNet subset ILSVRC2012 in its standard configuration, meaning coloured images of 224x224 pixels, additionally the dataset contains 1000 classes in this format.

Res-Net's are made from convolutional blocks and identity blocks, the number of filters in these blocks differ depending on the configuration, the configuration for ResNet-50 is shown in Table 33.

Table 33: Different configuration modes for Res-Net, the Selected model used in the research is highlighted in green.

Layer Name	Output Size	18-Layer	34-Layer	50-Layer	101-Layer	152-Layer
Conv1	112x112	7x7, 64, stride 2				
Conv2_x	56x56	3x3 max pool, stride 2				
		$\begin{bmatrix}3\times3,64\\3\times3,64\end{bmatrix}\times2$	$\begin{bmatrix}3\times3,64\\3\times3,64\end{bmatrix}\times3$	$\begin{bmatrix}3\times3,64\\3\times3,64\\3\times3,256\end{bmatrix}\times3$	$\begin{bmatrix}3\times3,64\\3\times3,64\\3\times3,256\end{bmatrix}\times3$	$\begin{bmatrix}3\times3,64\\3\times3,64\\3\times3,256\end{bmatrix}\times3$
Conv3_x	28x28	$\begin{bmatrix}3\times3,128\\3\times3,128\end{bmatrix}\times2$	$\begin{bmatrix}3\times3,128\\3\times3,128\end{bmatrix}\times4$	$\begin{bmatrix}3\times3,128\\3\times3,128\\3\times3,512\end{bmatrix}\times4$	$\begin{bmatrix}3\times3,128\\3\times3,128\\3\times3,512\end{bmatrix}\times4$	$\begin{bmatrix}3\times3,128\\3\times3,128\\3\times3,512\end{bmatrix}\times4$
Conv4_x	14x14	$\begin{bmatrix}3\times3,256\\3\times3,256\end{bmatrix}\times2$	$\begin{bmatrix}3\times3,256\\3\times3,256\end{bmatrix}\times6$	$\begin{bmatrix}3\times3,256\\3\times3,256\\3\times3,1024\end{bmatrix}\times6$	$\begin{bmatrix}3\times3,256\\3\times3,256\\3\times3,1024\end{bmatrix}\times23$	$\begin{bmatrix}3\times3,256\\3\times3,256\\3\times3,1024\end{bmatrix}\times36$
Conv5_x	7x7	$\begin{bmatrix}3\times3,512\\3\times3,512\end{bmatrix}\times2$	$\begin{bmatrix}3\times3,512\\3\times3,512\end{bmatrix}\times3$	$\begin{bmatrix}3\times3,512\\3\times3,512\\3\times3,2048\end{bmatrix}\times3$	$\begin{bmatrix}3\times3,512\\3\times3,512\\3\times3,2048\end{bmatrix}\times3$	$\begin{bmatrix}3\times3,512\\3\times3,512\\3\times3,2048\end{bmatrix}\times3$
	1x1	Average pool, 1000 Dropout Fully connected, Softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

The original model for this work is taken from the Keras applications webpage [79], this model was then pruned using the same image pre-processing techniques as the original Res-Net paper. Pruning a model and keeping the task the same is a good test of how well a pruning method can extract and transfer knowledge to a smaller model.

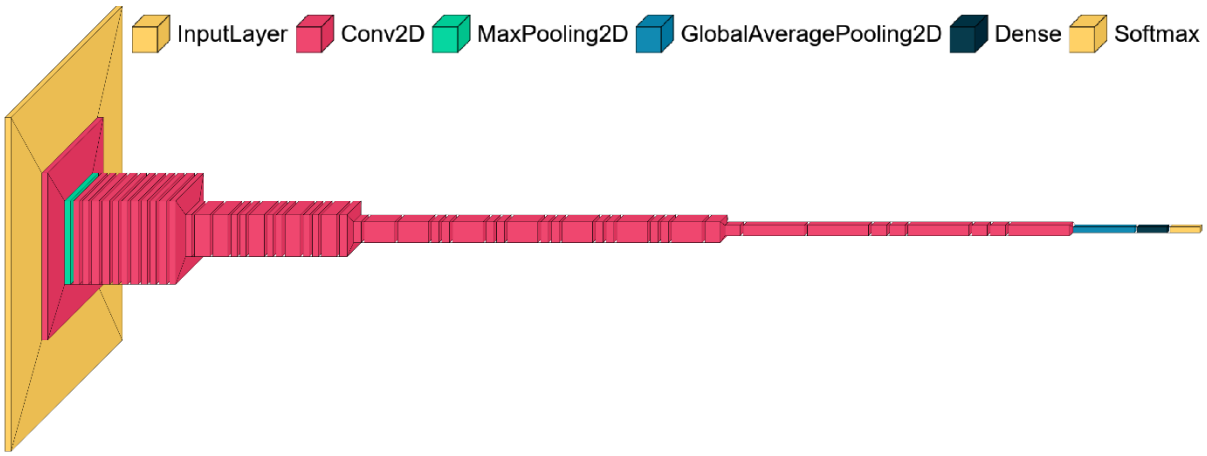


Figure 93:A simplified representation of ResNet-50 generated using [78].

Table 34: Describing each layer in a table would create a massive table therefore this table records each identity and convolutional block's FLOPs and parameters of ResNet-50.

Block	Flops	Parameters
Conv Block 1	463,224,832	74,624
Identity Block 1	437,534,720	70,304
Identity Block 2	437,534,720	70,304
Conv Block 2	745,414,656	377,392
Identity Block 3	437,534,720	278,864
Identity Block 4	437,534,720	278,864
Identity Block 5	437,534,720	278,864
Conv Block 3	745,213,952	1,507,608
Identity Block 6	436,932,608	1,114,280
Identity Block 7	436,932,608	1,114,280
Identity Block 8	436,932608	1,114,280
Identity Block 9	436,932,608	1,114,280
Conv Block 4	745,113,600	6,029,452
Identity Block 10	436,832,256	4,456,532
Identity Block 11	436,832,256	4,456,532

7.6 Hyperparameters

Table 35: Hyperparameters used for training and pruning all the models in the experiments in the following sections.

Dataset	Batch Size	Initial Training Epochs	Initial Pruning Epochs	Global Pruning	Learning Rate	Optimiser	Loss Function	Sparsity Pruning Percentage
LeNet-300 MNIST	256	75	75	True	Init 0.01 Drop 0.94 /5 epochs	SGD	Sparse Categorical Cross entropy	Init 50% Final 90%
LeNet-5 MNIST	256	75	75	True	Init 0.01 Drop 0.94 /5 epochs	SGD	Sparse Categorical Cross entropy	Init 0% Final 40%
LeNet-300 Fashion	256	75	75	True	Init 0.01 Drop 0.94 /5 epochs	SGD	Sparse Categorical Cross entropy	Init 0% Final 40%
VGG-16 CIFAR-10	256	70	70	True	Init = 0.001 Epoch 50 = 0.0001	SGD	Categorical Cross entropy	Init 50% Final 80%
VGG-16 Birds	64	15	10	True	Pre-warm up 0.0001 Post-warm up 0.00001	Adam	Categorical Cross entropy	Init 50% Final 80%
Resnet-50 ImageNet	128	-	10	Both	Init = 0.001 Epoch 25 = 0.0001 Epoch 50 = 0.00001	SGD	Sparse Categorical Cross entropy	Init 30% Final 60%

Dataset	Layer Percentage Consideration	Maximum Layer Deviation	Re-Training Epochs	Re-Pruning Epochs	Accuracy Drop before Retraining	Accuracy Drop before Stopping	Pruned With Signal or Noise	Dataset Percentage for Simulated Signals
LeNet-300 MNIST	20 %	0.01	50	50	0.1 %	1.5 %	Signal	4 %
LeNet-5 MNIST	10 %	0.01	50	50	0.1 %	1.5 %	Signal	4 %
LeNet-300 Fashion	10 %	0.01	50	50	0.1 %	3 %	Signal	4 %
VGG-16 CIFAR-10	10 %	0.01	45	45	0.3 %	1.5 %	Both	0.2 %
VGG-16 Birds	5 %	0.01	10	10	0.5 %	2.7 %	Signal	3.3 %
Resnet-50 ImageNet	5 %	0.001	15	10	0.5 %	2.7 %	Signal	0.4 %

RESULTS

7.6.1 LeNet-300 on MNIST

The model was trained to an initial accuracy of 98.4%, after which, initial sparsity pruning was applied, the sparsely pruned model also achieved 98.4% accuracy.

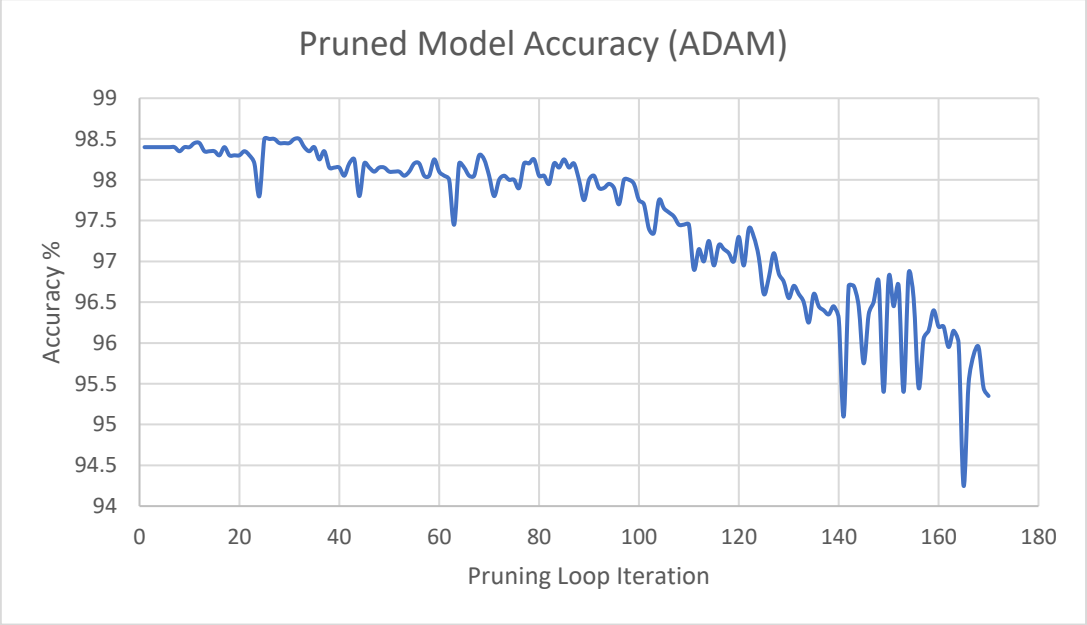


Figure 94: The accuracy results using the Adam optimiser over multiple pruning loops.

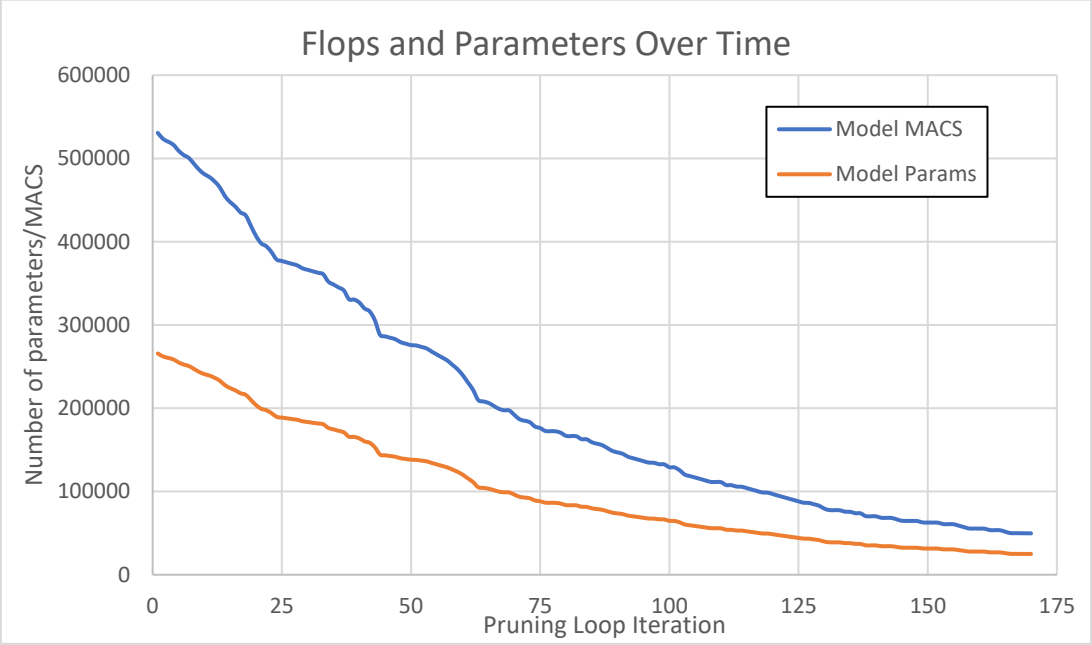


Figure 95: Results for the ADAM optimiser, here MACS were plotted as the FLOPS and parameters match so closely it would look as if only one plotted (MACS is just FLOPS x 2).

7.6.1.1 Comparison to Similar Techniques

Comparing to other methods is difficult because pruning is a broad field and there are many techniques, most focus on pruning weights in the network. Whilst this is a form of pruning commonly used for reducing the size of the model on disk, or reducing the computation required on

a CPU by using other additional techniques [80], this is not the type of pruning WAP achieves. Comparisons will therefore focus mostly on techniques that prune entire neurons and filters (structured pruning), this technique mirrors ours and provides inherent gains in computation time on CPU and GPU devices.

*Table 36: SOTA pruning methods using the model LeNet-300 and dataset MNIST, best result in **bold** second best underlined.*

Method	Parameters	Flops	Error	Pruning Type
Ours ADAM	66,453	66,282	2.05% (Δ- 0.45%)	Structured
Ours SGD Global	25,947	77,384	1.89% (Δ- 0.3%)	Structured
Ours SGD Local	24,807	74,133	1.99% (Δ- 0.4%)	Structured
Auto Prune [81]	-	23,978	1.82% (Δ- 0.22%)	Structured
Corset [82]	26,000	-	2.03% (Δ- 0.13%)	Structured
MIXP [83]	**3,199**	19,982	- % (Δ -0.3%)	Semi *
NeST [84]	7,844	**14,900***	**1.29%** (Δ- 0.31%)	Semi *
GrowPrune [85]	12,200	-	1.4% (Δ-)	Sparse
EffNN [86]	22,000	-	1.59% (Δ- 0.05%)	Sparse
DiffEvo [87]	10,000	-	2.07% (Δ- 0.19%)	Sparse
PartSwarm [88]	44,369	-	2.2% **(Δ+ 0.06%)**	Sparse
NoiseOut [89]	11,225	-	3.05% (Δ-)	Sparse

These results show that our technique performs worse than all other techniques except when compared by parameters, for some examples. However, all other techniques use input pruning, this is where connections are cut to the input layer. This provides a huge gain that is unachievable with our current approach. The second thing to consider in these results is that we found the SGD optimiser with local pruning (in green) performs much better on average than the Adam optimiser.

Table 37: Comparison of using Adam and SGD optimisers and using Global/Local pruning.

Our Method	1st Layer Neurons	2nd Layer Neurons	Flops	Parameters
ADAM Global	75	88	66,282	66,453
SGD Global	32	19	77,384	25,947
SGD Local	31	11	74,133	24,807

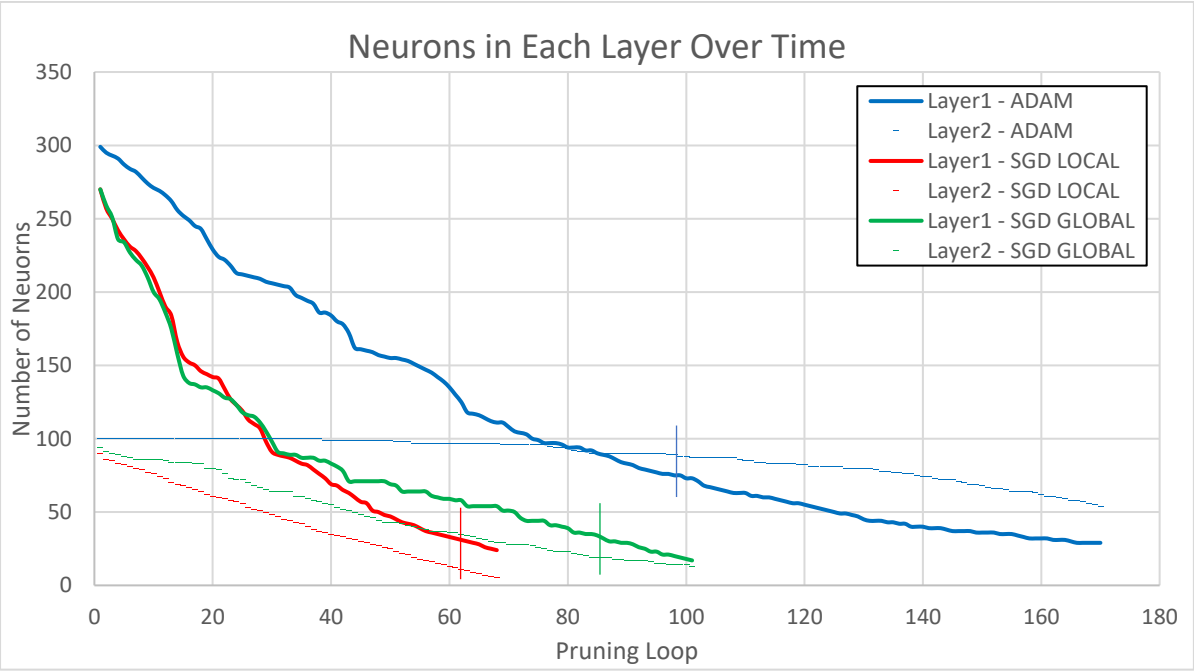


Figure 96:Number of neurons pruning in each layer over multiple pruning loops for the three different approaches; Adam Global, SGD Global and SGD local.

As Figure 97 shows, the main difference between the optimisers is the speed of pruning the second layer's neurons. When pruning using the ADAM optimiser, the 2nd layer isn't pruned much at all until the first layer has far fewer neurons. In stark contrast when SGD is used the layers are pruned uniformly, and at the selected model locations (indicated by the lines) there are still more neurons in the first layer than second.

7.6.1.2 Input Pruning

Input pruning is a technique used when the input to a neural network is highly sanitised and normalised, this is the case with MNIST as most numbers are centred in the middle of the image and are normalised in greyscale [83].

When this is the case, you can assume that a large proportion of the input data is redundant. For example, in the MNIST dataset the highly centred region of interest, shown in Figure 98, means that the periphery of the input may be ignored.

Tests were run with different pixel thresholds to see at which point each the model was impacted by this reduction in the number of input neurons. The training data was extracted, and each class was gathered and averaged on top of each other, this can be seen in Figure 98.

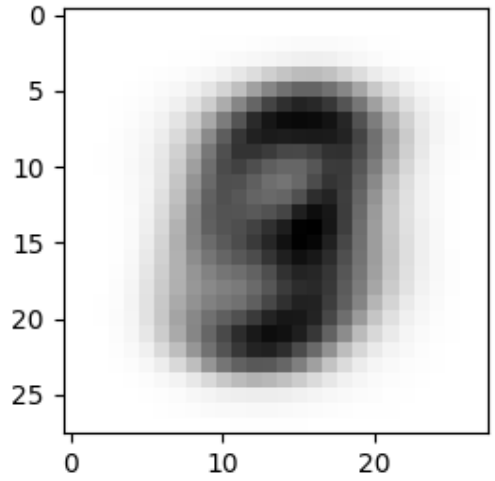


Figure 97: Total average output of the whole of the MNIST dataset.

This produced an average input, immediatley it's possible to identify regions that contain less data on average than the rest of the image. The next step was to provide a threshold to limit the input based on the averaged values. Figure 98 shows the averaged data over the whole dataset, Figure 99 shows how changing this threshold value changes the averaged image and the mask created for training.

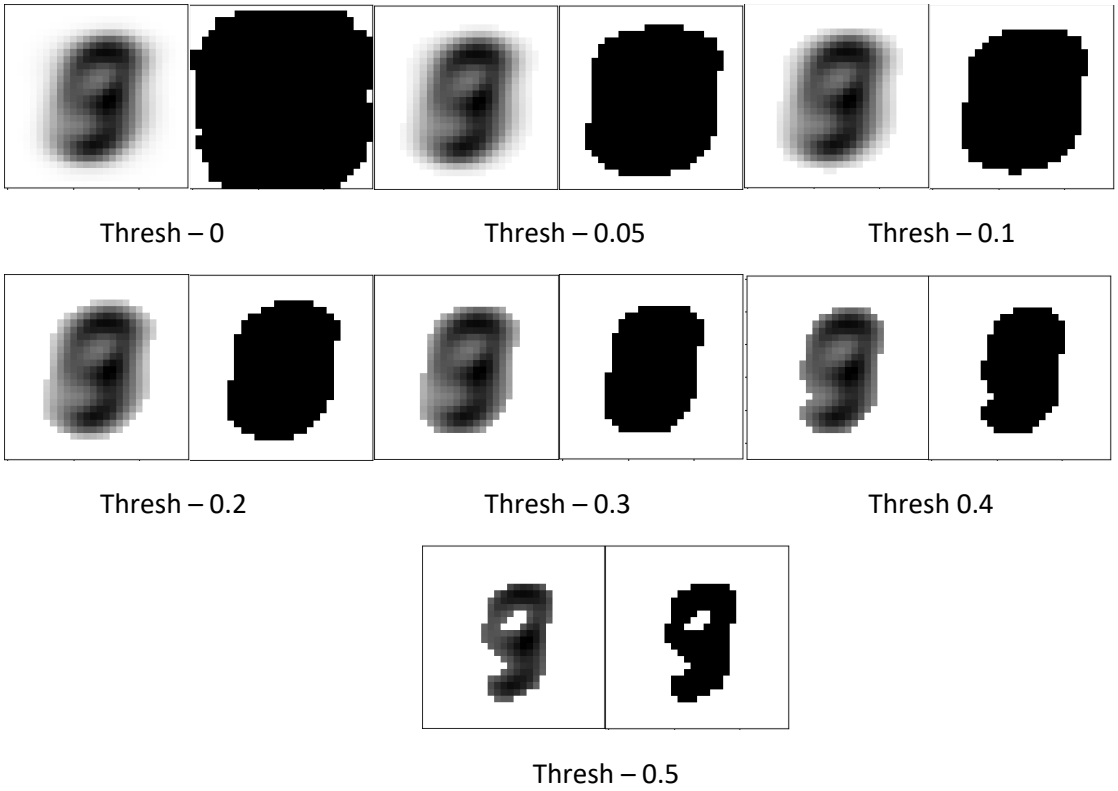


Figure 98: The image on the left in each example shows the averaged input that the network will train on, and the image on the right shows the mask that is applied. The black region is the area that will be used as an input for all numbers in the dataset. The text underneath indicates the threshold applied for that specific set of images.

Each threshold results in a different number of input neurons shown in table 38, interestingly a threshold of 0 already removes a significant number of pixels. This indicates there are regions with no pixel information in the whole of the training dataset. Each threshold has a different effect on the input and will result in a different accuracy from the trained network. Each threshold was trained from scratch using identical hyperparameters.

When training, a subset of the training data was extracted, 1000 examples of each class were gathered and averaged on top of each other to find the correct threshold mask. This was not done over the entire training set to keep a portion of the training data independent of the threshold data.

Table 38: Comparison of different threshold rates used and the effect that this has on the number of input pixels used for training.

Threshold Value	Original	0	0.05	0.1	0.2	0.3	0.4	0.5
Number of remaining pixels	784	673	396	343	285	237	198	139
Percentage of remaining pixels	100%	85%	50%	44%	36%	30%	25%	18%
Accuracy of model	98.3%	98.3%	98.3%	98.4%	98.1%	98.0%	97.8%	96.6%

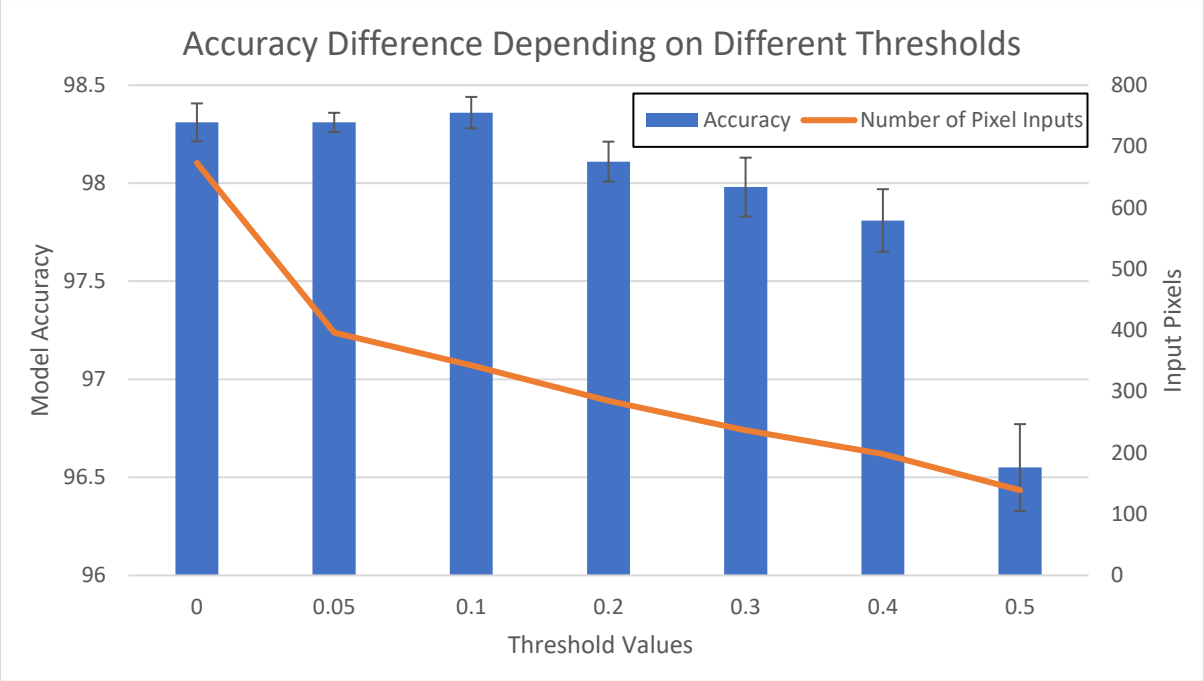


Figure 99: Detailed analysis on how varying the number of input neurons effects the accuracy achieved by LeNet-300 on MNIST.

The results in Figure 100 are interesting, not only because the number of inputs required for high accuracies on MNIST seem low (below 50%), but there is also an improvement in accuracy around the threshold of 0.1 over the baseline tests. This could be due to many things; however, it is most likely due to a specific style of digit or handwriting being overrepresented in the training set. By limiting the input, and hence some of the more unusual styles of written digits, this reduces confusion in the neural network and increases accuracy.

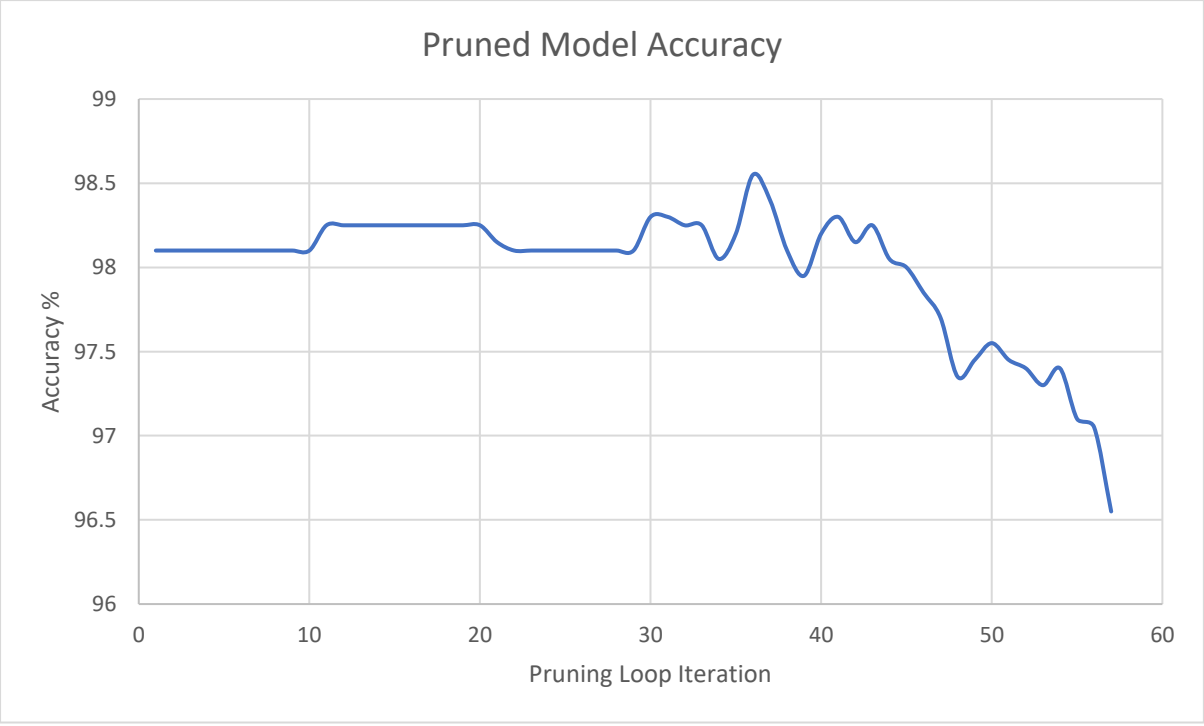


Figure 100: The pruning test was repeated, an input pruning threshold of 0.1 was used, using the SGD optimiser.

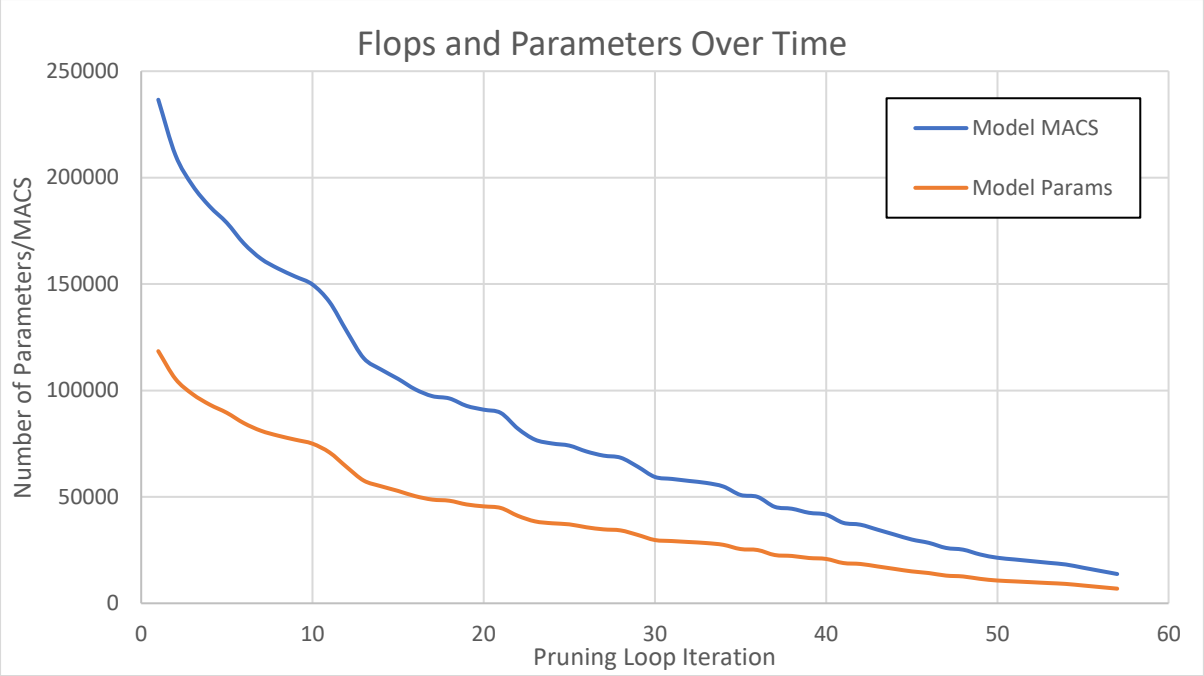


Figure 101: MACs and model parameters recorded over multiple pruning loops with an input threshold of 0.1 applied. Once again MACS are reported for readability.

Table 39 shows that the addition of input pruning, with the original technique, reduces the number of flops and parameters significantly. This brings our results more in line with other state of the art methods. It makes sense that most sparse only methods, are beaten by methods that employ both pruning methods, with regards to the number of parameters and FLOPs removed. A reasonable assumption to make is that the simple stacking of structured, and sparse pruning, squeezes as much information from the model as possible.

*Table 39: Comparison to SOTA methods pruning LeNet-300 with an input trim of 0.1 and 0.3 with parameter counting applied, best result in **bold** second best underlined.*

Method	Parameters	Flops	Error	Pruning Type
Ours SGD Global – Inp Trim - 0.3	11,068	**11,055**	2.01% (Δ -0.42%)	Structured
Ours SGD Global – Inp Trim - 0.1 +	5,025	12,279	1.89% (Δ- 0.3%)	Both
Ours SGD Global – Inp Trim - 0.1	12,286	12,279	1.89% (Δ -0.3%)	Structured
Ours SGD Local – Inp Trim - 0.1	14,994	14,982	1.99% (Δ -0.4%)	Structured
Auto Prune	-	24,346	1.82% (Δ- 0.22%)	Structured
Corset	26,000	-	2.03% (Δ- 0.13%)	Structured
MIXP	**3,199**	19,982	- % (Δ -0.3%)	Semi
NeST	7,844	14,900	**1.29%** (Δ- 0.31%)	Semi
GrowPrune	12,200	-	1.4% (Δ-)	Sparse
EffNN	22,000	-	1.59% (Δ- 0.05%)	Sparse
DiffEvo	10,000	-	2.07% (Δ- 0.19%)	Sparse
PartSwarm	44,369	-	2.2% **(Δ+ 0.06%)**	Sparse
NoiseOut	11,225	-	3.05% (Δ-)	Sparse

With regards to accuracy, there is an important issue across all pruning papers, in that there is no agreed accuracy to achieve. Most papers do not even start at the same baseline model performance. This is clearly shown in Table 39, and in fact, in the MIXP paper the accuracy of the model is never given, only the degradation due to the applied pruning is provided.

In our experiments, even before pruning was applied the maximum accuracy achieved was 98.4%, NeST claims an accuracy of 98.71% from their pruned model. Whilst 0.3% might not seem significant, these last few percent are extremely hard to achieve, whilst the paper does provide detail of the hyperparameters regarding training LE-NET-300 it does not provide any GitHub implementations. Nor any post-pruned models, replicating the hyperparameters from the paper exactly could not replicate the accuracy claims, even on a non-pruned model.

Luckily when pruning more complex models like VGG and Res-Net, pre-trained models exist, and accuracy metrics are well established, however as discussed later this issue is largely unsolved and prevalent in the pruning community.

Ours SGD Global – Inp Trim - 0.1 + shows the best result from our experiments. The + indicates the result from both structurally pruning and sparsely pruning LE-NET-300.

7.6.1.3 Confusion Matrix

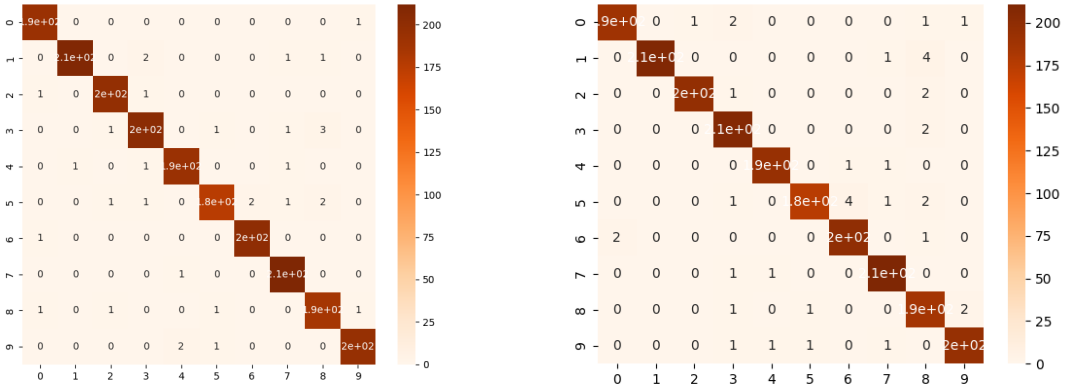


Figure 102: The confusion matrix on the left shows the un-pruned LE-NET-300 network and the confusion matrix on the right shows the confusion matrix of the pruned model highlighted in green in the table 39.

Figure 103 shows that the ability of the network to correctly identify the number 8 to be vastly decreased from the original model, and interestingly other numbers seem to have increased in the number of correct predictions. This is most likely because 8 is a digit that contains the constitute parts of many other digits. 9 6 3 and 0 all share curves and lines in very similar places to the digit 8, whilst the model may have previously used other input information as a process of elimination, to combat this, it can no longer do so. This theory will be further explored in the following section on saliency maps.

7.6.1.4 Ablation Studies

The model's architecture changes over time, but the final pruned model has 343 input neurons, 33 neurons in the first hidden layer and 21 in the second. A model was trained from scratch using the same number of hidden layer neurons. This was to see if a model with the same accuracy as the pruned model, could be achieved without the iterative pruning process.

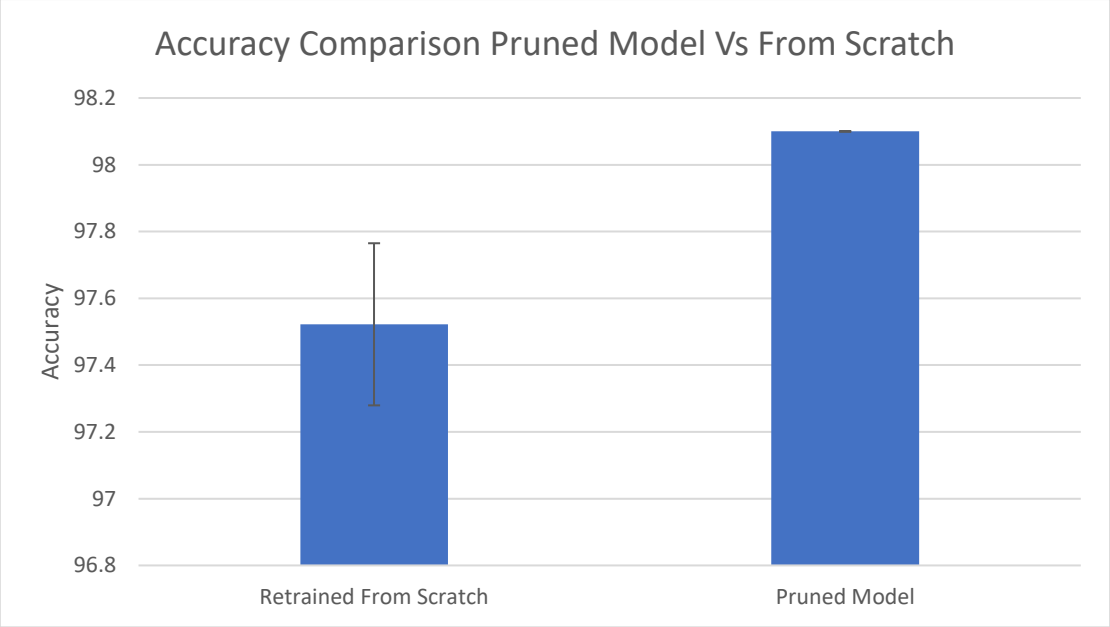


Figure 103: Testing to identify if a model of the same architecture as the pruned model can achieve the same accuracy being trained from scratch.

The model was retrained 10 times, and an average of the results were taken, Figure 104 clearly shows that the model performance of the pruned model is vastly higher. Even when considering the error bars, the highest performing model is still the pruned model. This shows that the pruning process has managed to preserve the knowledge learnt by the initially trained model and transfer it to the smaller pruned model.

Table 40: Results proving the efficacy of the sparsity pruning step.

Method	Parameters	Flops	Error	Pruning Type
SGD Global – Sparsely Pruned	12,286	12,279	1.89% (Δ 0.3%)	Structured
SGD Global – W/O Sparsity Pruning	19,506	19,479	1.95% (Δ 0.4%)	Structured

Additionally, it was imperative to prove the utility of the sparsity pruning within the proposed method. To achieve this the experiment was re-run with the removal of the sparsity pruning steps. The results in table 40 and Figure 105 show that without the sparsity pruning step, the model cannot achieve the same performance.

If we inspect how the model layers were pruned over time, there is also a big difference in the rate of pruning of both hidden layers, when sparsity pruning is not used. This shows that sparsity pruning step is aiding the structural pruning step by zeroing out weights.

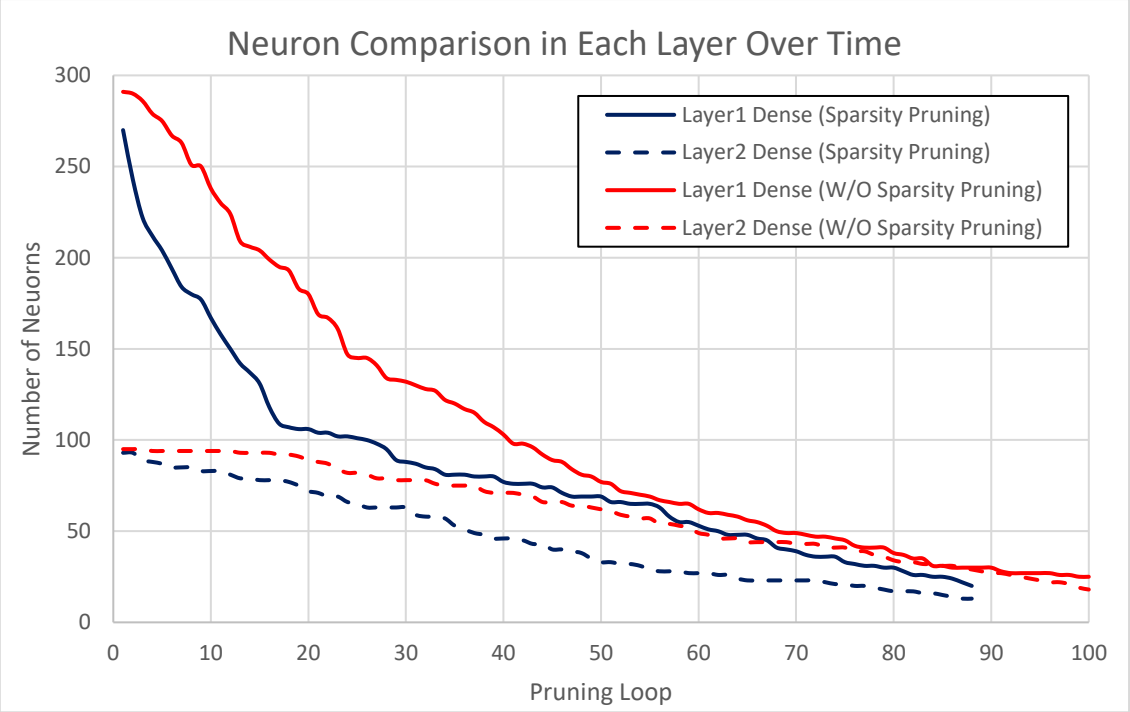


Figure 104: Pruning profile using sparsity pruning (blue line) and not (red line)

The accuracy graphs in Figure 106, show that after the initial 40 pruning loops the accuracy with sparsity pruning outperforms the accuracy without sparsity pruning (from loop 40-85). This the addition of these two techniques better retains the accuracy of a model once structurally pruned.

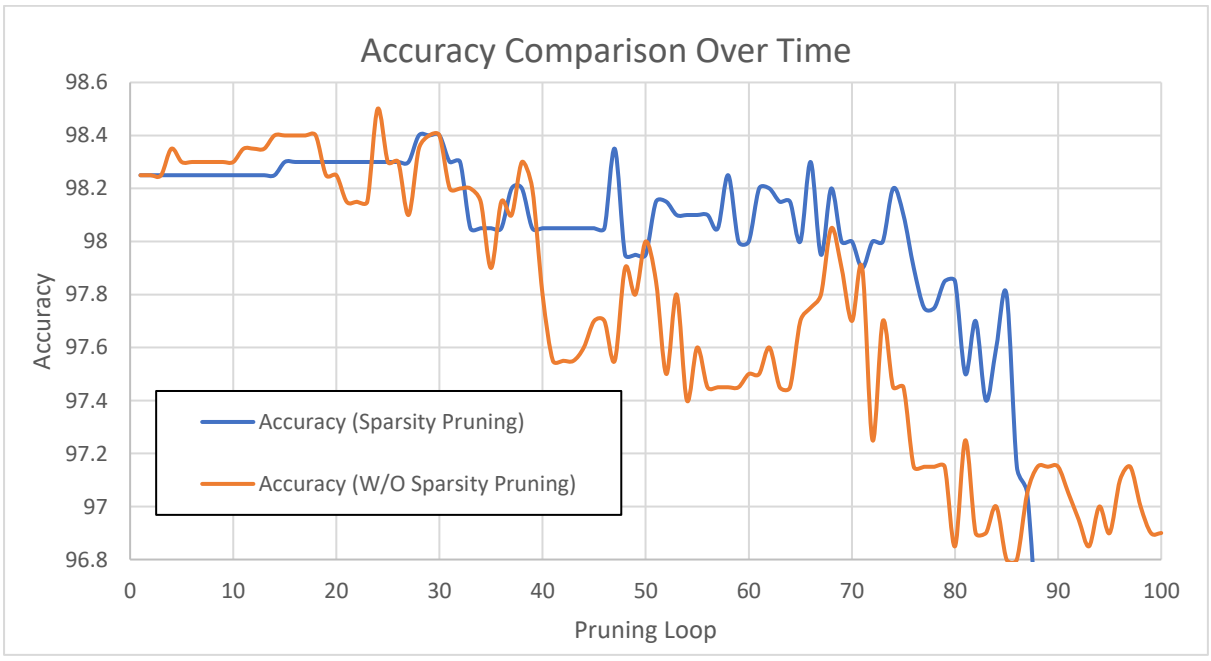


Figure 105: Accuracy pruning profile of the pruning method using sparsity pruning (blue line) and not (orange line)

Another important consideration is the number of retraining steps required by both techniques. To get to the pruned model the sparsity pruning technique must be fine-tuned 29 times, the technique without sparsity pruning must be fine-tuned 55 times. This re-enforces the evidence for the efficacy of the sparsity pruning technique. Additionally, the fine-tuning occurs much later in the process

when applying sparsity pruning. This means that the sparsity pruning technique can be initially applied and then no fine-tuning needs be applied, this is not possible without sparsity pruning.

7.6.1.5 Resource Usage

To test the inference time of the pruned and baseline model, each model was tested 20 times on a test set of 250,000 image samples with a batch size of 16,384. This was done to maximise the utilisation of the GPU. This model is so small that on small batch sizes the CPU can outperform the GPU in terms of inference time. This highlights the parallel nature of GPUs. All tests were performed on a RTX 2080 SUPER GPU and an Intel(R) Core(TM) i7-9700K CPU.

7.6.1.5.1 Speedup GPU

Table 41: Model speedup on a GPU.

Model	Time (s)	Error
Baseline	0.45	1.59%
Pruned	0.23	1.89%

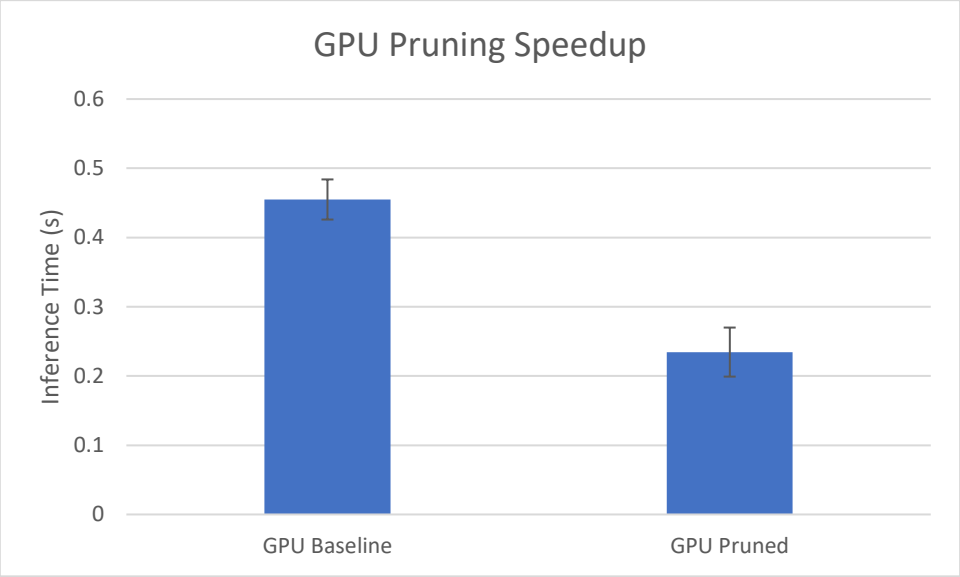


Figure 106: Model speedup on a GPU showing a speedup of 48.45% was achieved.

7.6.1.5.2 Speedup CPU

Table 42: Model speedup on a CPU.

Model	Time (s)	Error
Baseline	0.64	1.59%
Pruned	0.25	1.89%

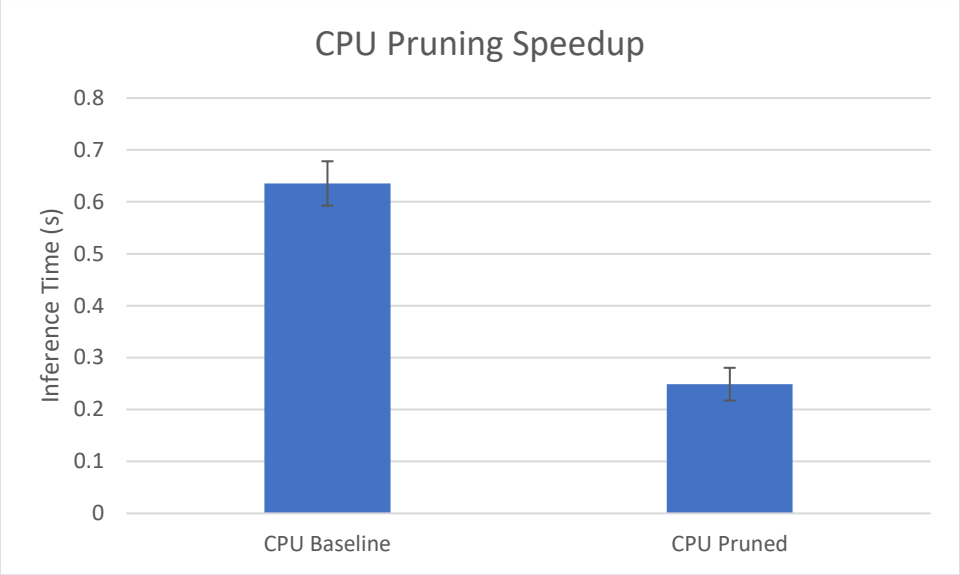


Figure 107: Model speedup on a CPU showing a speedup of 60.84% was achieved.

7.6.1.5.3 Memory Use

The size of these models is so small the GPU memory usage was not affected by the pruning, however the size on disk was, the model was reduced from 1,082 KB to 68 KB for the SGD Global – Inp Trim - 0.1 + model.

7.6.1.6 Arch evolution

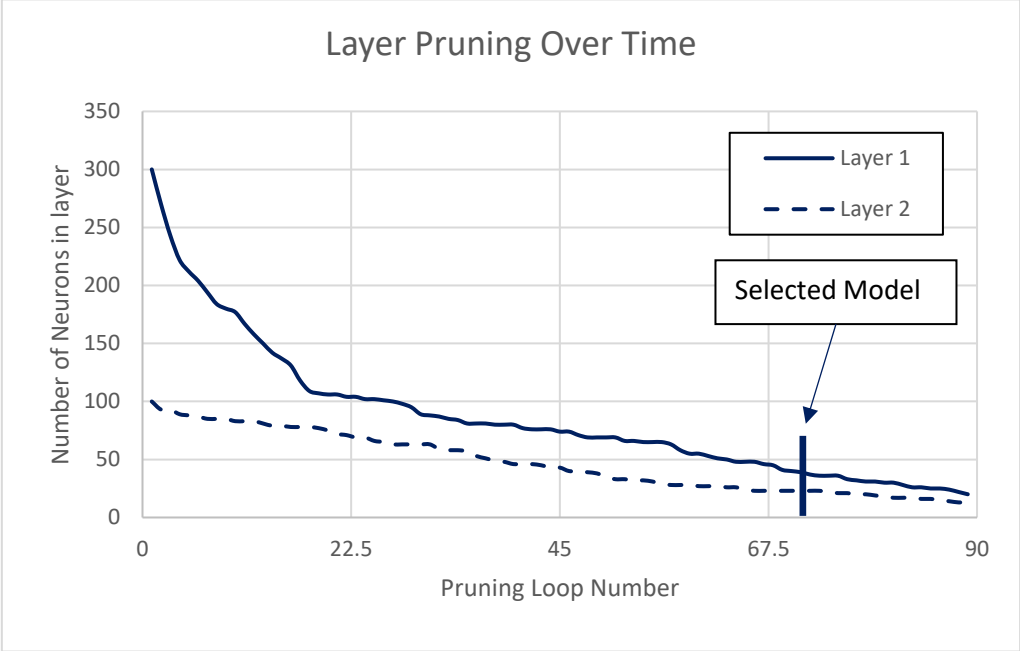


Figure 108: Model architecture evolution over multiple pruning loops showing the number of neurons in each layer.

This shows how the architecture of the model changes over time, if you match this to the parameter and FLOPs plots from earlier you can see they are all directly correlated for LeNET-300. A removal of a neuron results in a similar drop in FLOPs and parameters, as shown in figure 110 this also results in a reduction of inference time.

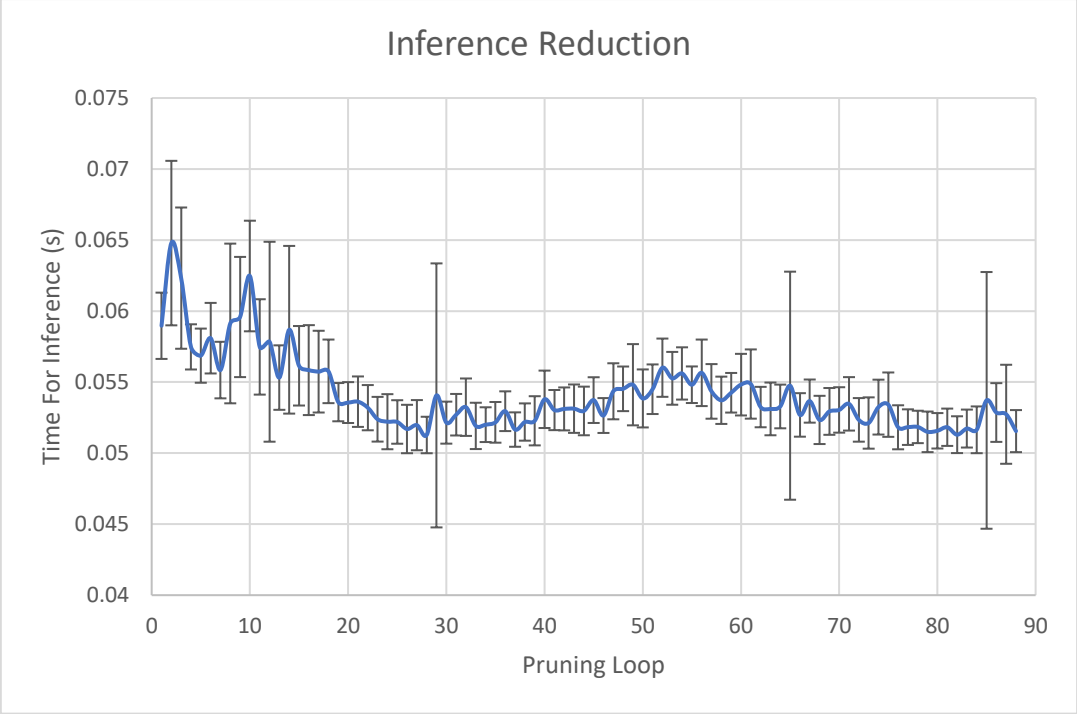


Figure 109: Inference reduction due to pruning recorded for each pruned model.

Inference testing for Figure 110 was performed on the CPU, with such small models it is hard to control the variability in inference time when testing on the GPU as the GPU is underutilised in such scenarios. It is hard from these results to draw a conclusion as to whether the parameters or FLOPs play more of a role in inference time, as they are directly correlated in this model.

7.6.1.7 Conclusion

Overall, these results show the best results for reduction in FLOPs when compared to other SOTA methods, importantly this reduction is also realised in the code and isn't a theoretical increase. Our model performs well when compared to SOTA pruning on parameters when measured in the same way as the other techniques (non-zero weight values). However, our method underperforms when compared MIXP, this could be due to the optimisation differences between the two methods. Our method deliberately targets FLOPs by identifying neuron and filter utilisation, MIXP focuses on weight utilisation and therefore inherently prunes parameters more than FLOPs.

These first set of results clearly demonstrate the importance of the sparsity pruning step, allowing the model to have a higher accuracy for longer whilst pruning. Additionally, these results show that the model structure developed cannot be trained from scratch to the same accuracy as the one "discovered" through the pruning process.

It is important to note, the sparsity pruning step is just a means to an end, that being non-zero weight parameters being reduced. This can be achieved through any of the sparsity pruning methods listed in this section and could potentially improve results of the method overall. However, this would add too much complexity and too many methodologies to test at this time, therefore the standard magnitude-based sparsity method from TensorFlow will remain as the sparsity pruning step method for WAP.

7.6.2 LeNet-5 on MNIST

7.6.2.1 Comparison to Similar Techniques

After testing the model on in the previous section, it was identified that the best set of results were achieved when using SGD with global focus. However, for completeness and because this network is convolutional as well as fully connected, we included the results for both global and local pruning.

Table 43: SOTA pruning methods using the model LeNet-5 and dataset MNIST

Method	Parameters	Flops	Error	Pruning Type
Original Model	430,000	2.29M	0.8 %	None
Ours SGD Global +	3,351	78,170	1 % (Δ -0.2 %)	Both
Ours SGD Global	16,649	78,170	0.9 % (Δ -0.1 %)	Structured
Ours SGD Local	27,782	**66,087**	0.9 % (Δ -0.1 %)	Structured
Ours SGD Global (HA)	20,315	81,827	0.8 % **(Δ 0%)**	Structured
MIXP	**2,580**	73,280	- % (Δ -0.3 %)	Semi
NeSt	5,772	105,000*	0.77 % (Δ +0.3 %*)	Semi*
HRel [91]	8,686	80,000	1.22 % (Δ -0.53%)	Structured
AUTO	**2,580**	160,300	0.80 % (Δ -0.2%)	Structured
GSM [90]	3,440	135,110	0.78% (Δ -0.01%)	Structured
PartSwam	8,504	-	0.91 % (Δ -0.03%)	Sparsity
GrowPrune	7,900	-	0.83 % (Δ -)	Sparsity
EffNN	36,000	-	**0.77 %** (Δ -0.03%)	Sparsity
DiffEvo	3,444	-	1.41 % (Δ -0.46%)	Sparsity

Some results have been highlighted with an asterisk, MIXP is highlighted due to the discrepancy with its accuracy. However, Nest's FLOPs result is also highlighted. This is because the paper uses so called "partial area convolutions" to reduce FLOPS. These are convolutions that use a mask to reduce the amount of input to a given filter in a neural network. They claim in the paper that this causes a reduction in FLOPs, however the only implementation that exists employs a mask of zeros to block the input from the previous activation map [84]. This increases the number of operations required for a conv operation, it is a misconception by the authors (and others in the field) that multiplying by 0 somehow requires no computation, this is not the case.

This also begins to show the difference in approaches, overall, I believe our best result is highlighted in green in Table 43. This result is the result before sparsity pruning is applied, I think it is a good middle ground showing the 2nd best results after MIXP in FLOPS but higher accuracy.

The result in blue shows the model selected by highest accuracy, this is comparable to other results that lose no accuracy when pruning. However, our result out-performs all 0% Δ models on FLOPs metrics, additionally the Local pruning model performs even better when reducing FLOPs.

Our results look bad when compared using number of parameters however this is expected, our method does not contain a complex sparsity pruning process, it is simply doing magnitude weight-based pruning. All the sparsity pruning techniques could be applied to the models marked in red, blue, and green here and achieve significant reductions in the number of parameters. The SGD Global + demonstrates this and is a sparsely pruned version of the network highlighted in green.

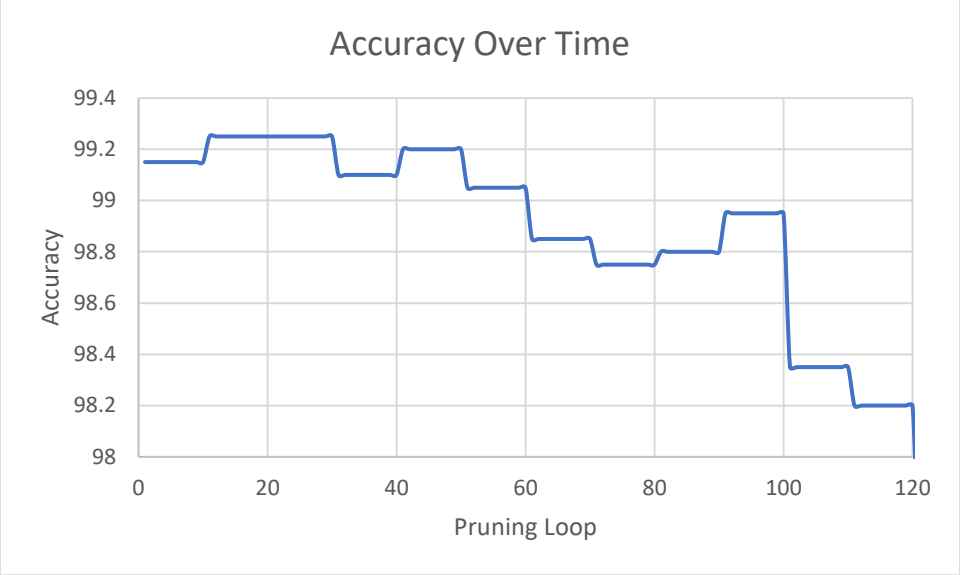


Figure 110: Accuracy of pruned model at all points in the pruning process using the model LeNet-300 and dataset MNIST.

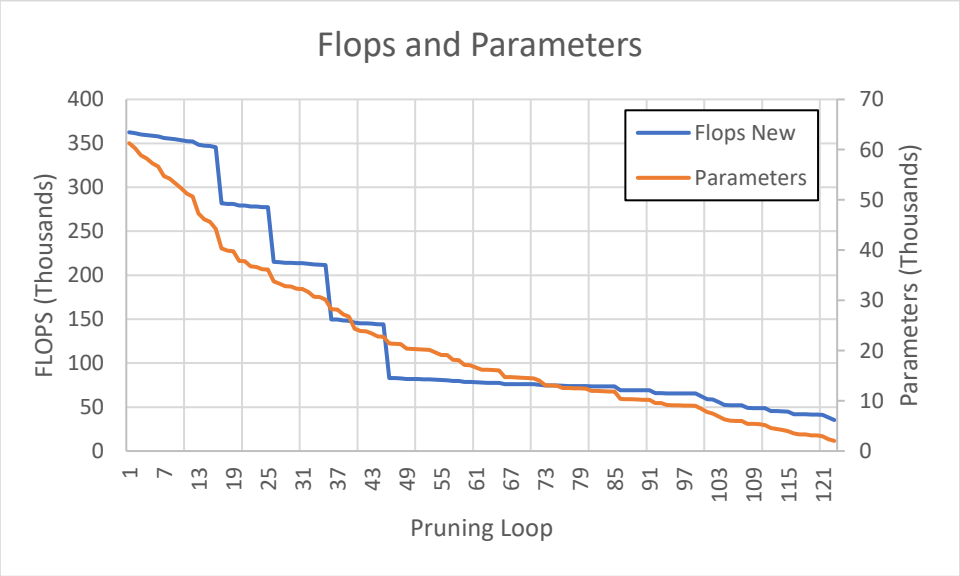


Figure 111: FLOPs and parameters of each pruned model at each pruning loop using the model LeNet-300 and dataset MNIST.

This shows a stark difference in relationship to the previous experiment on LE-NET300, parameters and FLOPs are not directly proportional, this can be observed by the distinctive staircase like drops in FLOPs shown by the blue line.

7.6.2.2 Confusion Matrix

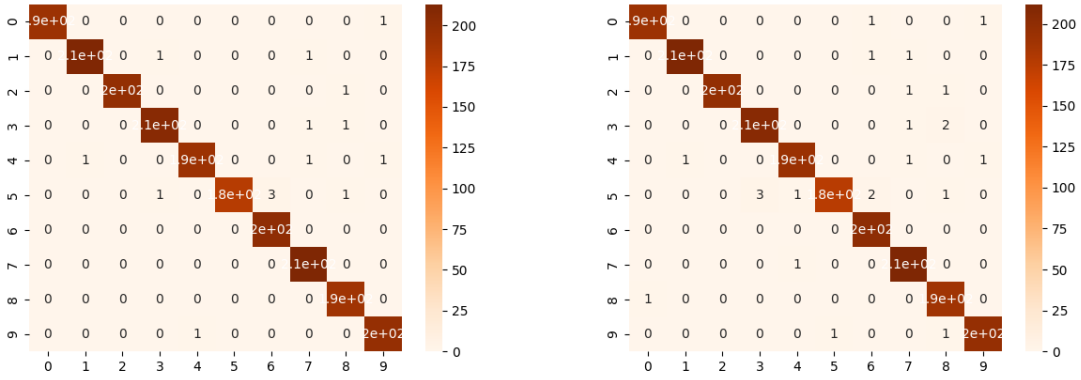


Figure 112: The confusion matrix on the left shows the un-pruned LE-NET-5 network and the confusion matrix on the right shows the confusion matrix of the pruned model highlighted in green in the table in the previous section.

As the reduction in accuracy is lower than that of the previous experiment the confusion matrices are very similar in their layout shown in Figure 113.

Recognition of 7,6 and 8 have been most affected by the change, the only other significant change is 5's being miss-classified as 3's. This is something that the fully connected network did not have issues with, this is probably because there is no spatial information when training with a fully connected network, as all inputs are flattened. All the pixels could be randomly jumbled into different locations and if this is kept the same for training and inference the accuracy of the model should be the same as an unjumbled input. However, a convolutional network takes advantage of this spatial information and hence the accuracy is higher. This results in the most "confusing" digits are very different for LE-NET5.

7.6.2.3 Resource Usage

For the inference tests the model was timed predicting 50,000 samples at a batch size of 4096 this was repeated 20 times using a GTX2080 Super and an Intel Core i7-9700K CPU @ 3.60GHz.

7.6.2.3.1 Speedup GPU

Table 44: Model speedup on a GPU

Model	Time (s)	Error
Baseline	0.26	0.8%
Pruned	0.11	0.9%

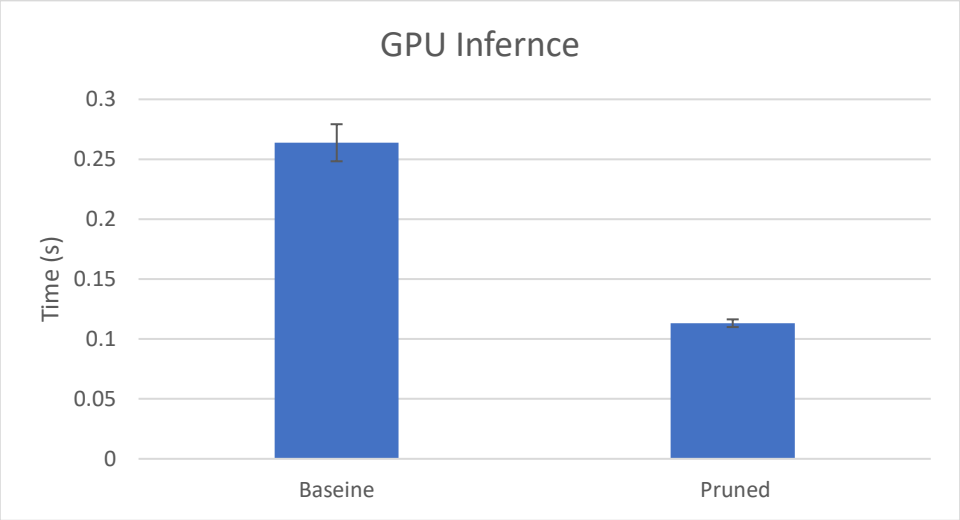


Figure 113: Model speedup on a GPU showing a speedup of 57.08% was achieved.

7.6.2.3.2 Speedup CPU

Table 45: Model speedup on a CPU

Model	Time (s)	Error
Baseline	0.37	0.8%
Pruned	0.26	0.9%

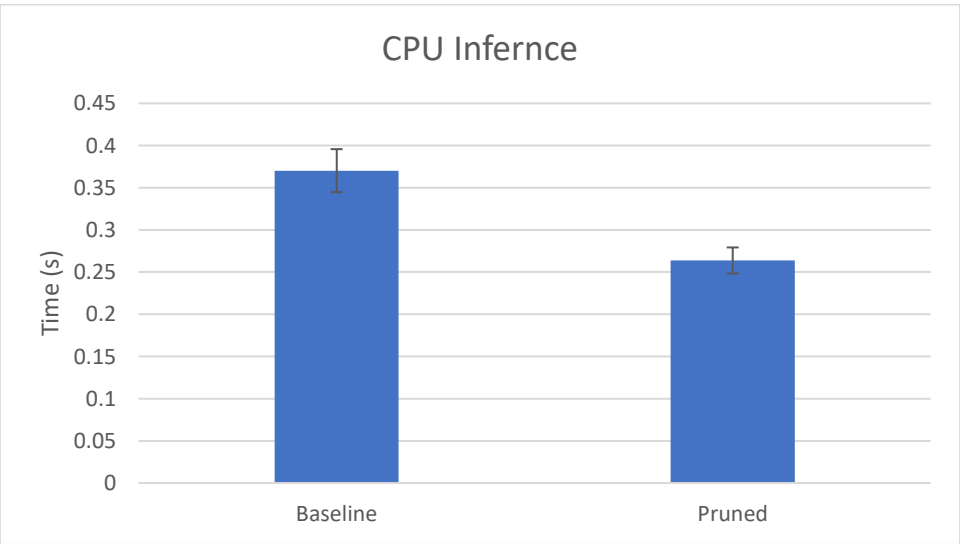


Figure 114: Model speedup on a CPU showing a speedup of 28.76% was achieved.

7.6.2.3.3 Memory Use

The GPU VRAM usage was reduced from the baseline at 7951MB to 6662MB for the pruned model meaning a reduction of 16.21%.

Additionally, the size on disk was reduced from the baseline 275KB to 101KB for the pruned model, a reduction of 63.27%.

7.6.2.4 Arch evolution

Here it is possible to see how the model layers are pruned over time, however, Figure 116 is misleading. Some layers start with more neurons and filters (units for short) than others, Layer 1 Conv only contains 6 filters but is reduced to 1 by pruning loop 48, however it looks like its barely pruned in this visualisation.

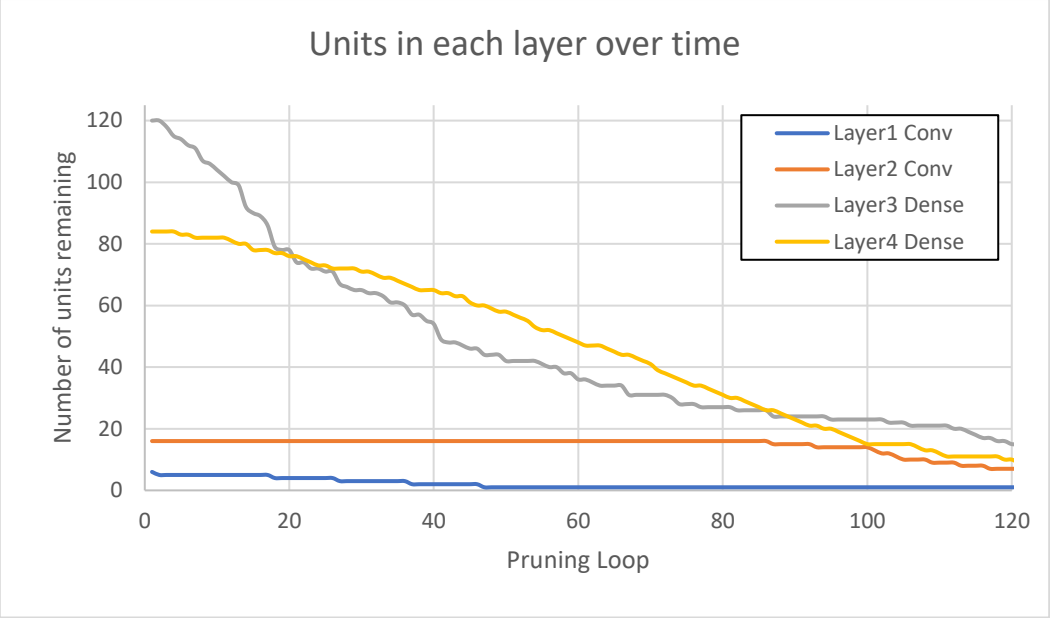


Figure 115: Number of neurons remaining in each layer over multiple pruning loops.

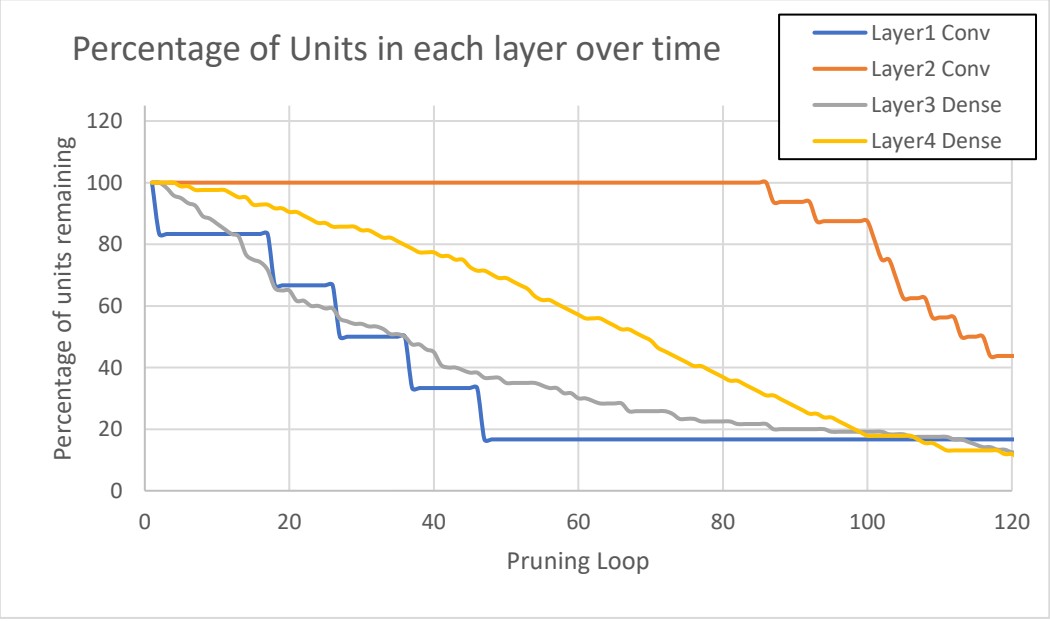


Figure 116: In this plot the remaining filters were plotted as a percentage of their original amount, this gives a better indication of what is being pruned and the magnitude of the pruning applied relative to the individual layer.

So, to best represent the pruning of a layer in comparison to its original state, units in any given layer will be represented as a percentage of their original number of units. This is an important change, looking back at Figure 112 you can see that the significant stepping down in FLOPs early in pruning is due to the pruning of the Layer 1 Conv when considering Figure 117. This behaviour is not easily seen when looking at Figure 116. Additionally Figure 117 shows the lack of pruning convolutional layers from loop 48 to 86 shown with a straight line, and this is reflected in Figure 112 with a straight line for FLOPs. It seems that this confirms that convolutional layers are heavy in FLOPs whilst Dense layers are heavy in parameters.

Interestingly this also shows that after the model prunes layer one to contain only one filter (at loop 45), the accuracy of the model can never recover to an error rate of 0.8%. For this reason, the test was re-run with a minimum filter pruning size set to two, the results can be seen in the ablation study section.

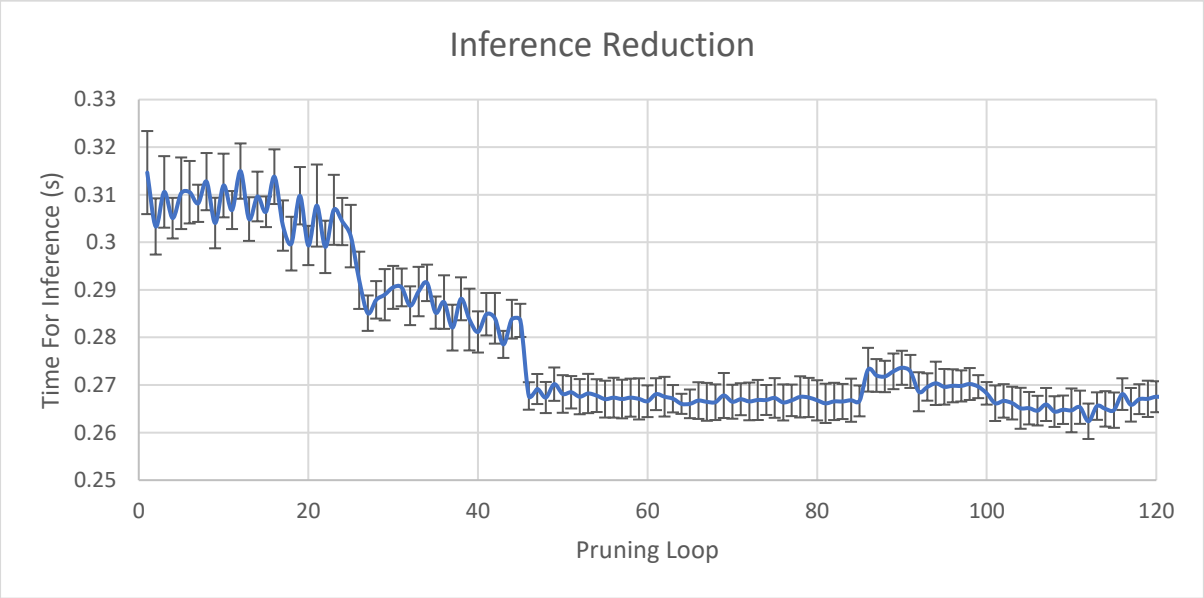


Figure 117: Inference time for each pruned model at the end of the respective pruning loop.

The test of inference time Figure 118 shows a strong correlation to the FLOPs metric as expected. There is also a small increase in inference time towards the end of pruning, but as shown by the error bars this is within the error margin and can be considered noise.

7.6.2.5 Ablation Studies

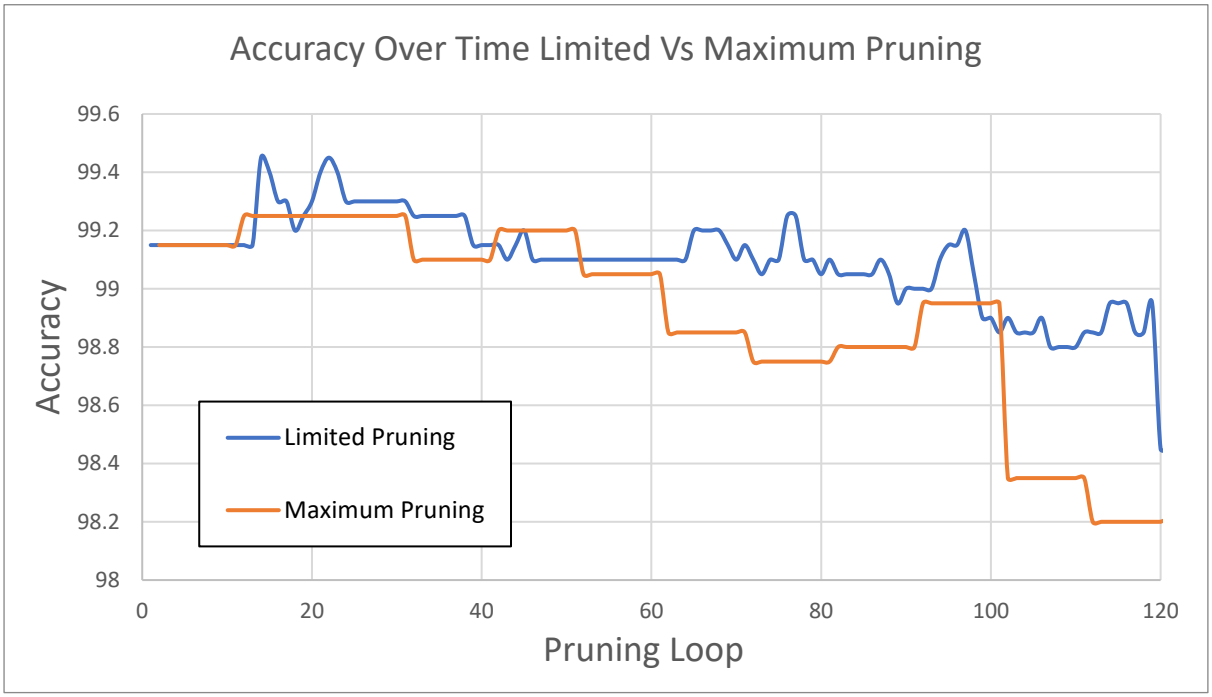


Figure 118: Comparing pruning a model that has a hard limit on the number of filters that can be removed from the convolutional layers (blue) against an unrestricted model (orange)

*Table 46: Performance increase due to limiting the number of filters that can be pruned by the pruning process, best result in **bold** second best underlined.*

Method	Parameters	Flops	Error	Pruning Type
Original Model	430,000	2.29M	0.8 %	None
Ours SGD Global +	3,351	78,170	1 % (Δ -0.2 %)	Both
Ours SGD Global	16,649	78,170	0.9 % (Δ -0.1 %)	Structured
Ours SGD Local	27,782	66,087	0.9 % (Δ -0.1 %)	Structured
Ours SGD Global (HA)	20,315	81,827	0.8 % **(Δ 0%)**	Structured
Ours SGD Global LIM	**1719**	**57,587**	0.9 % (Δ -0.1 %)	Structured
Ours SGD Global LIM (HA)	3923	69,878	0.8 % (Δ 0%)	Structured
MIXP	2,580	73,280	-% Δ -0.3 %	Both

Figure 119 shows that by limiting the number of prune-able channels to a minimum of two, the accuracy of the model is preserved further into the pruning process. Specifically, you can observe this in Figure 119 from loop number 50 onwards, where the limited pruning method is always more accurate. Giving our two final and best results. However, the approach of hard limiting prune-able layers was not updated into the algorithm, for this simple model the key layer to limit pruning was easy to spot. However, for this to be implemented into the algorithm a general case needs to be discovered. The final model configuration is listed below for SGD Global LIM.

Table 47: Final model configuration for SGD Global LIM.

Layer	Number of units
Conv - Layer 1	2
Conv - Layer 2	5
Dense - Layer 3	21
Dense - Layer 4	30

The model was once again trained from scratch to test the efficacy of the pruning method. Again, Figure 120 shows that the accuracy achieved via pruning cannot be achieved via training alone.

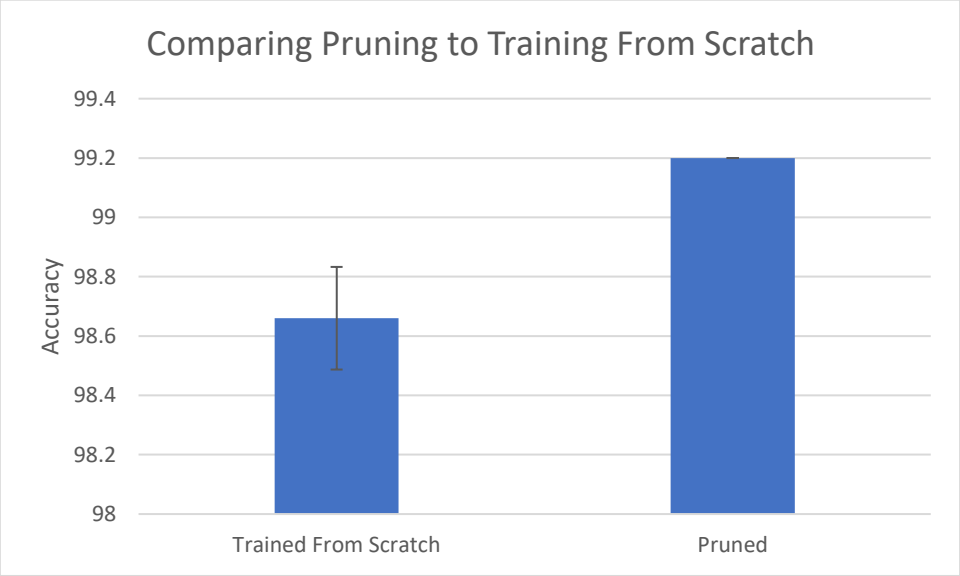


Figure 119: Testing to identify if a model of the same architecture as the pruned model can achieve the same accuracy being trained from scratch.

7.6.2.6 Conclusion

These results are interesting because at first glance the MNIST results look to have been beaten by most other methods, however this is where the intricacies of NN pruning come into play.

It is true that the default method underperforms slightly on FLOPs and majorly on parameters. But our ablation studies show that by limiting the number of filters that can be pruned in specific layers the results can be vastly improved. This is shown in table 46, where the GLOBAL LIM results in green, beat all SOTA methods when compared on both FLOPs and parameters. However, I think this result to be misleading, because limiting the number of filters to be pruned was only implemented after spotting a significant layer. And after analysing a similar pruning method (HREL), which also hard limits the number of filters pruned in each layer.

However, many other methods have a similar approach and give each layer different “weightings” before pruning has been applied to guide how much to prune one layer in comparison to others. For example, in the MIXP paper this value is obtained through “random search”. This is a troubling direction for a pruning technique to take, as it assumes that you know the pruning profile of the dataset and model before pruning is applied. This kind of search may be possible for smaller models and datasets, like LeNET-5 and MNIST, but this search method for hyperparameters becomes more difficult with larger models. This is due to the increase in layer size and the increase in training time for larger datasets like ImageNet.

As the reasoning for this is not given to a satisfactory level in the literature; I would suggest that the reasoning for these limits could be due to the parameter to FLOPs disparity in fully connected layers when compared to convolutional layers. This does make an argument for limiting pruning of convolutional layers when they are vastly outnumbered by other layers in a network, and vice versa.

Overall, the LIM results in the ablation section perform better than all other SOTA methods. However, the process of limiting the number of filters pruned in a specific layer needs to be refined in future implementations of the pruning method before fully implemented.

7.6.3 LeNet-5 on Fashion-MNIST

For this set of results a comparison to similar techniques was not shown as it is not normally a test run by other researchers, however I thought it was important.

This experiment shows what happens to pruning when the dataset has been replaced by an identical dataset in terms of input size and number of examples. However, the classification task is considered "harder". The idea is to learn how more complex datasets affect pruning neural networks.

For these experiments identical pruning and hyperparameters were used as the SGD Global LIM from the last section.

7.6.3.1 Comparison to Unpruned Model

Table 48: Pruning results for the model LeNet-5 on dataset Fashion MNIST

Method	Parameters	Flops	Error	Pruning Type
Original Model	430,000	2.29M	9.9 %	None
Ours SGD Global LIM	18,133	119,414	10.7 % (Δ -0.8 %)	Structural
Ours SGD Global LIM (HA)	29,755	151,216	10.0 % (Δ -0.1 %)	Structural

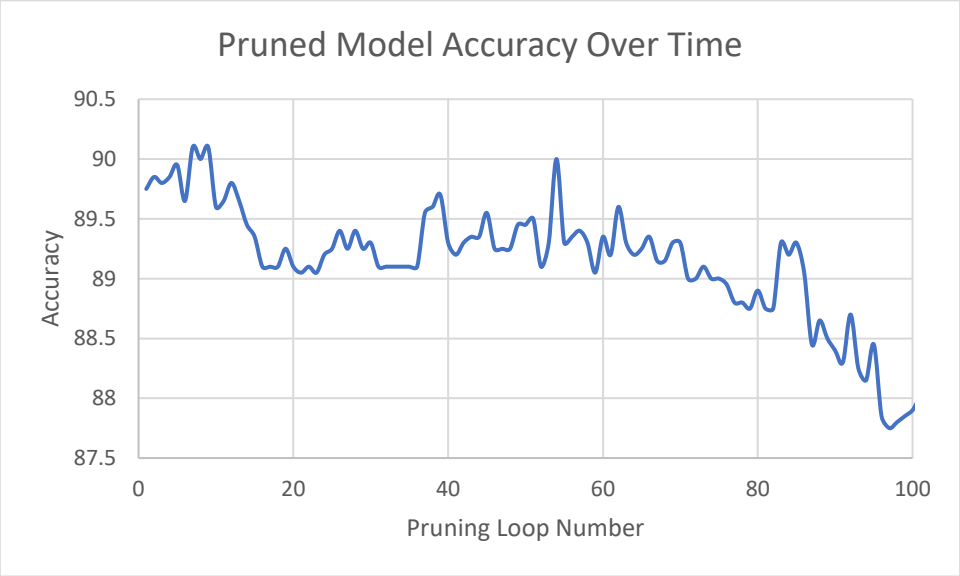


Figure 120: Accuracy of pruned model at all points in the pruning process.

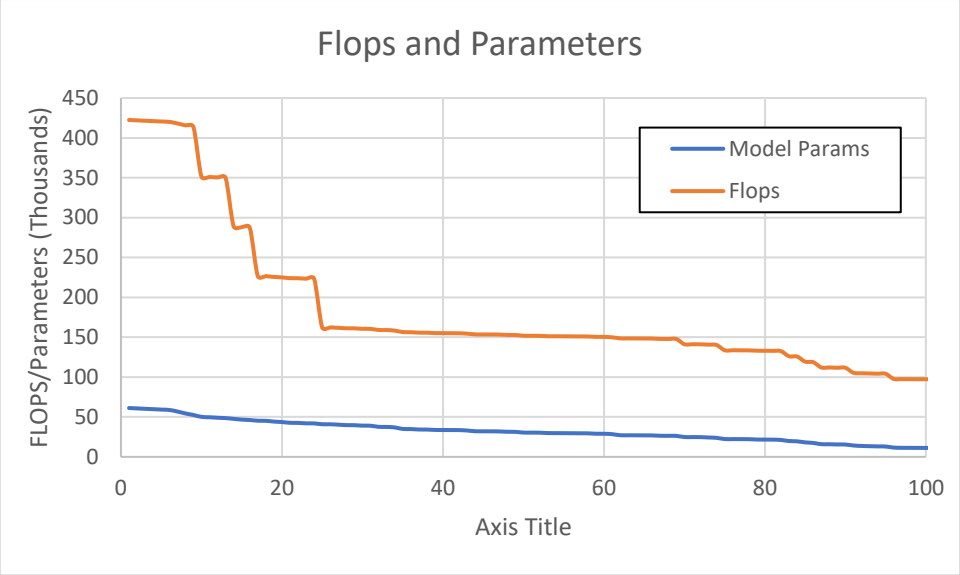


Figure 121: FLOPs and parameters of each pruned model at each pruning loop.

The first thing of note is that this dataset is much more complex than MNIST-digits. The maximum accuracy reached by the neural network before pruning was 90.1% shown in Table 48. This is a whole 9% less than the previous MNIST tests, additionally the number of FLOPs and parameters pruned is significantly less than the tests on MNIST-Digits. Here the maximum number of pruned FLOPs was 119,414 compared to the previous section that reached 57,587 (more than half).

7.6.3.2 Confusion Matrix

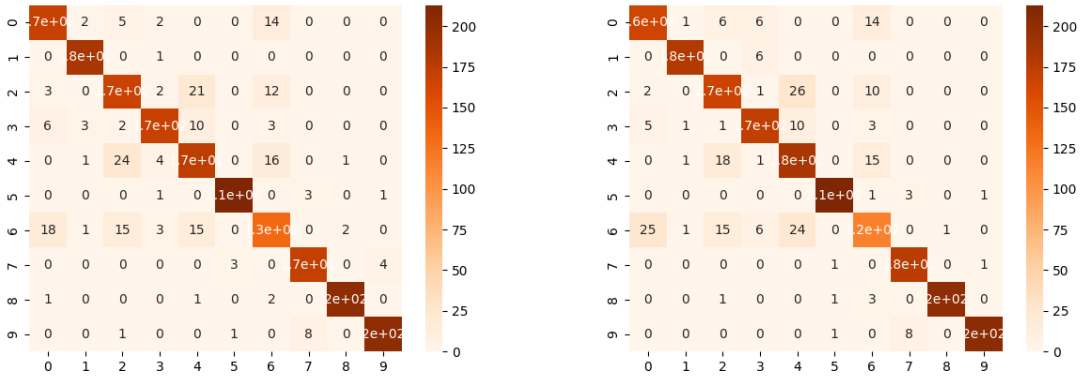


Figure 122: The confusion matrix on the left shows the un-pruned LE-NET-5 network and the confusion matrix on the right shows the confusion matrix of the pruned model highlighted in green in the table in the previous section.

Once again in Figure 123 it can be observed that it's not as simple as the neural network getting generally worse across the board. For instance, class 4 being identified as class 2 reduces from 24 to 18. However, there are some classes that get worse everywhere, class 6 is a good example of this, and leads to the following issue when pruning neural networks.

Because the pruning process is tuned for highest accuracy there is a "loophole" that the neural network can take, and that is essentially by making a "dump class". Here the accuracy of the system is below 90%, the dataset contains 10 classes meaning that by focusing on 9 of the classes and tuning to them, and ignoring the final class, you can achieve an accuracy of 90% whilst miss identifying a whole class.

More investigative tests will be run in the final chapter of the thesis to test the internal understanding in pruned neural networks. However initial signs of this can be seen here in class 6. This might not seem like an issue when dealing with a dataset containing 10 classes, but this issue becomes much more important when dealing with datasets of 1000's of classes. For these larger datasets a whole unlearnt class would only impact the model accuracy by 0.1%.

7.6.3.3 Arch evolution

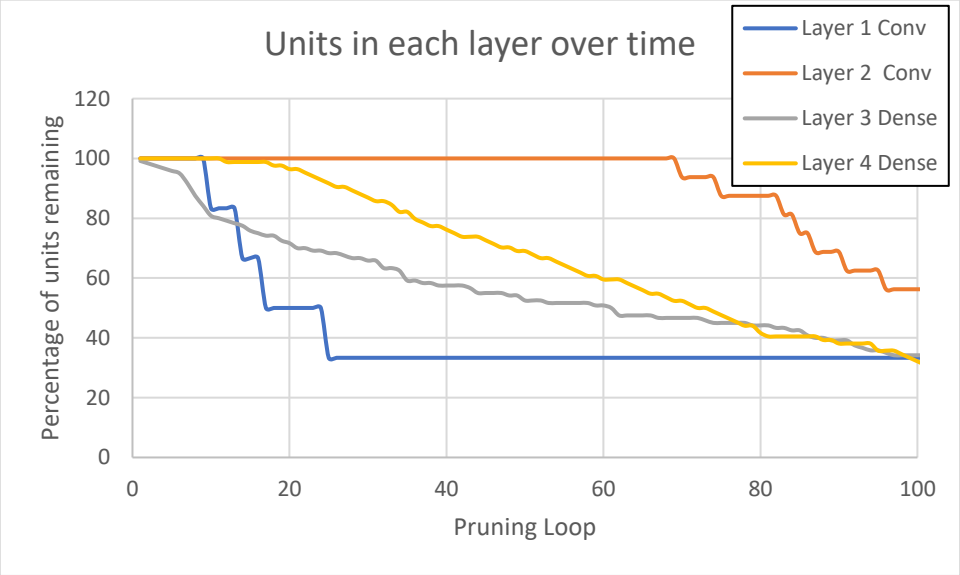


Figure 123: Percentage of neurons remaining in each layer over multiple pruning loops.

Table 49:Final configurations for the model trained and pruned on Fashion MNIST and classic MNIST.

Layer	Number of units - FASHION	Number of units - DIGITS
Conv - Layer 1	2	2
Conv - Layer 2	12	5
Dense - Layer 3	51	21
Dense - Layer 4	34	30

Here is the final configuration for the Global LIM results for both MNIST experiments. The first thing to notice is the number of conv units in layer 2 that were preserved over the MNIST-digits results. And the number of units in dense layer 3 show a similar near double relationship. Figure 124 also seems to show the FASHION results have a slightly quicker pruning profile. The pruning of the 2nd conv layer starts at loop number 69 for FASHION, when pruning on the Digit - MNIST network this started at loop 86.

7.6.3.4 Conclusion

The Fashion results show that the method can easily adapt to other datasets and is inherently aware of the complexity in a model. This can be seen through a different architecture evolution with more pruning of the fully connected layers at the start of pruning and less towards the end when compared with the pruning results for MNIST.

Additionally, the final network structure is vastly different, showing many more convolutional filters being retained and overall, more FLOPs and parameters being preserved. This is most likely because this dataset has more detail in its images and therefore requires more filters to properly separate the classes.

7.6.4 VGG-16 on CIFAR-10

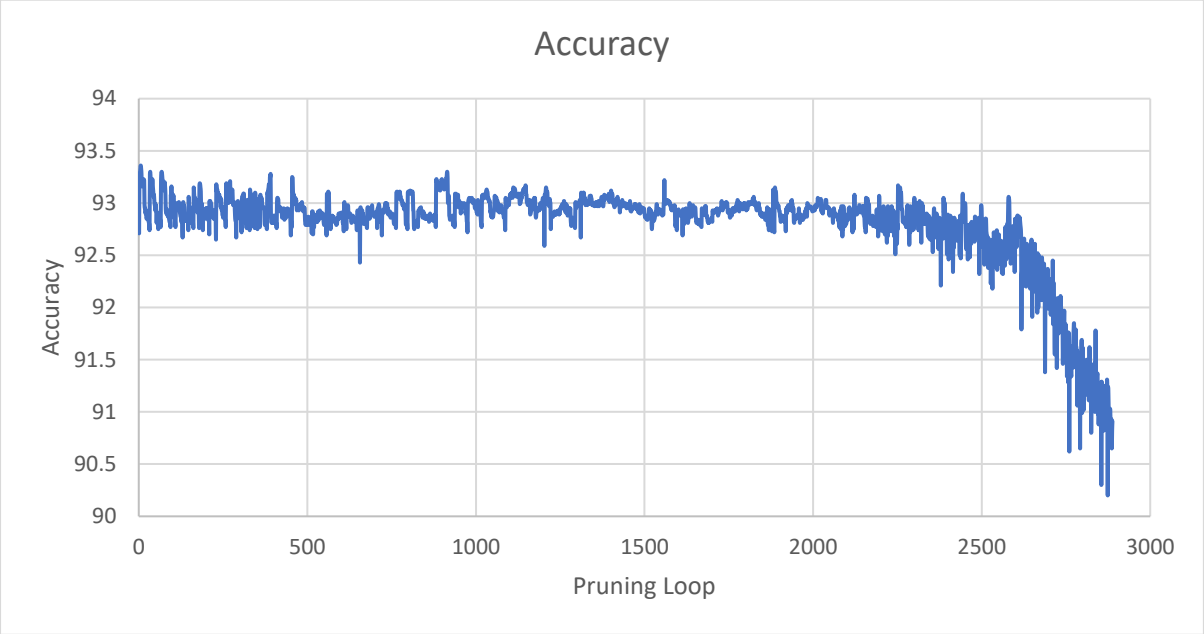


Figure 124: Accuracy of pruned model at all points in the pruning process

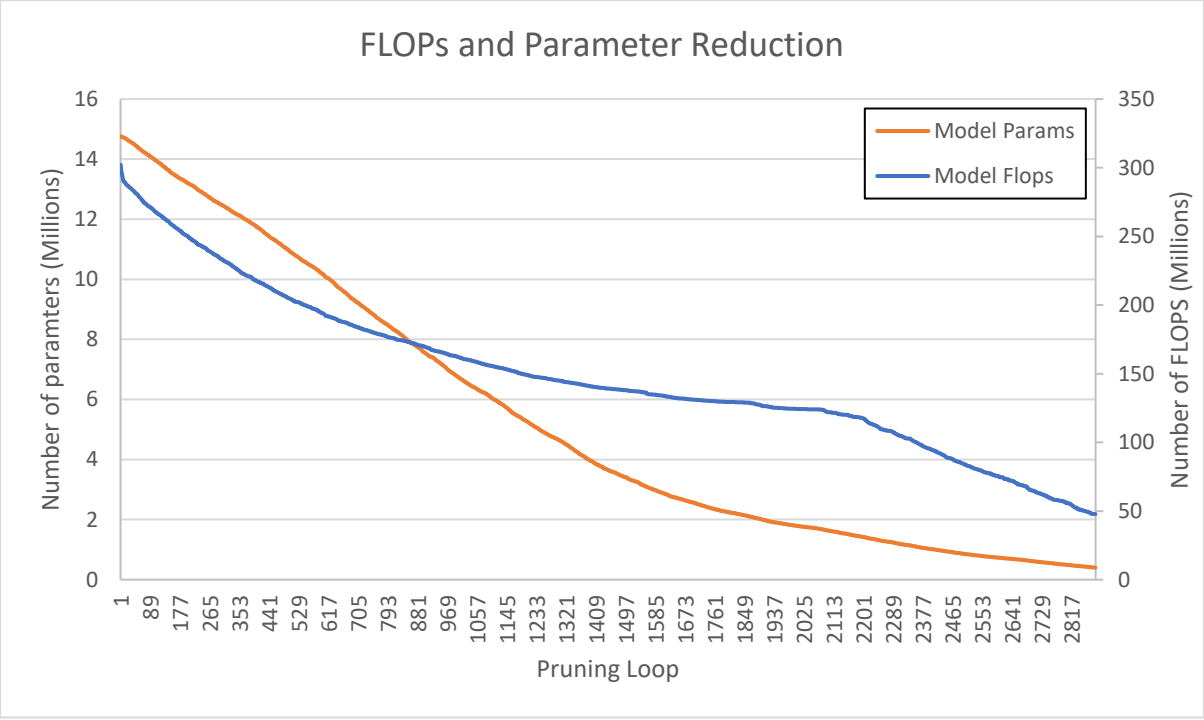


Figure 125: FLOPs and parameters of each pruned model at each pruning loop

Figure 125 shows good accuracy over time, with the process having to reach pruning loop 2580 before any significant degradation in the model accuracy can be observed. Also, clearly observable in Figure 126 is the significant reduction in the parameters near the start of pruning. Comparing this to Figure 125 shows that this reduction in parameters has a minimal impact of model performance. Whilst the reduction in parameters is initially faster than FLOPs Figure 133 shows that the fully connected layer was not significantly pruned until loop 1750, meaning the conclusion drawn previously cannot be used for VGG-16.

It seems that each network has its own unique relationship between FLOPs and parameters which depends on the input data dimensionality, the type of layers and their connections.

These results indicate that FLOPs are more indicative of "intelligence" of a neural network, or at least FLOPs are a better indication of learnt characteristics than parameters. This conclusion comes from the rate of change of these two parameters with relation to accuracy. The FLOPs decrease at an increased rate after loop 2200 seen in Figure 126. Also observed in Figure 125, loop 2200 is where the results move away from the 93% accuracy that had been retained since the original training.

7.6.4.1 Comparison to Similar Techniques

*Table 50: SOTA pruning methods using the model VGG-16 and dataset CIFAR-10 , best result in **bold** second best underlined.*

Model	Flops	Params	Top 1%
Original (Ours)	314.59M (0%)	14.73M (0%)	93.02
VC NNP [92]	190M (60.40%)	3.92M (73.34%)	93.18
GAL [93]	171.89M (45.2%)	2.67M (82.2%)	93.42
FPC [94]	107.85M (66.00%)	0.71M (95.17%)	**94.08**
OAS [95]	106.23M(66.23 %)	1.02M (93.05%)	93.79
EZCrop [96]	104.78M (66.60%)	2.50M (83.3%)	93.70
AACP [97]	94.37M (70.00%)	-	93.39
AEF [98]	74.42M (76.34%)	1.65M (88.80%)	93.08
HRANK [99]	73.70M (76.50%)	1.78M (92.00%)	91.23
Ours Global (HA)	70.00M (77.00%)	750.0K (94.91%)	93.03
GBIP [159]	60.53M (80.76%)	2.50M (83.03%)	93.70
HBFP [100]	51.90M (83.42%)	2.40M (83.77%)	91.99
Ours Global (HP)	50.99M (83.79%)	**450.0k (96.94%)**	91.78
HRel [101]	**47.98M (84.85%)**	750.0K (94.91%)	93.54

These results show that when pruning VGG-16 all the way down to 70 million FLOPs the accuracy stays within a reasonable range of all other approaches. However, when inspecting networks past this point, we can see that our approach begins to get out performed.

Unfortunately, there is no contest when it comes to comparisons with HRel. They achieve higher accuracy with 20 million less FLOPs and an equal number of parameters to our 70 million FLOP model.

HRel is slightly different from our method as it selects which filters to prune by comparing how similar each filter is. Our approach does not take into consideration the similarity of filters, and this is probably the reason for the gap in performance. This is also the first indication of an unfortunate truth when pruning neural networks, and that is your pruning technique is only as good as your retraining technique.

7.6.4.2 Base Models and Retraining

When structurally pruning neural networks, it is essential to retrain or "fine tune" the network after pruning has been applied [101,100,99,96]. Fine tuning raises the following questions.

- Is everyone using the same training techniques?
- Is everyone starting from the same model accuracy?
- Is everyone using the same model architecture?

The unfortunate answer to the first two questions is no, and the answer to the third is not always. This can be plainly seen in some of the results published in HRel [101] shown in table 51.

Table 51: This table highlights the disparity in the accuracy of the initial models used by SOTA pruning papers.

Paper	Baseline Accuracy	Pruned Accuracy	Acc
ℓ_1-norm	93.25	93.40	Δ + 0.15
Ayinde et al.	93.80	93.67	Δ - 0.13
GAL	93.96	90.78	Δ - 3.18
ABCPruner [102]	93.02	93.08	Δ + 0.06
MINT [103]	93.98	93.43	Δ - 0.55
CFP [104]	93.49	92.90	Δ - 0.59
HRel	93.90	93.54	Δ - 0.46

Here you can see the large spread of baseline models used by these papers, some are available online and some have been trained from scratch by the paper authors. The main discrepancy comes in the training and retraining of these models. For instance, below is HRel's training parameters for VGG-16.

- Training epochs 300
- Learning Rate 0.1 reduced by 10 at epoch 80,140 and 230

Importantly batch size and image pre-processing are not mentioned in the paper, however from the attached GitHub we know the following was applied.

- Random 32x32 crop with 4 padding pixels
- Random horizontal flip
- Normalised using values ((0.4914, 0.4822, 0.4465), (0.2023, 0.1994, 0.2010))
- Momentum 0.9 and weight decay of $5x10^{-4}$
- Batch Size = 100

In contrast GBIP [159] uses the following.

- Training epochs 160
- Batch size = 64
- Learning Rate 0.1 reduced by 10 at 80 and 120
- Momentum of 0.9 and weight decay of $1x10^{-4}$

And GBIP does not mention any image augmentation techniques and does not have a GitHub associated with it.

Additionally, sometimes the network architecture is also changed, for example the standard configuration for VGG16 has max pooling layers after each block of similarly sized convolutional layers. However, in the HRel implementation of the VGG-16 model the final max pooling layer is replaced by an average pooling layer, this is not mentioned in the paper and the implications of a differing model structure are unknown.

These techniques need to be far more clearly laid out than they currently are by the neural network pruning research community. Hiding an inferior pruning technique behind a better training technique or longer training cycles or a superior network structure is misleading and arguably makes all these results somewhat incomparable.

However, these are the best metrics the pruning community has, so I will continue to compare these models pointing out inconsistences where I believe relevant.

7.6.4.3 Confusion Matrix

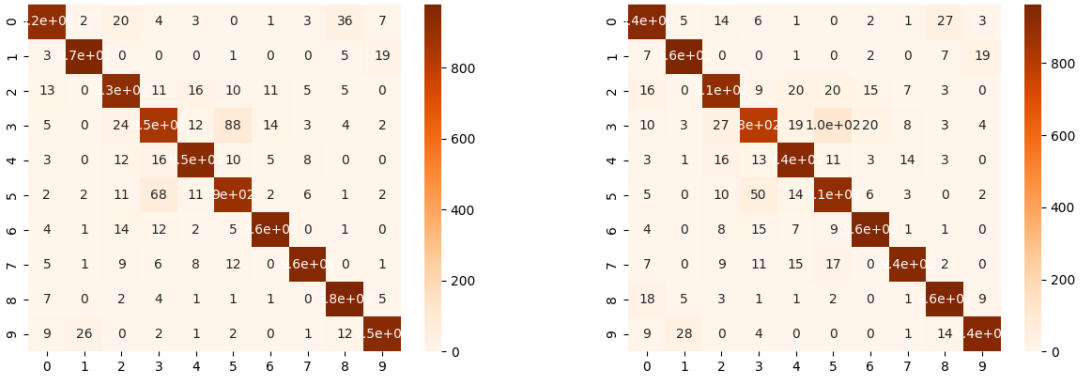


Figure 126: The confusion matrix on the left shows the un-pruned VGG-16 network and the confusion matrix on the right shows the confusion matrix of the pruned model highlighted in green in the table in the previous section:

These confusion matrices are the most interesting so far, this shows the biggest increase in error of a single class, looking at class 5 misidentified as 3 the number of examples increase from 68 to 191. The rest of the confusion matrix shows that the model does not change much other than this one class, this suggests that these two classes are very similar in structure and need fine-tuned convolutional filters to differentiate between the two.

Interestingly class 5 and 3 correspond to cat and dog, which seem quite different, however in the CIFAR dataset they may represent the most similar classes. As a reminder the classes in order are airplane, automobile, bird, cat, deer, dog, frog, horse, ship, truck.

Additionally, there may be some issues with the CIFAR dataset, for example Figure 128 is entered into the dataset as a cat, however I can understand how it has been identified as a dog.

Figure 127: Example of an ambiguous image in the CIFAR-10 dataset

This is just one example however, it shows that the task of separating a cat and dog in this dataset may in fact represent one of the harder tasks.

7.6.4.4 Resource Usage

For the inference test the model was tested on predicting 10,000 samples at a batch size of 64, this was repeated 20 times. All the following results are performed on the HA model using a GTX2080 Super and an Intel Core i7-9700K CPU @ 3.60GHz.

7.6.4.4.1 Speedup GPU

Table 52: Model speedup on a GPU

Model	Time (s)	Error
Baseline	0.09	6.98%
Pruned	0.06	6.97%

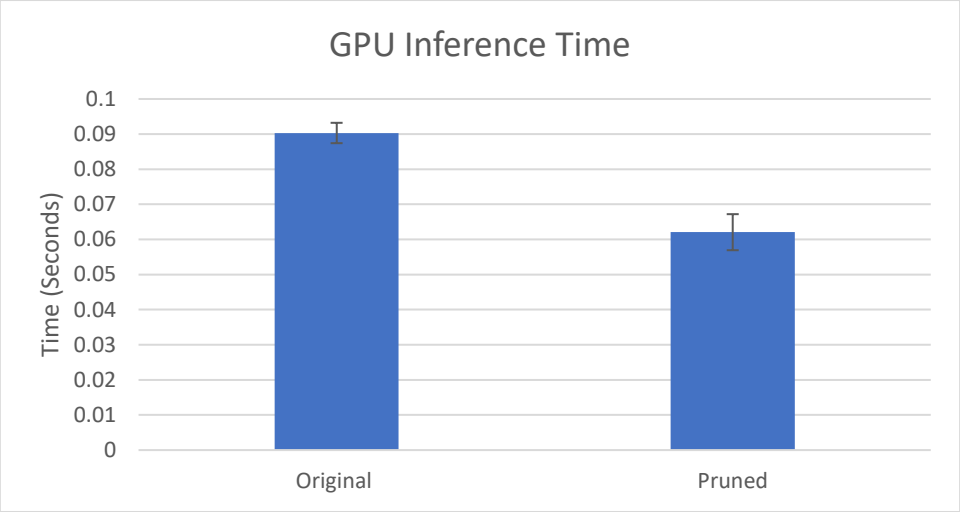


Figure 128: Model speedup on a GPU showing a speedup of 31.34% was achieved.

7.6.4.4.2 Speedup CPU

Table 53: Model speedup on a CPU

Model	Time (s)	Error
Baseline	1.38	6.98%
Pruned	0.72	6.97%

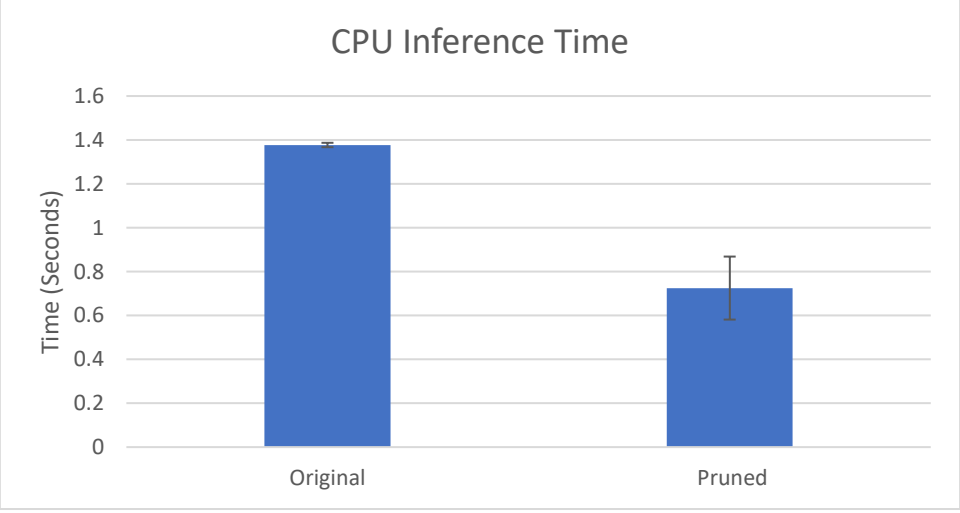


Figure 129: Model speedup on a GPU showing a speedup of 47.35% was achieved.

7.6.4.4.3 Memory Use

Whilst the model was pruned the amount of VRAM used also reduced, this will differ depending on the hardware being used, in this case it was deployed on a NVIDA A100 GPU.

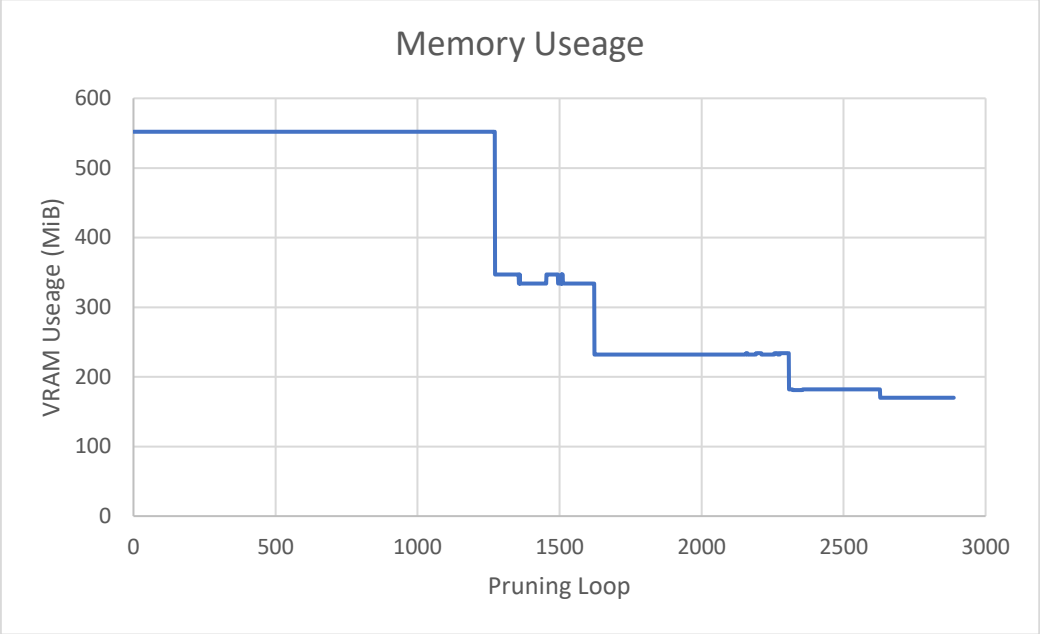


Figure 130: Plot showing how pruning affects VRAM usage.

The model starts using 552MB of VRAM and the pruned model uses 182MB of VRAM, this is a reduction of 67.03%, the model size on disk reduced from 58.7MB to 6.1MB a reduction of 89.60%.

7.6.4.5 Arch evolution

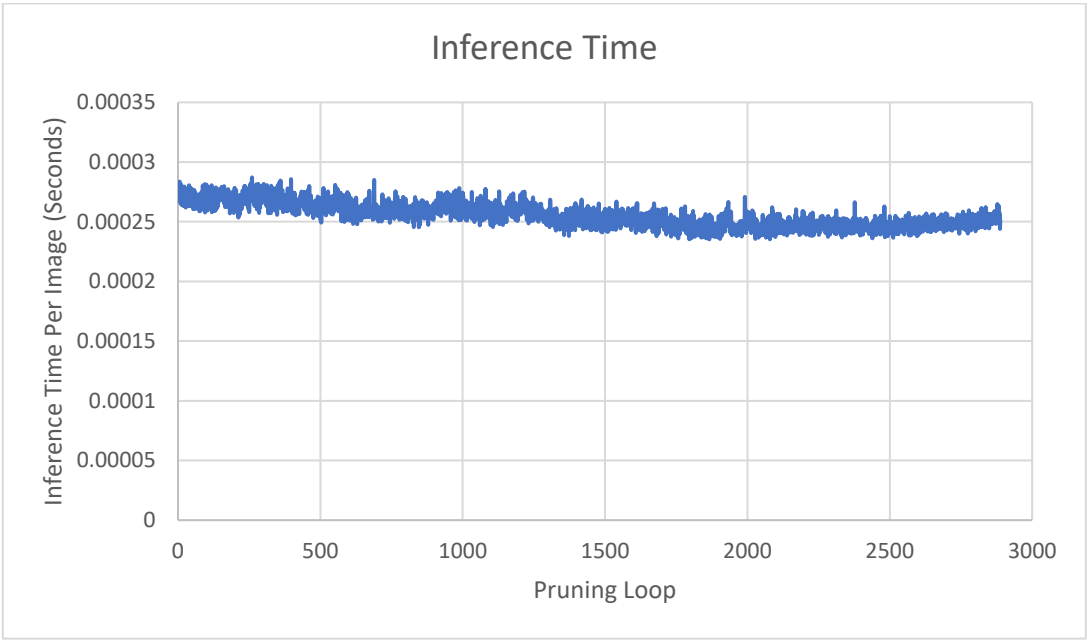


Figure 131: Inference reduction of pruned models deployed on an A100.

Figure 132 shows the inference time decrease, the times here are GPU inference per sample and were tested on an A100 GPU. This shows the benefit of pruning is highly hardware dependent. Figure 132 shows the reduction is on average 0.000275s -> 0.00025s which only represents a 10% inference reduction.

At the time of writing an A100 GPU is the 2nd most powerful GPU for machine learning applications, and whilst the impacts of pruning are still statistically significant the inference reduction is far less than on devices with lower compute power.

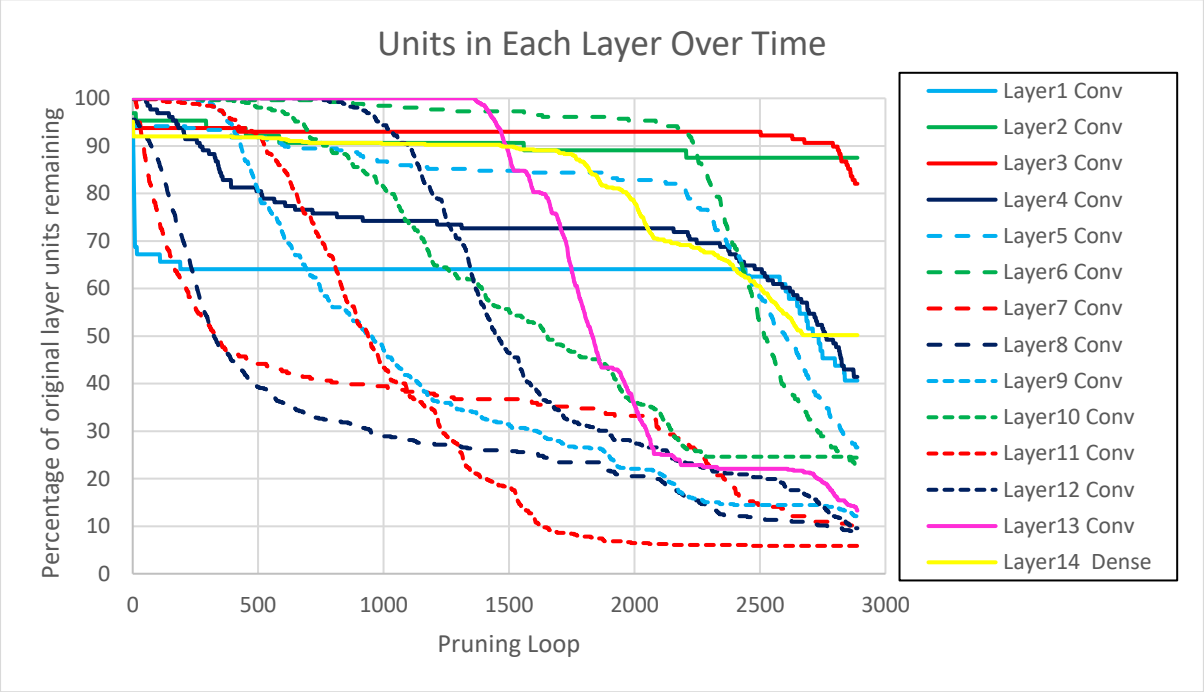


Figure 132: Percentage of neurons remaining in each layer over multiple pruning loops.

Figure 133 shows the number of filters pruned over time. Contrary to past experiments, where there has been a strong correlation between pruning fully connected layers and the rate of pruning parameters. This link does not exist in these results.

The pruning of the dense layer is primarily contained between loops 1500-2000 and 2250-2600, this does not correspond to the parameter graph in the previous sections. This could be because the fully connected layer in VGG-16 represents a small portion of the overall model. It's still true that proportionally fully connected layers contain more parameters than FLOPs, however, the pruning in the rest of the model dwarfs the previous effect we have been observing.

There is a sharp increase in the rate of FLOPS pruned at loop 2200, this looks to be closely linked to the pruning of convolutional layers 6 and 5. At this point in training these two layers represented the highest concentration of information in the network. Both layers are directly connected to each other and still had more than 80% of their original filters whilst most other layers had been reduced to less than 30%. This is why the pruning of FLOPs seems to speed up. It is just a side effect of the fact that there are not many other layers left to prune, and these layers were pruned at a high rate, shown by the plunge in their respective lines over a small period.

Table 54: Final configuration of the model at two specific points; The "inflection point" and the "high accuracy model".

Model Layer	Original Units	Pruned Units (Inflection point)	Pruned Units (HA)
Conv1	64	41	36
Conv2	64	56	56
Conv3	128	119	118
Conv4	128	92	78
Conv5	256	205	131
Conv6	256	238	105
Conv7	256	71	36
Conv8	512	83	58
Conv9	512	34	74
Conv10	512	135	126
Conv11	512	31	30
Conv12	512	119	99
Conv13	512	117	113
Dense1	512	354	284

The pruned model seems to have discovered a new form of filter organisation, shown in table 54 and Figure 134. “Funnels” have been created at multiple points in the network. Looking at layers conv1, 7 and 11 you can see all units are around the same value of 30-36. After each of these layers there is a slow expansion of the number of units in subsequent layers until it is reset again. The location of Max pooling layers is indicated by black bars in table 54. These layers were highlighted to check if there was a correlation. However, there does not seem to be any corelation between the two. This experiment was repeated multiple times, and “funnel” structures were found every time.

Table 54 also has another model named “inflection point”, this model indicates the point at which accuracy begins to drop and re-training increases. This could indicate that at this point a part of the network is being restricted too much, and perhaps some kind of pruning limitation on the most pruned layers needs to be enforced.

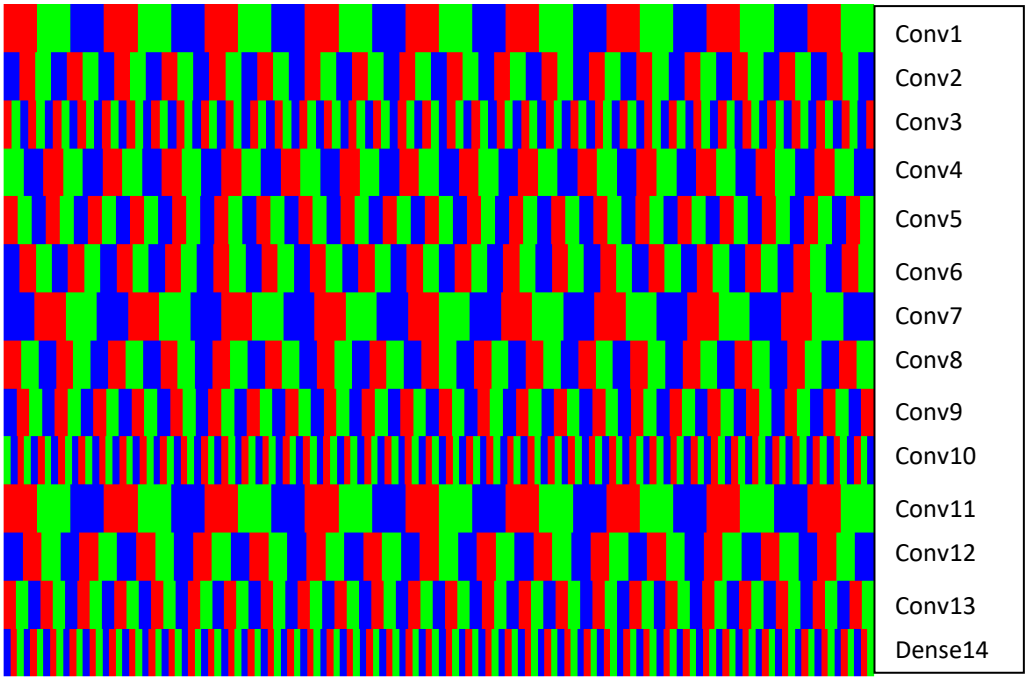


Figure 133: Visualisation of the number of units in each layer in a cascading fashion. This visualisation is taken from the very final model before pruning is stopped which shows the phenomenon to the greatest extent. (An animation of this structure forming can be found in appendix B.1)

7.6.4.6 Ablation Studies

7.6.4.6.1 Training From Scratch

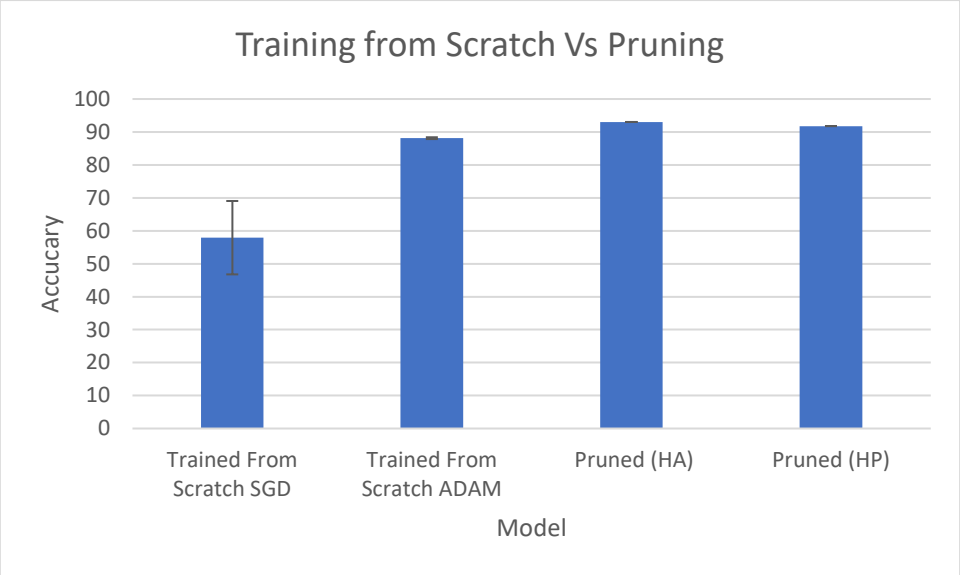


Figure 134: Testing to identify if a model of the same architecture as the pruned model can achieve the same accuracy being trained from scratch, multiple optimisers were used after low retraining accuracy was noticed.

Figure 135 shows the HA model retrained from scratch cannot achieve the same accuracy as the pruned model. These models were originally retrained using the same parameters as the original model, however, when the accuracy training with SGD was extremely low ADAM was adopted to properly represent the capabilities of the network.

This is an indication that the model structure has changed drastically enough so that the previous hyperparameters no longer work to appropriately train the model.

Additionally, to test the changes made in the HRel paper the model was trained from scratch using the same image augmentation techniques and replacing the final max pooling layer with an average pooling layer.

This model achieved a peak accuracy of 93.04%. This doesn't come close to the 93.90% used as a baseline in their paper. It was therefore assumed that some other unmentioned technique was responsible for the improvement seen in their results.

7.6.4.6.2 Using Noise instead of data

The following results show the impact of using noise to prune the model instead of data.

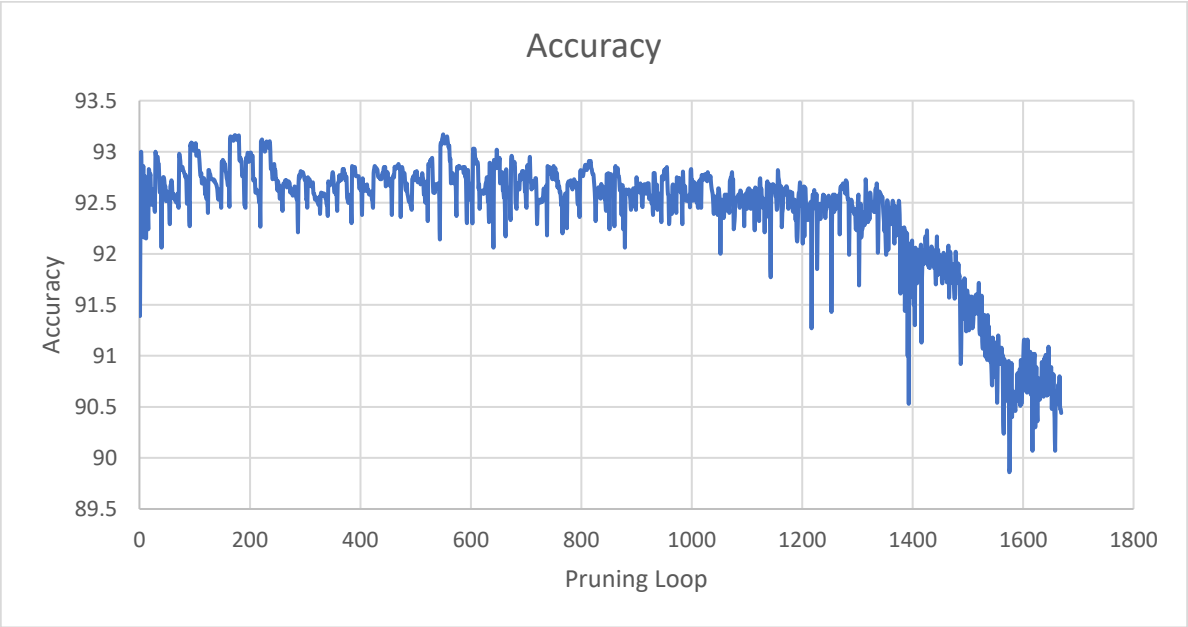


Figure 135: Accuracy of pruned model at all points in the pruning process, using noise instead of signal to locate insignificant neurons.

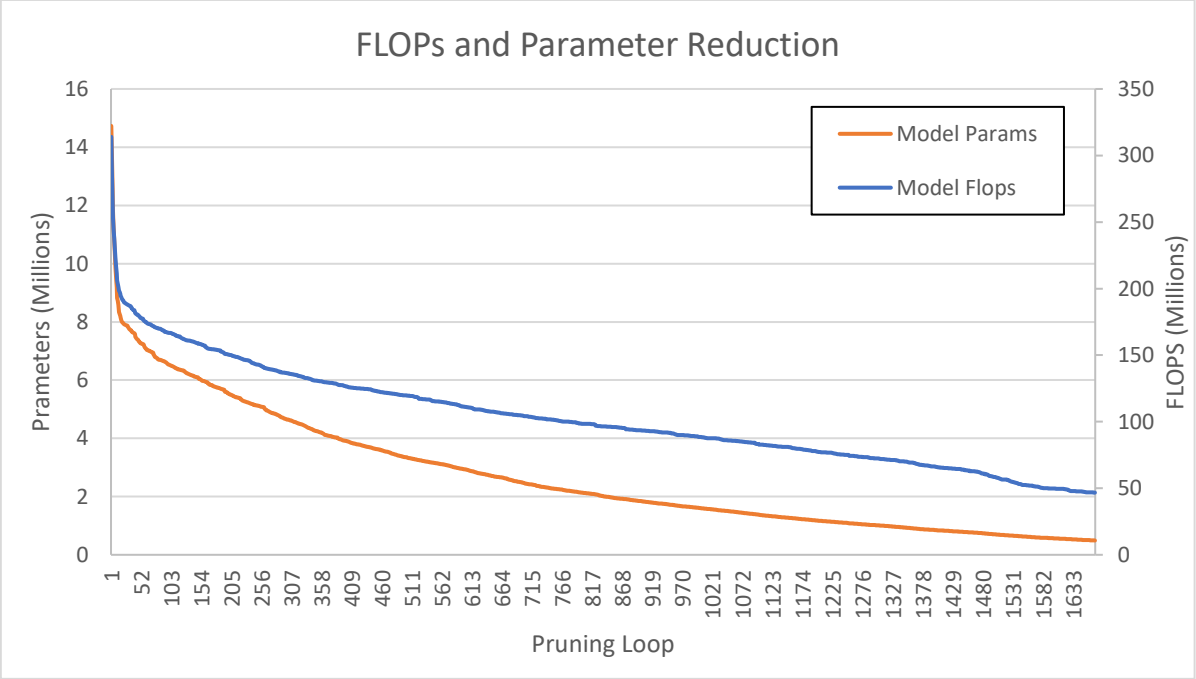


Figure 136: FLOPs and parameters of each pruned model at each pruning loop, using noise instead of signal to locate insignificant neurons.

Figure 137 shows the drastic pruning at the beginning of the pruning loop, by loop 21 this process has reached the same point that takes the data driven approach 550 iterations. The accuracy of this approach whilst on average lower than the data driven approach shown in Figure 136, does recover to a comparable accuracy all the way till loop number 600.

These results make a strong argument that these two stimulation methods should be combined in some way to achieve an overall faster process.

To make a fair comparison to the data driven tests, models were selected using a similar number of FLOPs as the data driven results test.

Table 55: Comparison of Noise or Signal techniques to prune VGG-16.

Model	Flops	Params	Top 1%
Original (Ours)	314.59M (0%)	14.73M (0%)	93.02
Ours Data Global (HA)	70.00M (77.00%)	750.0K (94.91%)	93.03
Ours Noise Global (HA)	68.01M (78.38%)	892.3K (93.94%)	92.52
Ours Data Global (HP)	50.99M (83.79%)	450.0K (96.94%)	91.78
Ours Noise Global (HP)	52.48M (83.31%)	629.2K (95.72%)	91.28

Table 55 shows a consistent .5% deficit to the pruning with data results. Additionally, even though both graphs show that the accuracy for noise and data varies, there is more variance in the results for noise, this can be seen in the jagged accuracy profile in Figure 136.

This indicates that whilst pruning with noise may work, albeit at a deficit to the data driven method, it is not correctly identifying which parts of the network to remove. Or at least it is not performing this process with the same efficiency as the data driven method. This also has the side effect of triggering more model retraining, which makes both methods similar in total pruning time. Put simply, although this approach takes 1200 less pruning loops to achieve a pruned model, the total pruning time is balanced out by the increase in retraining.

Interestingly the model also has a very different configuration than when using a data driven approach.

Table 56: Comparison of the final configuration of the pruned models for VGG-16, using Noise or Signal pruning methods.

Model Layer	Original Units	Pruned Units Data (HA)	Pruned Units Nosie (HA)
Conv1	64	36	41
Conv2	64	56	56
Conv3	128	118	91
Conv4	128	78	84
Conv5	256	131	120
Conv6	256	105	90
Conv7	256	36	26
Conv8	512	58	61
Conv9	512	74	54
Conv10	512	126	59
Conv11	512	30	69
Conv12	512	99	233
Conv13	512	113	128
Dense1	512	284	172

Table 56 suggests that these two approaches are optimised for different goals. Whilst the noise pruning method is demonstrably faster to prune in the early stages, it seems to slip out of the local minima quicker than the data driven pruning.

7.6.4.7 Conclusions

Overall, our results show that up until a model size of 70 million FLOPs our pruning method is comparable to all other SOTA methods, however when the model is pruned further to 50 million FLOPs HRel beats out our method by nearly 2% in accuracy metrics.

When results are compared on the number of parameters, our results are closer to HREL with only a .5% decrease in accuracy for the same number of parameters.

The ablation results show using noise instead of data as a stimulus to the neural network results in worse pruning than if data was used. This is no doubt since the network needs to be excited in a similar fashion to the training data, to properly highlight the true utilisation of a given filter. However, if data is lacking for this purpose, or the network needs to be pruned quickly, the noise method can be used to achieve similar (yet slightly degraded) results.

Again, when the HREL code is inspected there are unmentioned pruning limits provided for the first 2 layers of VGG-16. Limiting the amount of pruning to 57 units in these convolutional layers, this is once again not justified in the paper.

However, it must be noted that despite this HREL seems to outperform most other SOTA methods and our own, this is most likely down to the underlying method of HREL, which instead of locating "redundant" filters or neurons in a vacuum. HRel locates the most "similar" filters. Assuming that these similar filters are the most important because they have been generated twice by the model seems to preserve the pruning performance.

For future research a dual-purpose pruning system that considers similar filters and unimportant filters would most likely produce the best results.

7.6.5 VGG-16 on Birds 300 Kaggle

This example was used as a case study to show collaborators at the BBC, how neural network pruning can be useful in the real world. This work focuses on a scenario that is already using deployed neural network models to aid with video production.

7.6.5.1 Real world example and use-case for the BBC

The BBC covers many topics from football, to elections, it aims to provide quality content to the UK public and beyond [156]. As the BBC grows the ability to take advantage of automation for ease of video production has grown.

The main news studios in London, and in many other locations, now use automated cameras, replacing the need for manual camera operators [157]. Additionally, this makes shows like the news much more predictable, with set camera shots that are 100% repeatable, every time the show is filmed.

This case study will focus on the BBC family of shows under the umbrella name Bird Watch, Bird Watch is a shorthand to describe the specific shows of Winter watch, Summer watch, Autumn watch and Spring watch.

Birdwatch has an issue when it comes to content, birds are not actors, they cannot be paid to appear on set when summoned. Because of this birdwatch uses 32 cameras, recording 24/7 for 3 weeks whilst filming.

In total that's 672 days of raw footage that needs to be analysed, and birds identified for editors to use the footage in an efficient way. Currently machine learning systems are used to aid in the process, these systems help identify birds in live footage. However, performing analysis on all the footage is considered unfeasible. Additionally, the infrastructure to return 32 camera feeds to one central live location is also unfeasible. This is because birdwatch is normally filmed in woods and rural areas, meaning the camera feeds must be switched between and cannot all be monitored at once. To combat this, the BBC have suggested a low power machine learning device, this would be at the location of the camera and send a much lower bandwidth, detection signal, when an animal of interest is detected. This allows operators to know which cameras to switch to, without having to send the whole camera feed 24/7.

This problem forced an expansion of the pruning technique to include a transfer learning step. this expansion was necessary as the BBC takes responsibility very seriously and this would allow them to use their own dataset to tune any detection algorithms to the mammals and birds that they wish.

7.6.5.2 Transfer Learning Step

The base network used for the following experiment was VGG-16, however the top was removed, this means removing the final prediction layer and the fully connected layer before it. These two layers were replaced with 3 new layers, 2 which were fully connected dense layers of size 4096 and one more of size 300 for classification.

The weights from a pretrained VGG-16 model on the ImageNet dataset were loaded into the top of the model. The top of the model was then frozen, and the three new layers were trained for 15 epochs using the ADAM optimiser at a learning rate of $1x10^{-4}$, the model was then unfrozen, and the learning rate was reduced to $5x10^{-5}$ and trained for a further 15 epochs.

This completes the transfer learning step and achieves an accuracy of 93.7%.

7.6.5.3 Pruning Results

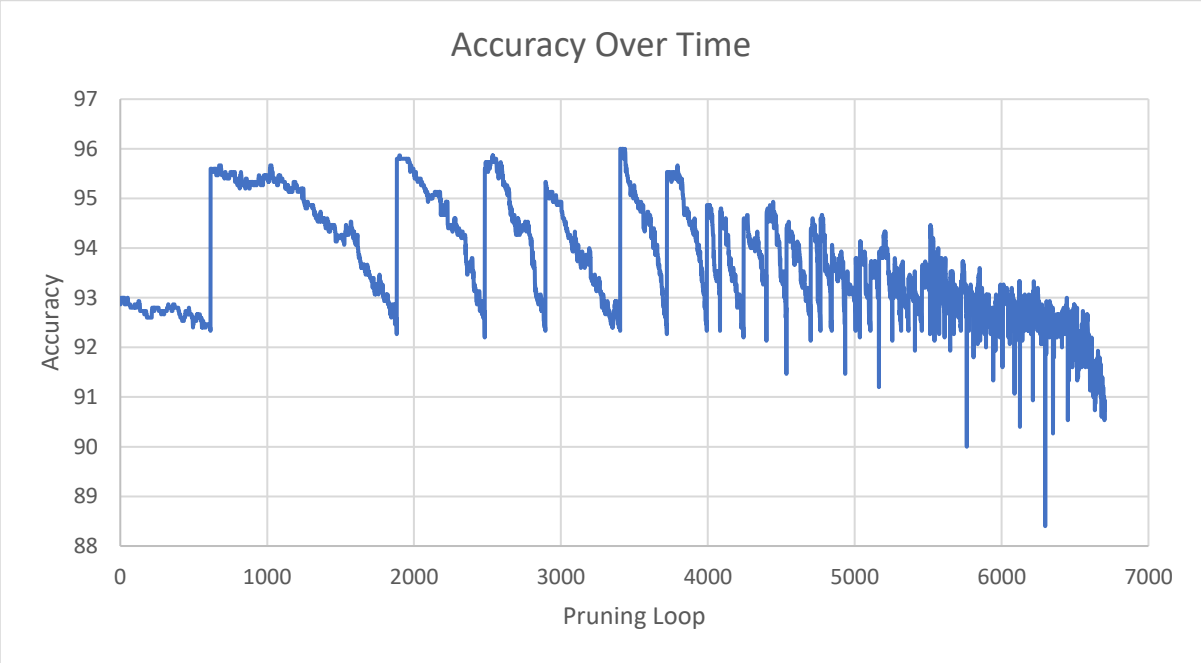


Figure 137: Accuracy of pruned model at all points in the pruning process.

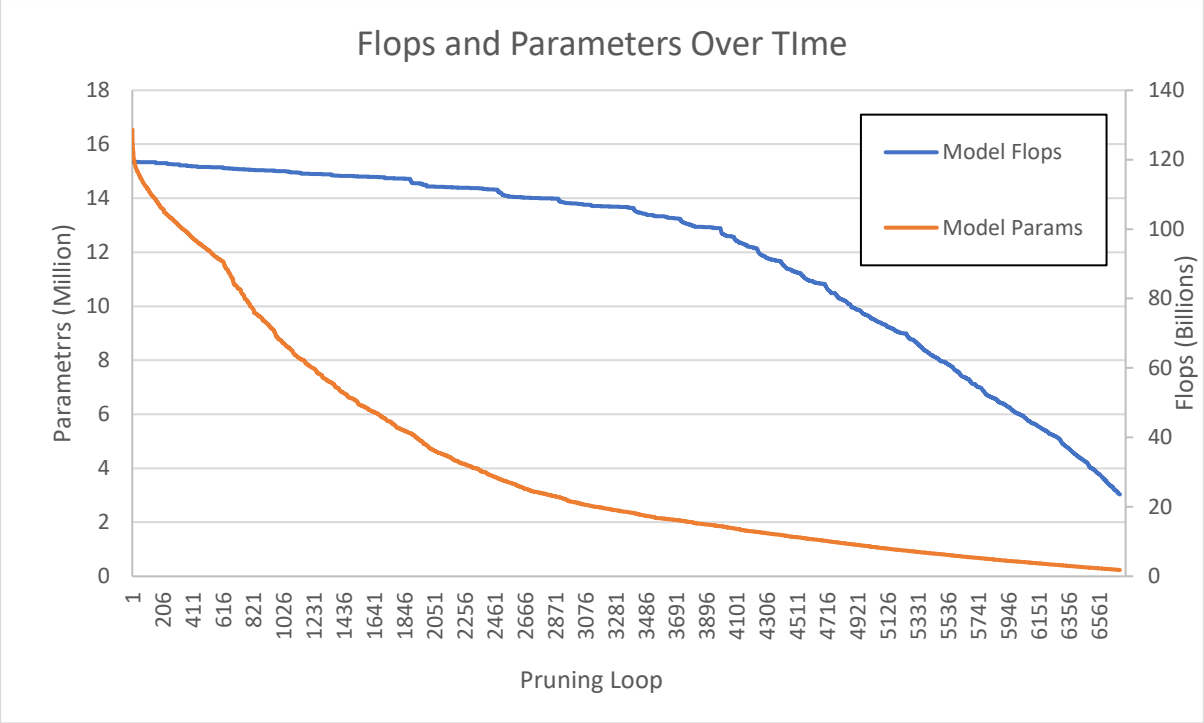


Figure 138: FLOPs and parameters of each pruned model at each pruning loop.

Figure 138 is of interest here as it is the first time that model retraining is obvious through this metric. Retraining can be seen in the graph as a sharp vertical increase in accuracy, this highlights how many pruning loops can occur before any retraining is required. This high accuracy is followed by a slow decrease in accuracy over many pruning loops creating a “shark fin” shape in the graph. The frequency of this “shark fin” gives an indication of how often retraining is required by the pruning process, this starts off at a low frequency and increases to a point where eventually the fin is no longer visible. At the point where the fin disappears it can be assumed that the model is having to re-train nearly every pruning loop and the accuracy soon falls below the given threshold ending the pruning process.

Additionally, the pruning process seems to significantly boost the accuracy of the model, this seems to be a direct impact of the model being over parametrised. the initial pruning produces a decrease in accuracy from loop 0 to 614. However, at this point retraining occurs and 35% of the neurons have been removed from both dense layers. This causes a significant boost in accuracy up to 95.53%, this increase in performance could also be an indication that the transfer learning step is not very robust. The removal of convolutional and dense units may have allowed the model to move out of a local minimum, overall making the model more cohesive improving accuracy by 1.8%.

Table 57: Selected models at various points in the pruning process

Network	% Accuracy ↑	Flops ↓	Params ↓
Unpruned	93.7	150B	15M
Pruned_1	94.1	78.7B	6.1M
Pruned_2	93.6	65.8B	4.8M
Pruned_3	92.1	35.1B	2.1M
Pruned_4	91.4	31.3B	1.9M

Table 57 shows the importance of selecting a final model, depending on if accuracy is more important, or the memory usage or compute power. Ultimately further tests would be required before a model was selected for an end user, these tests would be performed on edge compute devices to conclude the “best” model for the scenario. Luckily because this pruning method is iterative, and models can be stored at the end of every loop, a bespoke model for the situation can be found.

7.6.5.4 Confusion Matrix

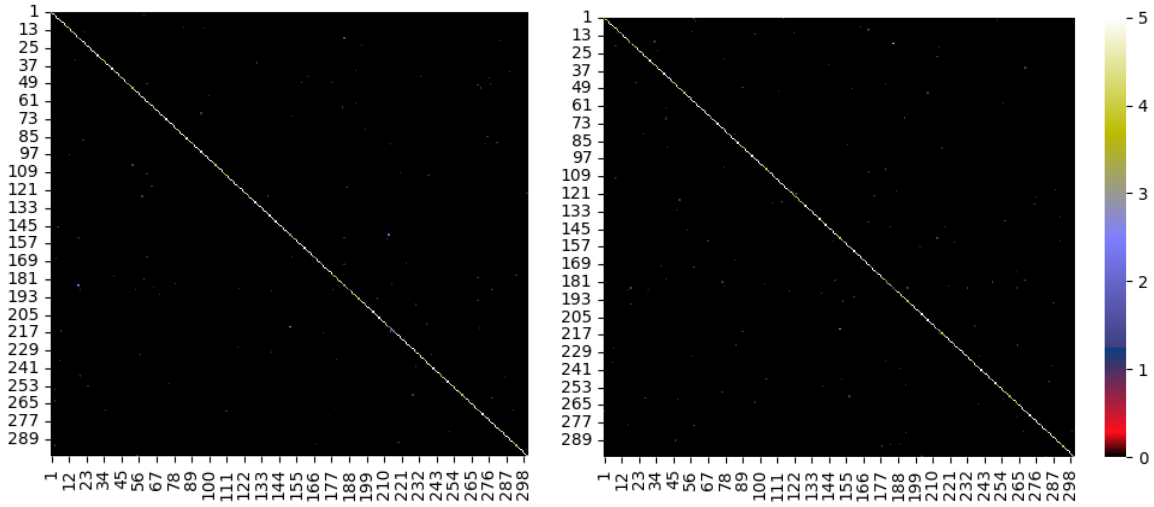


Figure 139: The confusion matrix on the left shows the un-pruned VGG-16 network and the confusion matrix on the right shows the confusion matrix of the pruned model highlighted in green in the table in the previous section

Because the validation set only contains 5 examples per class the confusion matrices in Figure 140 are not very enlightening in this case, I have changed the colour scheme to try and highlight in as much detail as possible the differences.

7.6.5.5 Resource Usage

For the inference tests the model was timed predicting 1,500 samples at a batch size of 8, this was repeated 20 times using a GTX2080 Super and an Intel Core i7-9700K CPU @ 3.60GHz.

For the following comparisons I will focus on the results of the 65.8B FLOP model, this model has very similar accuracy to the original model but has significant savings on resources.

7.6.5.5.1 Speedup GPU

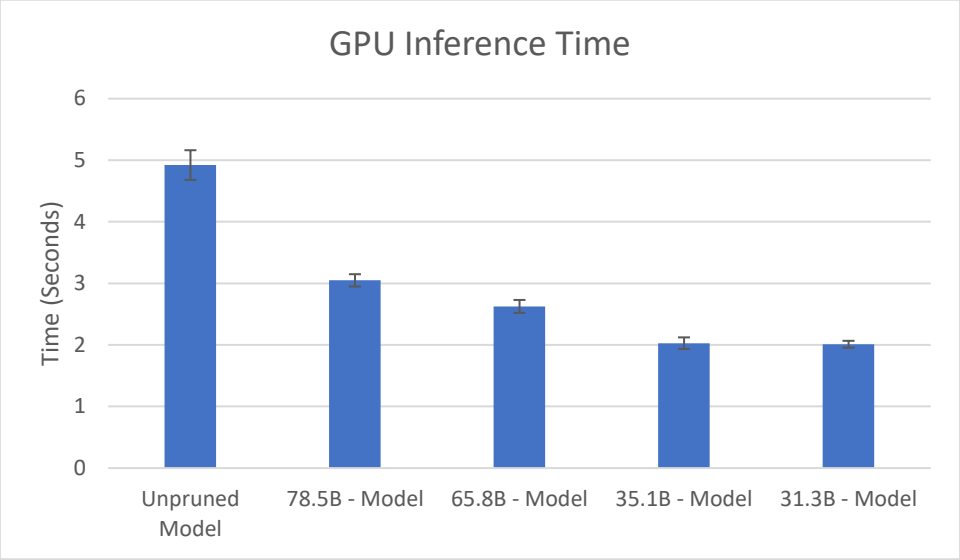


Figure 140: Model speedup on a GPU showing a speedup of 46.64% was achieved.

7.6.5.5.2 Speedup CPU

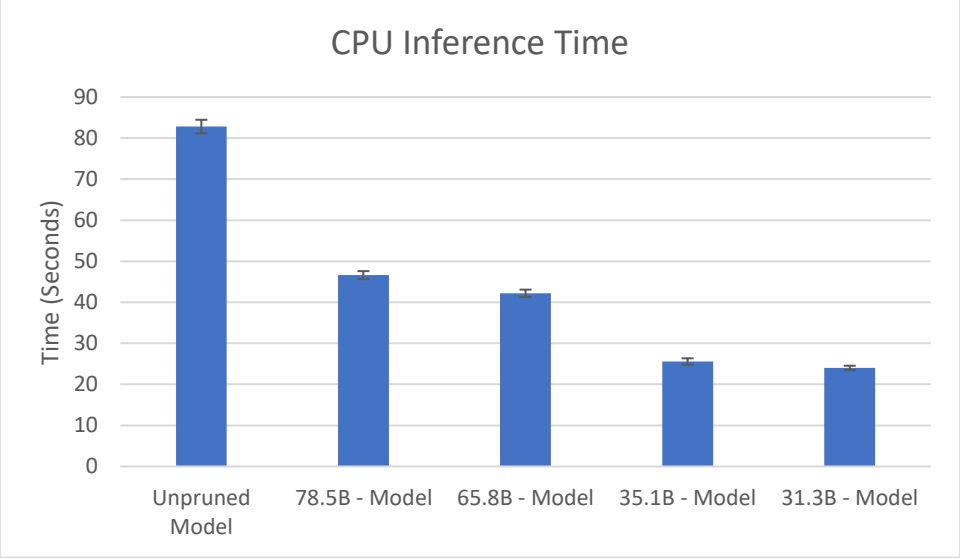


Figure 141: Model speedup on a CPU showing a speedup of 49.04% was achieved.

7.6.5.5.3 Memory Use

Table 58: VRAM and Disk usage for multiple pruned models

Model	VRAM	Size On Disk	Accuracy
Unpruned Model	6654 MB	516 MB	93.7
78.5B – Model	4806 MB	23.5 MB	94.1
65.8B – Model	4806 MB	18.5 MB	93.6
35.1B – Model	2790 MB	8.20 MB	92.1
31.3B – Model	2790 MB	7.30 MB	91.4

VRAM usage was reduced by 27.77% for the first two models and 58.07% for the second two, the 65.8B FLOPS model was reduced in size on disk by 98.41%.

7.6.5.6 Arch evolution

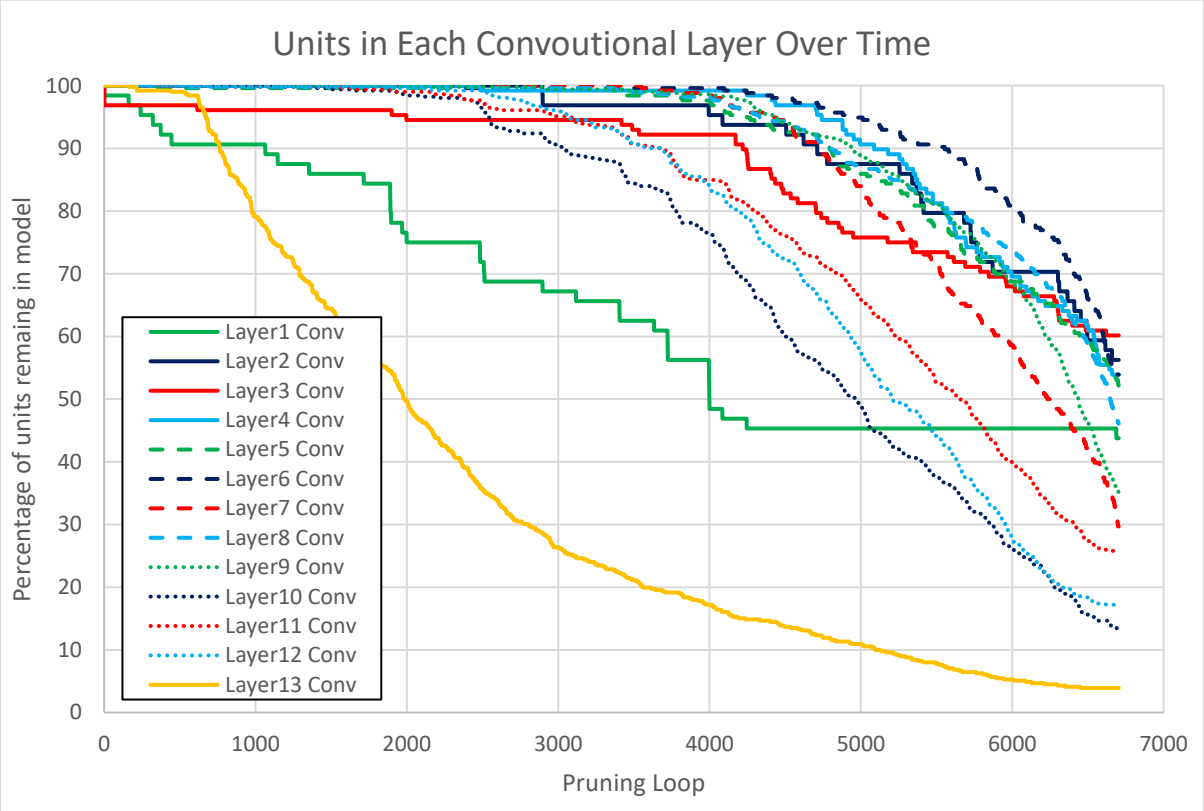


Figure 142: Percentage of neurons remaining in each convolutional layer over multiple pruning loops.

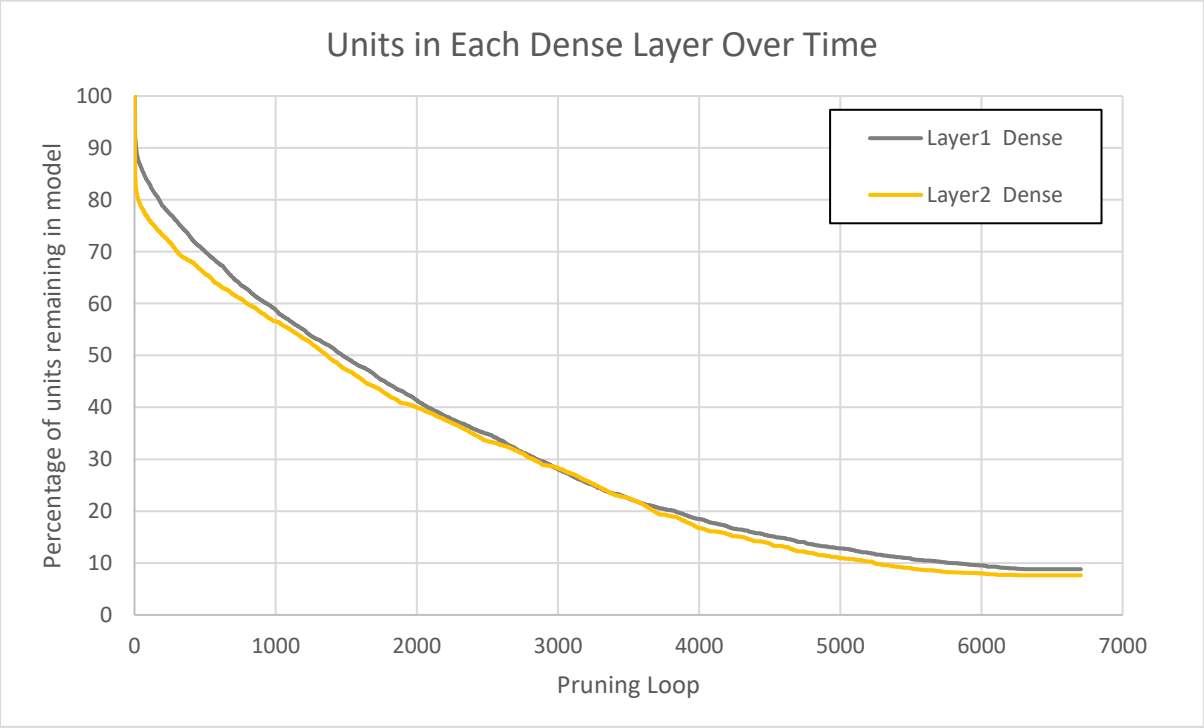


Figure 143: Percentage of neurons remaining in each dense layer over multiple pruning loops.

Of note in Figure 143 the process prunes convolutional layers at a higher rate towards the end of the pruning loops. Additionally, Figure 144 shows the initial parameter reduction can be attributed to the early pruning of dense layers again. The later reduction of FLOPs once again matches the increased in reduction of convolutional units in Figure 143. As with the previous pruning of VGG16 there seems to be an inflection point at 5500 where pruning of all convolutional layer's increases, this increase triggers more retraining by the pruning process seen by an increase in frequency of the "shark fin" shape in the accuracy graphs. And eventually results in a loss of accuracy.

Table 59: Final configuration for the optimal 65.8B parameter model.

Model Layer	Original Units	65.8B - Model
Conv1	64	29
Conv2	64	46
Conv3	128	89
Conv4	128	93
Conv5	256	183
Conv6	256	214
Conv7	256	158
Conv8	512	389
Conv9	512	373
Conv10	512	154
Conv11	512	224
Conv12	512	168
Conv13	512	30
Dense1	4069	405
Dense2	4069	335

A reduced validation set was used to perform the following inference test, models were tested every 15 pruning loops to produce Figure 145.

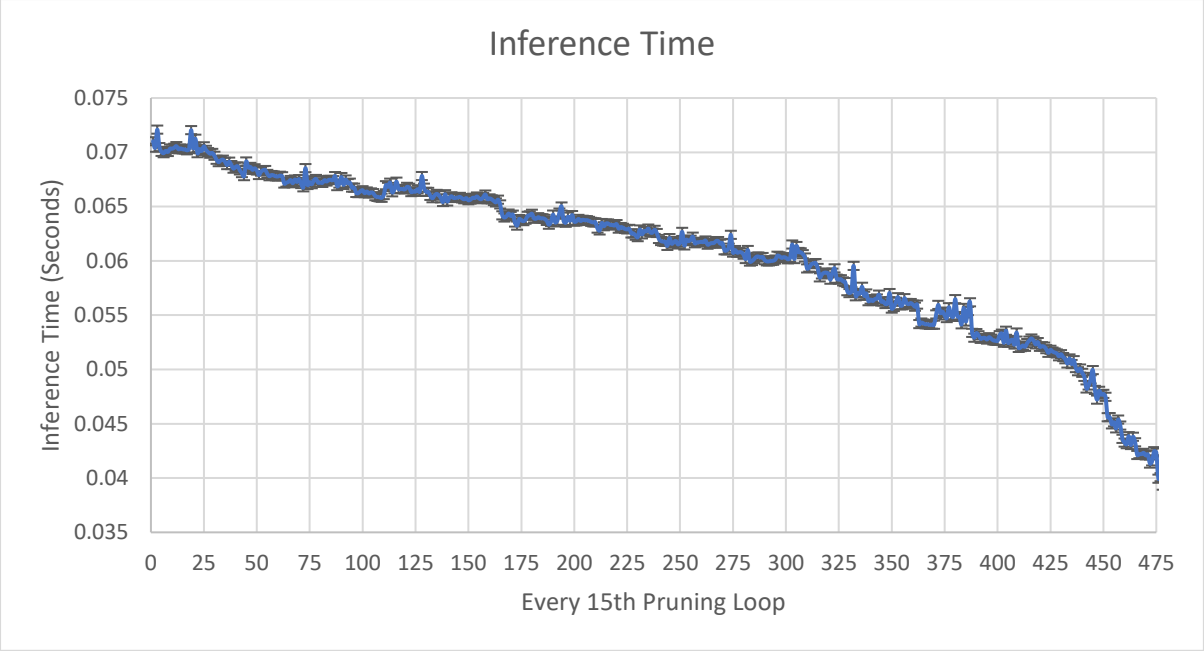


Figure 144: Inference time of each pruned model

Figure 145 shows the general decrease in model inference time as the pruning process is performed, however there are a few occasions where inference time seems to increase. This can be attributed to TensorFlow's automated GPU deployment, as discussed previously a model must be parallel to be fast and sometimes this automated deployment can result in a more optimised model by pure luck. Because each pruning loop changes the number of filters in each layer, this has a direct effect on the automated TensorFlow CUDA deployment, and hence results in some slight increases in inference time even when the model is decreasing in size.

7.6.5.7 Conclusions

We found that by pushing this model to its limit it was possible to reduce inference time on a CPU by 50% whilst only reducing the accuracy of the model by 1.5%. Whilst I cannot reveal the exact amount of money that the BBC currently spends on image recognition tasks this reduction could either represent a saving of 50% on current costs, or a decrease in processing time of 50%.

The team at the BBC has taken this work onboard and continues to develop efficient low-power machine learning solutions.

7.6.6 Resnet-50 on ImageNet

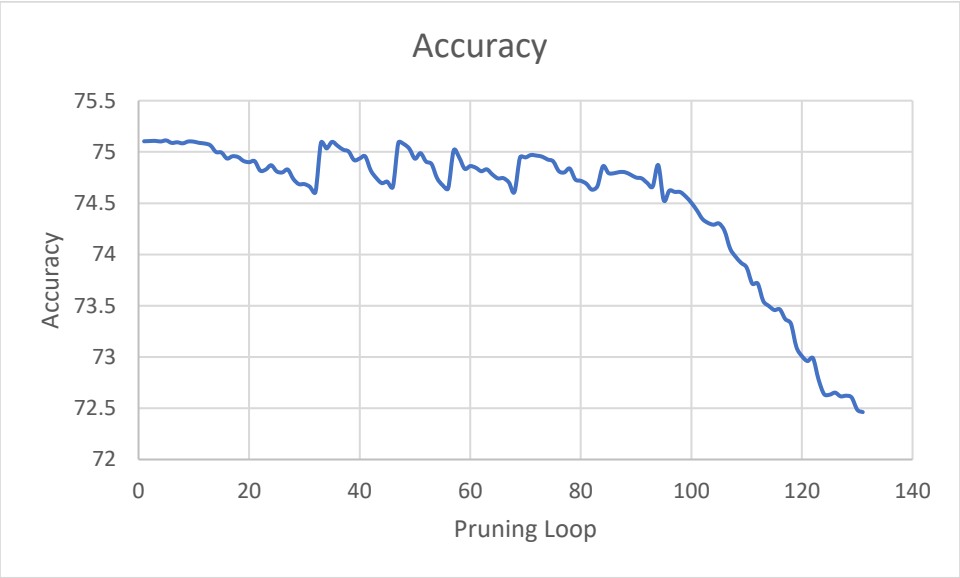


Figure 145: Accuracy of pruned model at all points in the pruning process.

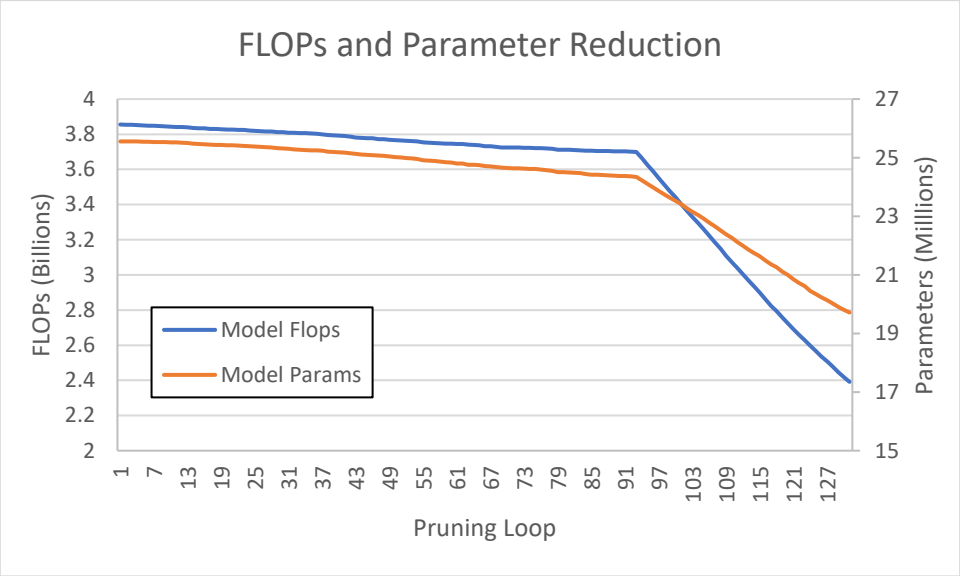


Figure 146: FLOPs and parameters of each pruned model at each pruning loop

Figure 147, at point 97 a stark increase in pruning rate can be observed, this was triggered by the pruning method being changed from global to local. This was done because the time to re-train ResNet50 was much longer than previous experiments and changing the pruning to local increases the amount of the network pruned per loop.

This does however have the side effect of reducing the efficacy of pruning, this is because when local pruning is used instead of global pruning, more assumptions about unimportant neurons are made.

7.6.6.1 Comparison to Similar Techniques

*Table 60: SOTA pruning methods using the model ResNet-50 and ImageNet dataset, best result in **bold** second best underlined.*

Model	Original Top 1%	PrunedTop 1%	Flops Remaining (Removed %)	Params Remaining (Removed %)
SFP [105]	76.15	62.14 (Δ -14.01%)	41.80 %	DNR
GAL	76.15	71.95 (Δ -4.20%)	2.33B (43.03 %)	21.20M (16.86%)
HRank	76.15	74.98 (Δ -1.17%)	2.30B (43.76 %)	16.15M (36.80 %)
Ours	74.90	72.80 (Δ -1.10%)	2.39B (44.80 %)	19.71M (23.00 %)
DMCP [106]	**76.60**	**76.20 (Δ -0.40%)**	2.20B (46.47 %)	DNR
HRel-1	76.15	75.47 (Δ -0.68%)	2.11B (48.66 %)	13.23M (48.24 %)
MetaPruning [107]	**76.60**	75.40 (Δ -1.20%)	2.00B (51.33 %)	DNR
ABCPruner	76.01	73.52 (Δ -2.49%)	1.79B (56.61 %)	11.24M (56.01%)
HRel-2	76.15	74.54 (Δ -1.61%)	**1.69B (58.88 %)**	**10.82M (57.67%)**

There are a few obvious differences in table 60, the largest difference is that of the starting accuracy. This is caused by a divergence in the machine learning backends used to test the methods, TensorFlow 2 is used throughout this work, however PyTorch is used by many competing methods.

Here all methods that use PyTorch start with an accuracy of 76.15% this is 1.25% ahead of any TensorFlow model before pruning has been applied. This is unfortunate because the pruning methods developed in this work are not easily transferable to PyTorch, and the PyTorch model is not easily transferable to TensorFlow.

For this reason, we will focus once again on the % difference between the trained and pruned models. Regardless of this difference, it is plainly obvious from the results that our model underperforms when compared with any method below it in the table. Even after considering the discrepancy in original model accuracy, the pruned model is either outperformed in every reported metric, or out pruned. DCMP and HREL-1 both outperform our results, Meta pruning has a -0.1% accuracy difference when compared to our method, but prunes 40M more FLOPs. Outliers at the bottom of the table (ABC and HRel-2) have a bigger accuracy drop, however they both prune 70M and 80M more FLOPs respectively.

Additionally, results that reported parameter reduction also show more of a decrease in overall parameters. This is most likely since ResNet50 contains 1 fully connected layer at its output, and this cannot be pruned. Our method prunes all layers in the network, whilst other methods focus on convolutional filters only. In this scenario, our method cannot take advantage of being able to prune both layer types, and because as we have previously observed fully connected layers are parameter dense, we observe this deficit.

7.6.6.2 Confusion Matrix

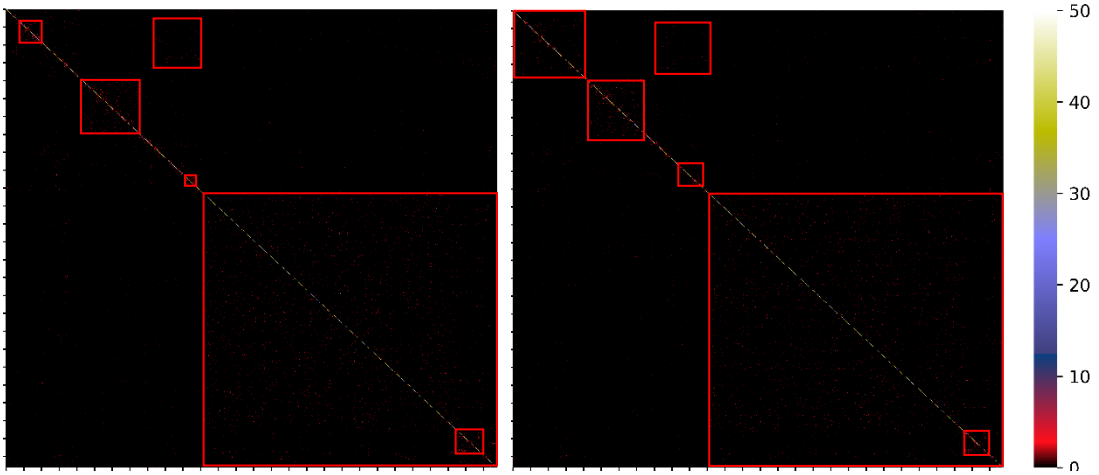


Figure 147: The confusion matrix on the left (HQ version in appendix D) shows the un-pruned ResNet-50 network and the confusion matrix on the right (HQ version in appendix E) shows the confusion matrix of the pruned model highlighted in green in the table in the previous section. A high quality version of the image can be found in appendix

There are red squares plotted in the confusion matrices in Figure 148, these are to highlight areas of confusion. Because of the nature of confusion matrices there are often rectangular areas of "confusion". This is simply because if A is misclassified as B, then B will also tend to be misclassified as A. This causes a symmetry around the diagonal forming rectangular areas.

Of note in Figure 148 the general shape and location of these areas are the same, however they are slightly larger in the pruned model, and some of the smaller regions of high confusion have merged into a larger area of sparse confusion. These plots show that the areas of confusion have generally grown in size which we would expect given the lower accuracy of the pruned model.

7.6.6.3 Resource Usage

For the inference tests the model was timed predicting 50,000 samples at a batch size of 256, this was repeating 10 times using a GTX2080 Super and an Intel Core i7-9700K CPU @ 3.60GHz.

7.6.6.3.1 Speedup GPU

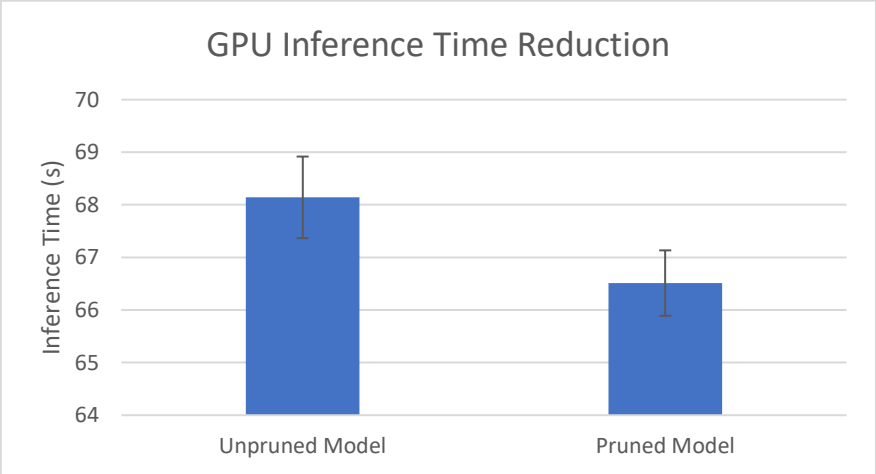


Figure 148: Model speedup on a GPU showing a speedup of 2.39% was achieved.

The unpruned model had an average inference time of 68.14s and the pruned model 66.51s representing a 2.39% decrease, far less than all other models and perhaps indicates that ResNet50 was already well optimized for this task.

7.6.6.3.2 Memory Use

Overall, the reduction in GPU memory usage was negligible however the reduction in the size on disk was reduced from 98.2MB 79.3MB a reduction of 18.9MB.

7.6.6.4 Arch evolution

For the following plot I have excluded layers that didn't change in their number of layers

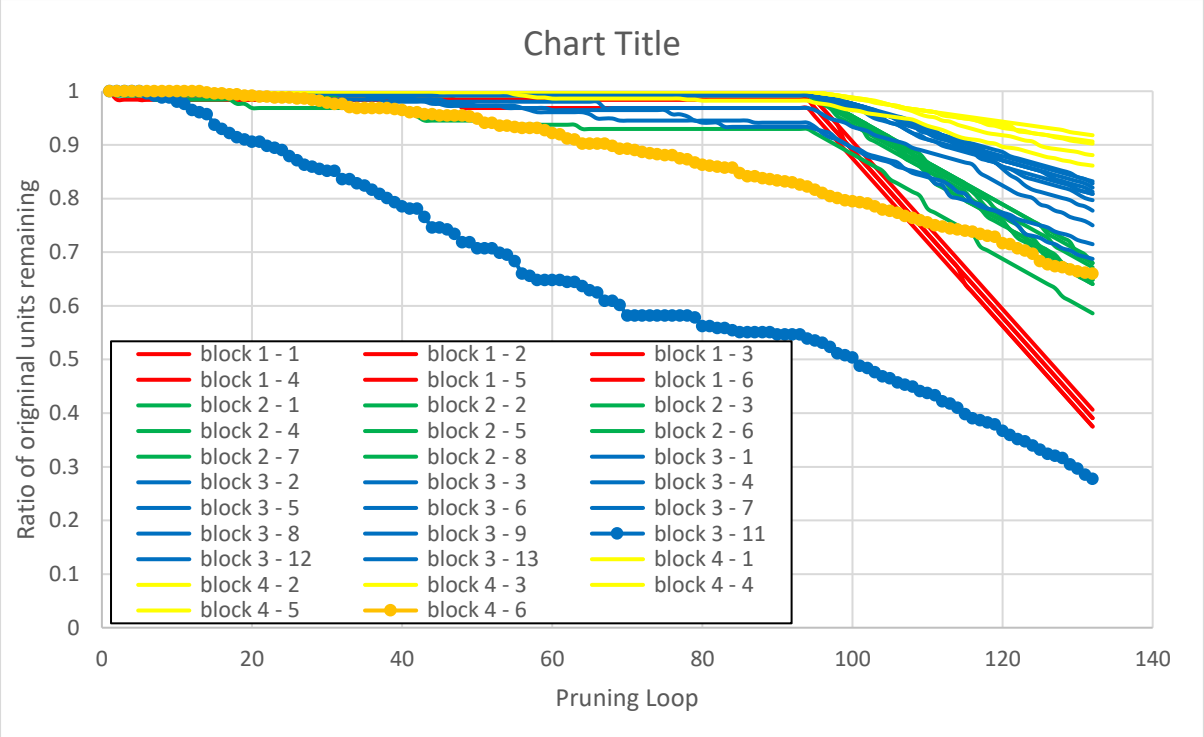


Figure 149: Ratio of neurons remaining in each layer over multiple pruning loops.

Interestingly most layers in Figure 151 remain unpruned until the pruning method was changed from global to local, other than block3 – 11 and block4 – 6 which continue to prune at a similar rate. This is expected, as when local comparisons are made, at least 1 unit per layer will be removed from the model each pruning loop. This is indicated by the straight red lines. A potential future improvement would be for local pruning to only remove filters from a layer, when two have been located for removal within the same layer.

This change would limit the removal of important filters, whilst still only making a comparison within the same layer, however, this also creates a possibility that no filters are pruned in a pruning loop.

7.6.6.5 Conclusion

These results show that our method unfortunately vastly underperforms in all metrics, this could be blamed on several things from original model performance to hard layer pruning limits. To better and longer retraining techniques, or local vs global pruning. However, I think in this case the problem is more fundamental to the structural pruning applied. This is reinforced by the observation of a falloff in accuracy before any changes were made to local vs global pruning.

For our method pruning was applied internally in the side branches of the convolutional blocks, this was done to provide higher control over the pruning process. This is in stark contrast to other pruning methods that prune these blocks in parallel removing many filters at a time. Unfortunately, this caused the side effect of pruning taking vastly longer due to all the filter comparisons needed. Additionally, the filters located had less of an impact in the reduction of FLOPs and parameters as they represent a smaller ratio of the overall network.

This is very different to the results from the original research that this method is based on, where it was found pruning these blocks internally preserved performance and reduced inference time [60]. It seems that this is due to the different tasks being performed by the neural networks, in the previous work the flow of data from input to output was very important as the output was an edited image based on the input. In addition, previous work [60], dealt with images of size 1920x1080 these are much larger than the size for image recognition 224x224, this has the side effect that filters pruned in a classification model have vastly less impact on the runtimes of the model than when the model is performing an image-to-image task.

Here it seems that when doing recognition tasks this deliberate preservation of data is not needed and may in fact be detrimental to the neural network's predictions. Unfortunately, this did not show up in any previous tests as this is the first classification residual neural network that this method has been applied to.

7.7 CONCLUSIONS

Overall, the pruning method developed performs the same as, if not better than SOTA methods on smaller neural networks. The method also shows good adaptability to different datasets and models. The method slightly underperforms when compared to the top SOTA method when pruning VGG-16 and vastly underperforms many SOTA methods when pruning Resnet-50.

This process has highlighted a few issues in the literature of pruning neural networks and has also highlighted that each application requires slightly different considerations, when pruning there is not a "one size fits all". All the following influence the results.

- Model architecture
- Model size
- Dataset size
- Dataset shape

In all cases where tested, when models were trained from scratch, with identical model architectures as the final pruned models, they would perform with lower accuracies than the pruned models. This would indicate that WAP performs better when optimising a model than simply reducing the model size manually and retraining.

7.8 FUTURE WORK

These are suggestions for any future development on the pruning algorithm.

- The methodology of pruning convolutional blocks internally for classification networks seems flawed, if the pruning method was applied, instead, in parallel. Results for the current method may improve.
- Hard limits on the number of units pruned in a layer need to be addressed and a more refined version deciding on these values should be included in future versions.
- A method for identifying the most alike filters [91] and combining this with the current method of identifying the least used filters may produce even better results for pruned models.

8 IMPACTS OF PRUNING NEURAL NETWORKS

8.1 PREFACE

Previous sections in this thesis have developed and applied a pruning algorithm to multiple different neural networks and datasets. However now that we have established that learning more about the inner computation of a neural network can aid in the pruning process, we must ask… What has changed?

Especially in situations where models have been pruned, however no accuracy has been lost we could assume that the learnt information has changed. But perhaps this is not the case, perhaps once a solution is found that only consumes a small computational proportion of a neural network the remaining parts do nothing of use? These are the questions we hope to answer with the following tests.

8.2 EXPERIMENTS

Three experiments will be conducted, one will simply look at the changes in the saliency maps of a pruned network and an unpruned network. Although inspecting all the feature maps in our networks would be eye opening, after a certain network size this kind of investigation becomes far too large to draw any meaningful conclusions from. Saliency maps work better in this regard, they show which pixels in the input image have most contributed to the given classification, saliency maps will therefore be test number one.

The second experiment will run a “Xrai test”, this test was devised to identify the sections of the image that matter the most for the classified class using circular regions and outputs a heat map [108]. The heat map can then generate a mask, the mask’s shape can be varied based on the different contributions of the regions of the heat map, for our tests we will use the top 30% of contributing regions.

A test will also be conducted to determine the maximal input for a given output class. This experiment will show the most “cat like cat” or “dog like dog” image, this will hopefully shed light on the knowledge that the pruned model is retaining.

Finally, an inspection of the weights of the networks will be conducted, by inspecting the statistical distribution of weight values we can more deeply understand the optimal structure that the pruning process is trying to achieve.

8.3 LENET-5 MNIST

For these evaluations relevant models have been selected from the previous section’s pruning results. For this first evaluation the unpruned model has been selected and the optimal model from pruning was also selected.

8.3.1 XRai and Gradient Maps

There are many ways we could evaluate these results; however, we will focus on 5 main cases.

- When both networks predict the correct class.
- When both networks predict the wrong class, but the class is identical.
- When the original network is correct, but the pruned network is wrong.
- When the pruned network is correct, but the original network is wrong.

8.3.1.1 Correct Classification

8.3.1.1.1 Xrai Plots

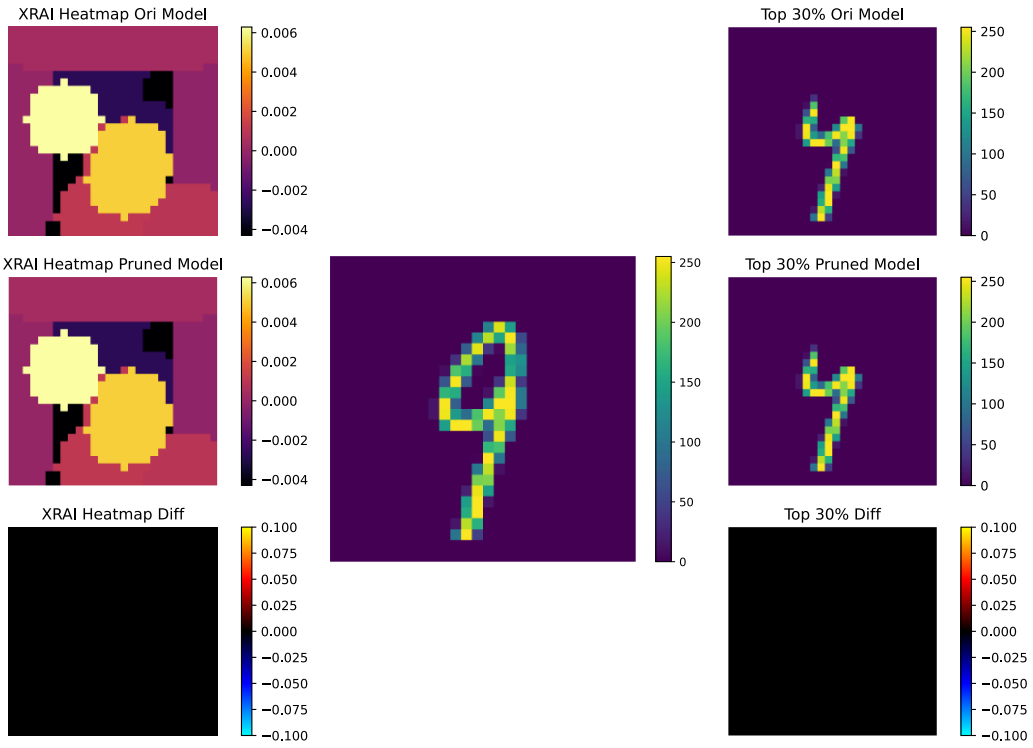


Figure 150: Both models predicting the correct class and the corresponding XRAI and the 30% most important pixel plot

8.3.1.1.2 Saliency Plots

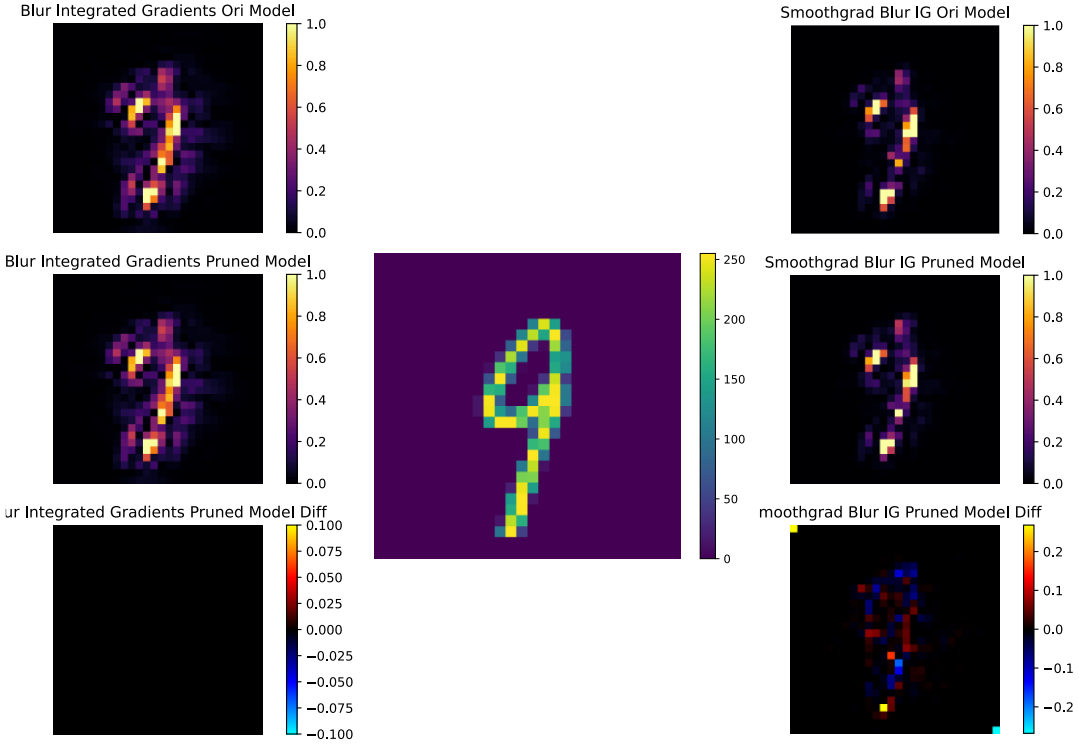


Figure 151: Both models predicting the correct class and the corresponding saliency plots

8.3.1.2 Both Wrong Classification – Same

8.3.1.2.1 Xrai Plots

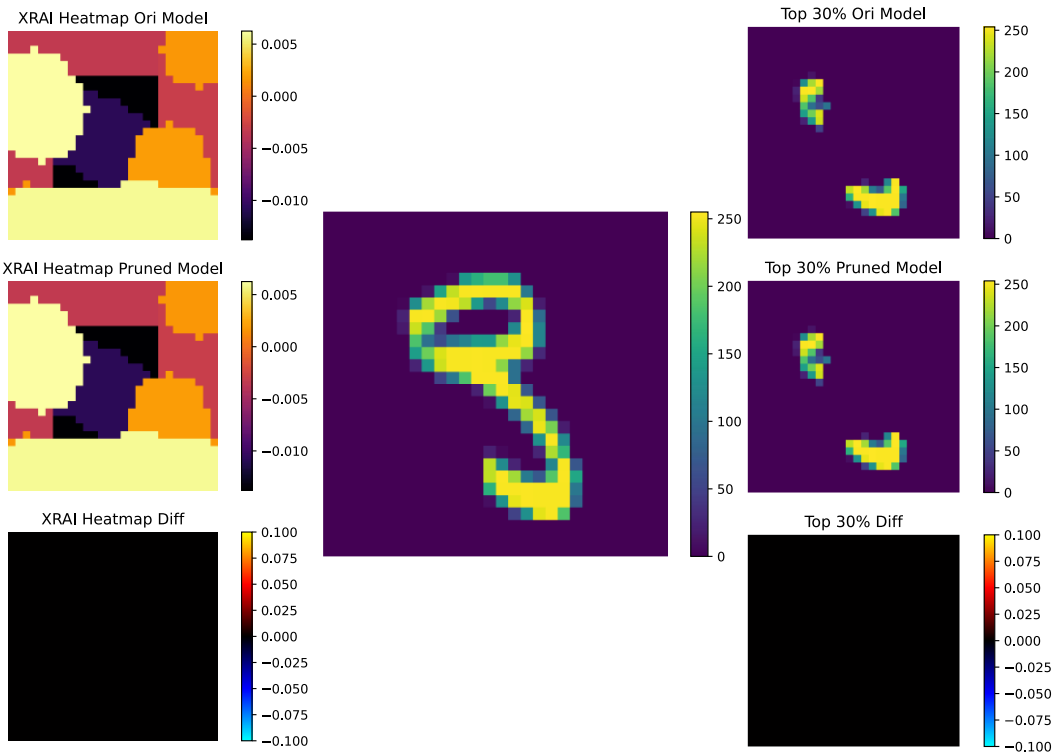


Figure 152: Both models predicting the incorrect class and the corresponding XRAI and the 30% most important pixel plot

8.3.1.2.2 Saliency Plots

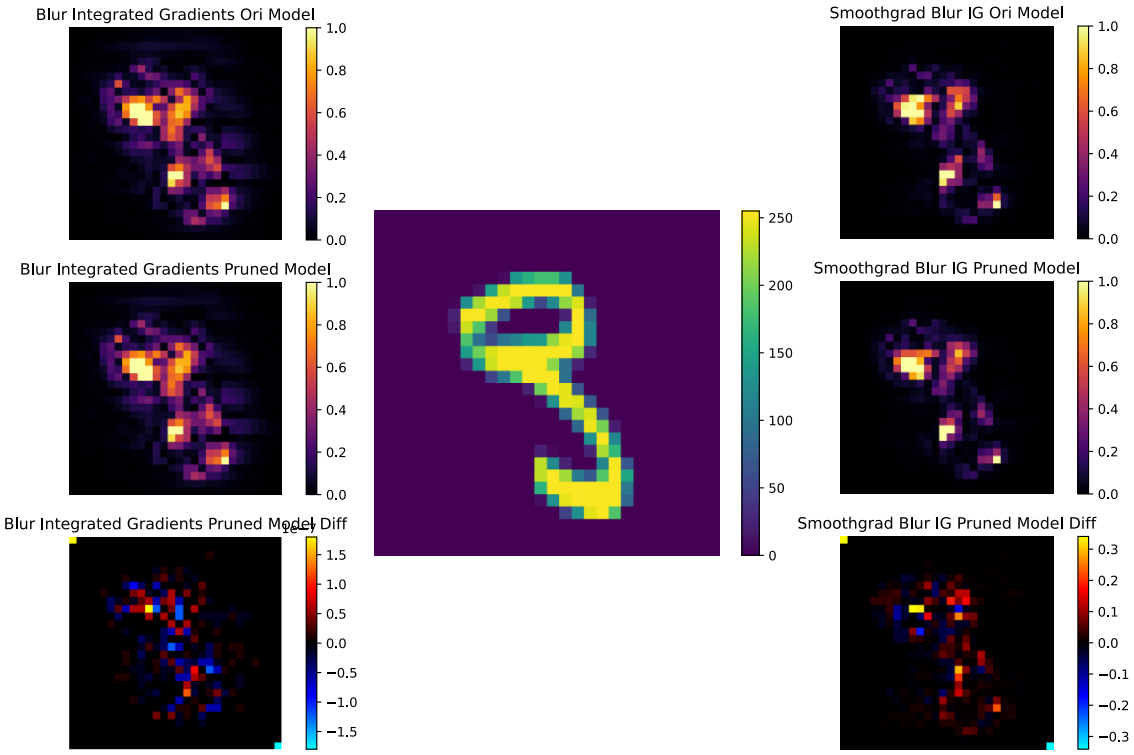


Figure 153: Both models predicting the incorrect class and the corresponding saliency plots

8.3.1.3 Correct Classification for original model

8.3.1.3.1 Xrai Plots

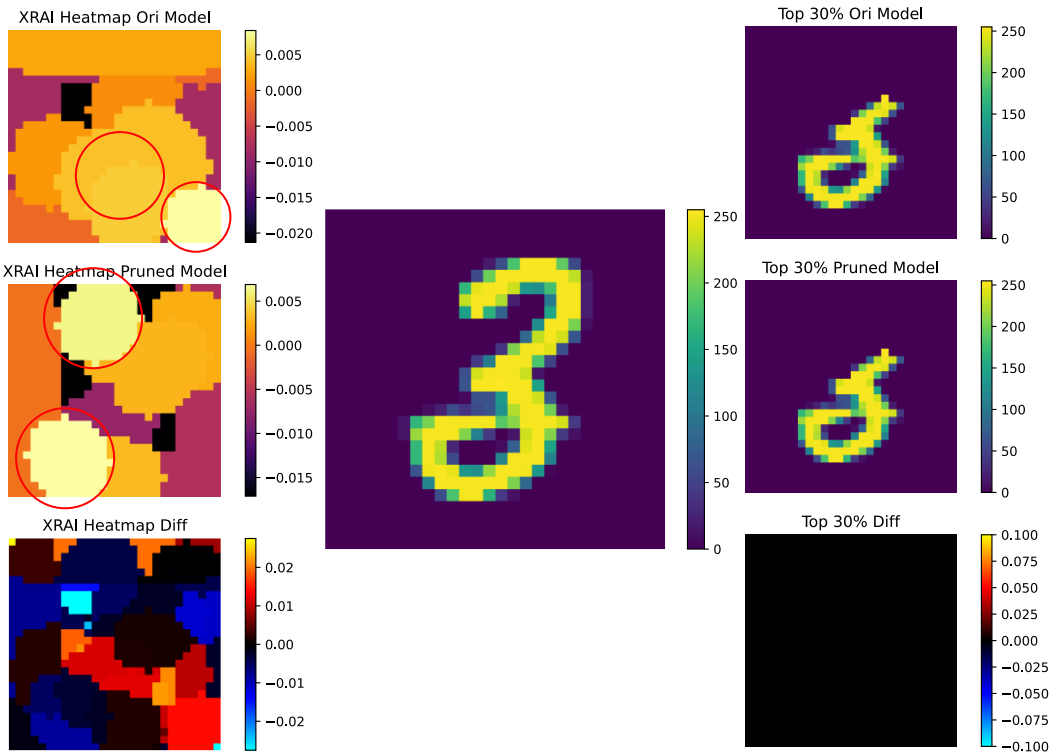


Figure 154: Original model predicting the correct class and the pruned model predicting the incorrect class and corresponding XRAI and the 30% most important pixel plot.

8.3.1.3.2 Saliency Plots

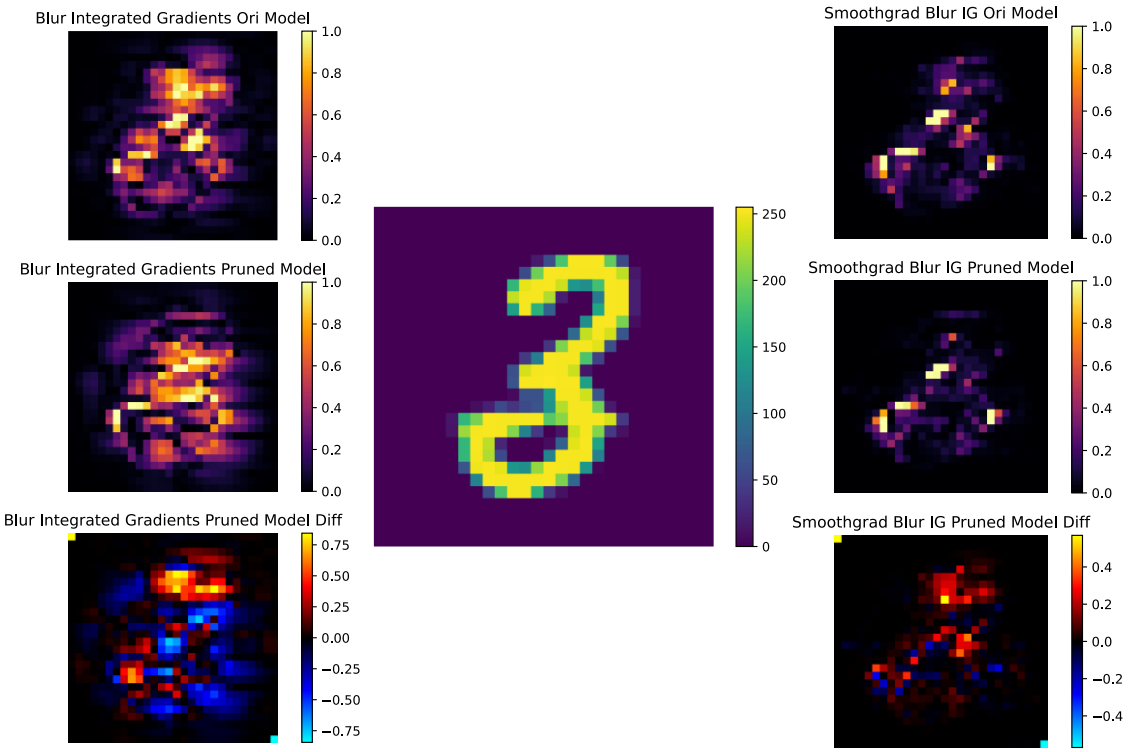


Figure 155: Original model predicting the correct class and the pruned model predicting the incorrect class and corresponding XRAI and the corresponding saliency plots.

8.3.1.4 Correct Classification for pruned model

8.3.1.4.1 Xrai Plots

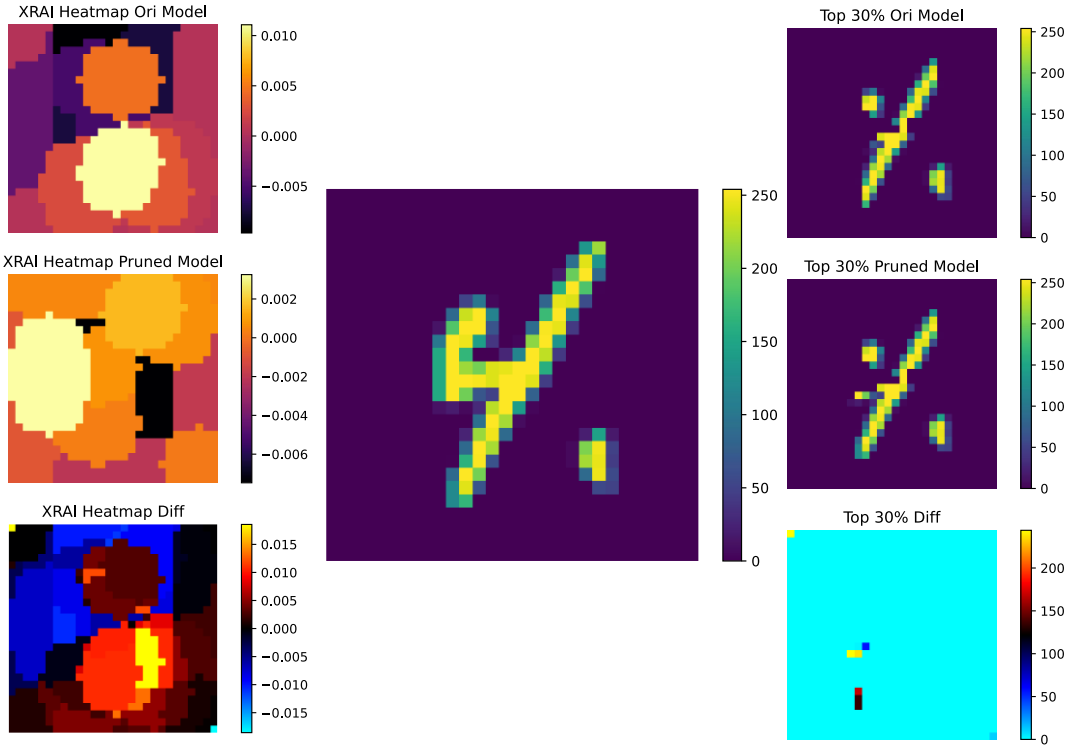


Figure 156: Original model predicting the incorrect class and the pruned model predicting the correct class and corresponding XRAI and the 30% most important pixel plot.

8.3.1.4.2 Saliency Plots

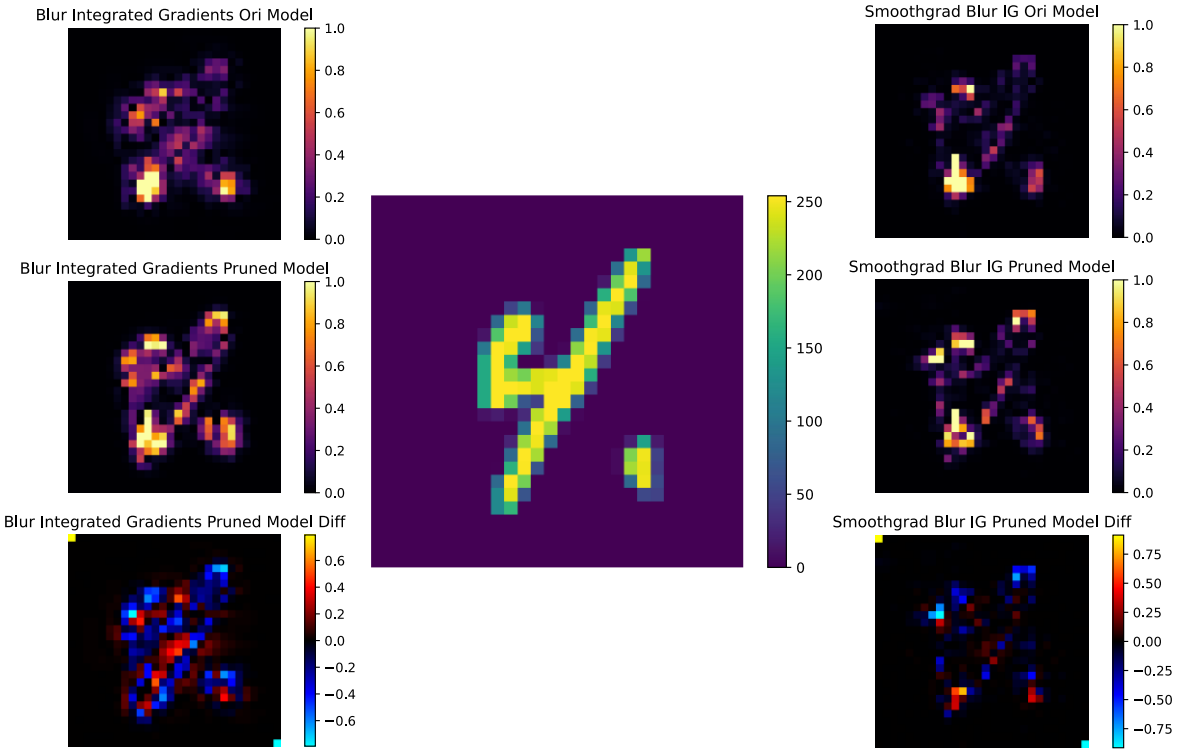


Figure 157: Original model predicting the incorrect class and the pruned model predicting the correct class and corresponding XRAI and the corresponding saliency plots.

8.3.1.5 Discussion

Looking at the first examples when both models predict the correct number, you can see that the models are focusing on identical parts of the image, shown by the XRAI plots Figure 152.

The saliency plots Figure 153 show both blur Integrated gradients and SmoothGrad blur integrated gradients. These are two ways to visualise saliency maps and help with identifying how each model differs. For the first example the gradients are so similar, that to seven decimal places there is no difference in the blur saliency plot. The SmoothGrad plot shows some differences in the tail of the number 9, showing that there are some small differences in how the model is interpreting the image. However, all tests run here make it seem this model's understanding of this number 9 has changed minimally.

The next example shows that when both models are wrong, they are wrong in nearly identical ways, the XRAI plots in Figure 154 show that both models focus on the same parts of the image, however the saliency plots in Figure 155 show more variation. Both the bur and SmoothGrad saliency plots show differences, and these differences are also noticeable even without the difference plots.

This result shows that when the models are incorrect in the same way, they are less similar than when they are correct.

When the original model is correct and the pruned model wrong, the first thing to notice is that the XRAI heatmaps, in Figure 156, are very different from each other. Whilst the combination of these heatmaps does produce the same top 30% of important pixels, the regions of high attention are vastly different, highlighted by red circles.

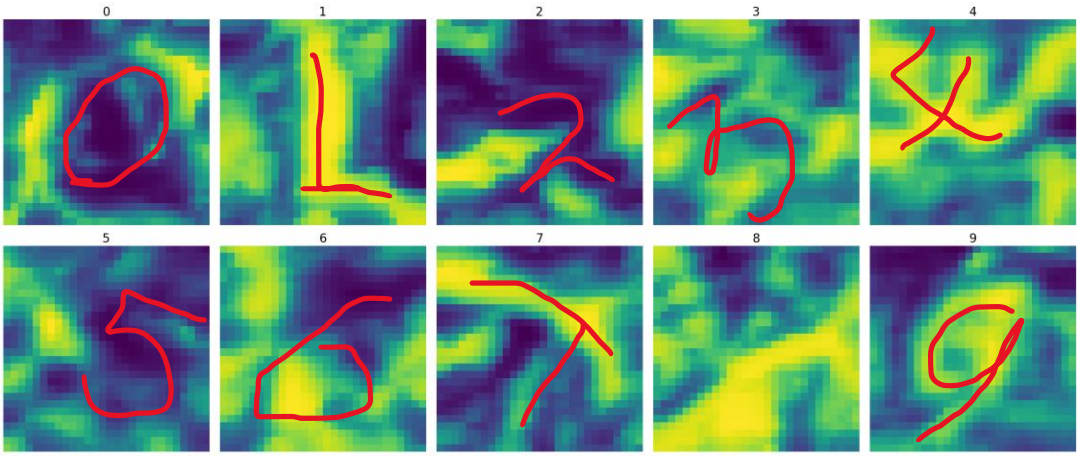
The saliency maps in Figure 157, show that the pruned model has a much broader saliency plot for the blur saliency plot. This shows that the model was widely activated but, did not focus enough on the parts of the image that make up the digit three, and this backs up the findings from the XRAI tests.

When the pruned model is correct and the original model is wrong in Figures 158 and 159, these observations are reversed, with the pruned model now showing a far more focused saliency map, and this makes intuitive sense.

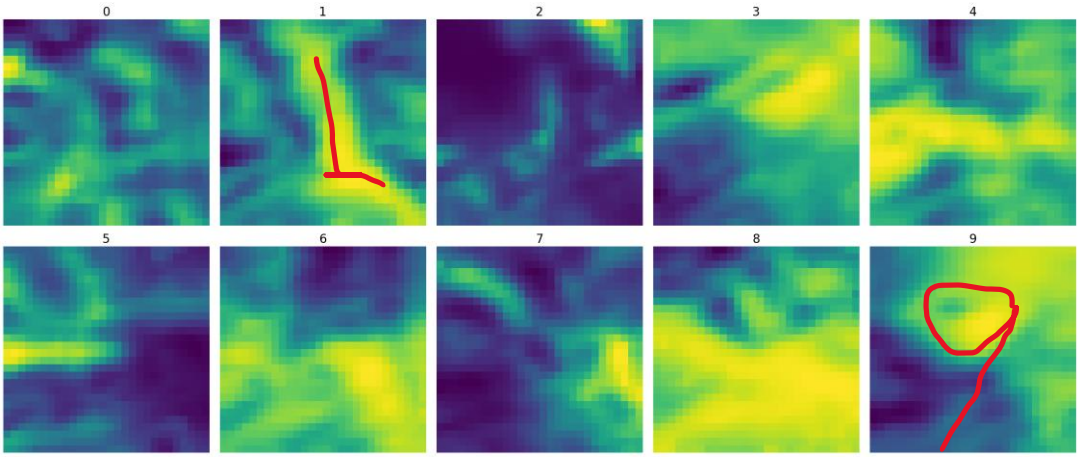
8.3.2 Maximising Neuron Output

In this section we will be looking at checkpoints of the model in the following states: Unpruned, pruning loop 20 and pruning loop 60. This should give a good spread of model performance from slightly pruned to fully pruned.

Unpruned



Pruning Loop 20



Pruning Loop 60

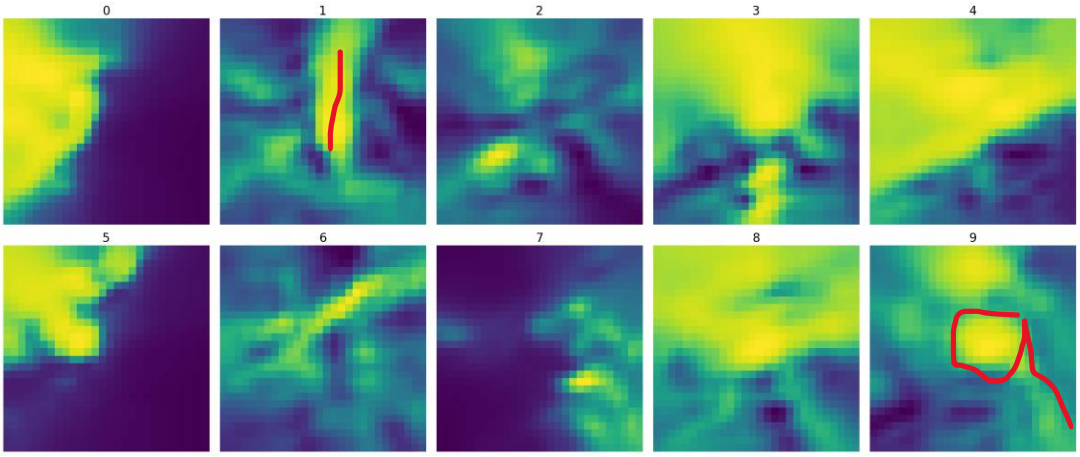


Figure 158: Maximisation maps for all classes for the unpruned model, the model at pruning loop 20 and the model at pruning loop 60.

8.3.2.1 Discussion

These maximisation maps shown in Figure 160 output the result that maximally activates the output neuron for each digit. The results are the inputs that produce these results. Where obvious I have highlighted the input in red to show the shapes for the digits you would expect.

As is clearly visible the structures in these inputs diminish the more the model is pruned, this is because as the model is pruned, the specificity of an individual class is reduced. When no pruning has occurred the exact shape for a single digit class is very well defined. However, as pruning progresses, the size of the network is reduced, and the network holds fewer representations of each digit. This means that this maximisation process is much easier for the optimiser, given the smaller model. Therefore, the input required to produce a high confidence value has a far less meaningful digit structure.

This is an interesting finding, as it suggests that even though the model is still performing at a high level of accuracy, a deeper level of understanding about the classification problem has been forgotten.

8.3.3 Weight Analysis

One very interesting observation from these results is that the sparsity pruning process seems to prioritize dense layers when zeroing out weights in LE-NET5 shown in Figure 161.

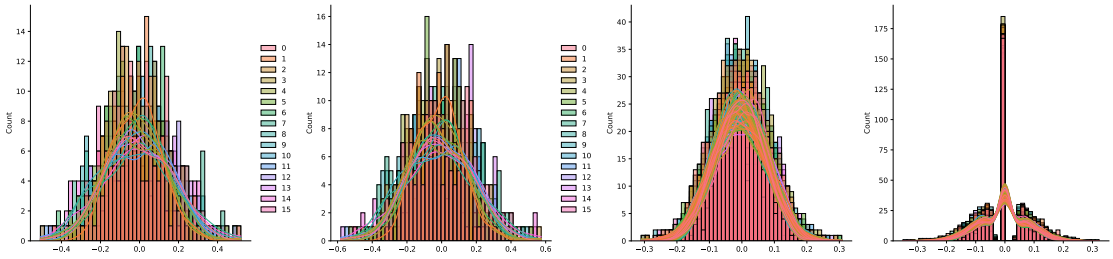


Figure 159: Weight values of all filters individually in the second convolutional layer of the original trained model (left) and pruned original model (centre left) first dense layer of the original trained model (centre right) and pruned original model (right).

This means that inherently dense layers are much easier to prune using the data driven approach developed in this thesis.

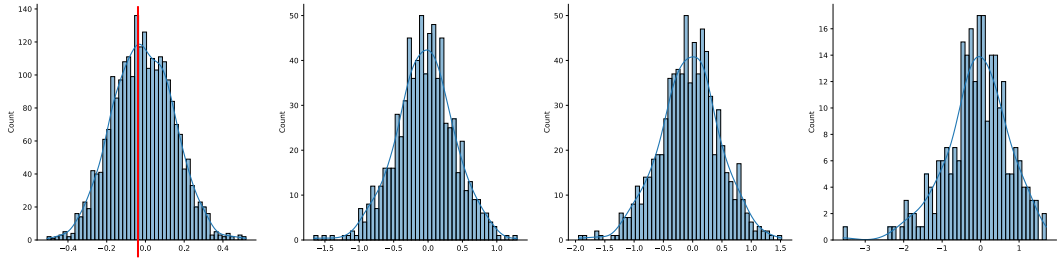


Figure 160: Weight values of all filters summed together in the second convolutional layer of the original trained model (left) pruned model 30 (centre left) pruned model 60 (centre right) and pruned model 90 (right).

As the histograms show in Figure 162 the model generally maintains the same shape in the convolutional layers. However, as the model reaches the best performance to model size ratio the histograms shift, and individual weight values begin to hit extremes. This can also be observed in the range of weight values, here they increase from -0.4 to -1.5, -2 and finally -3.5, and 0.4 to 1 and 1.5. The general shape of the distribution is maintained, but it's stretched over a much wider range as pruning continues.

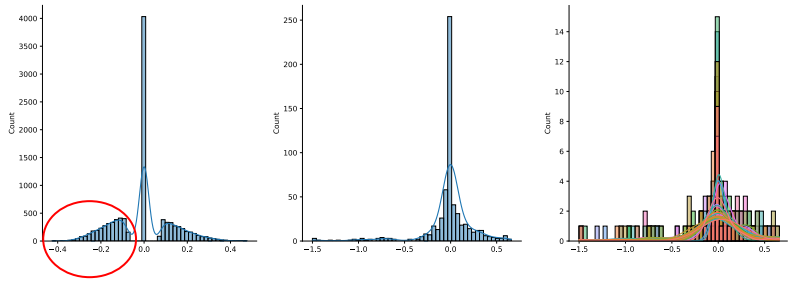


Figure 161: Weight values of all filters summed together in the second dense layer for the original sparsely pruned model (right), pruned model 90 (middle) and the individual filters for pruned model 90.

Figure 163 demonstrates that it's not one filter that shifts the results. In fact, the unsymmetrical and wider range of weight values is displayed even within individual filters. Both results also show that slight offsets in symmetry in the unpruned weights seems to become amplified in the pruned model.

This can be seen in Figure 162, where the original model weights are close to a normal distribution, however slightly shifted to the left of 0 (shown with a red line). This initial condition results in a much more exaggerated shift in the pruned model. This is also shown in Figure 163 highlighted with a circle, indicating where the two lobes of the pruned model are very similar however the left-hand side lobe is slightly larger, resulting in the pruned model once again having weights shifted towards the same direction.

8.4 VGG-16 Birds

8.4.1.1 Correct Classification

8.4.1.1.1 Xrai Plots

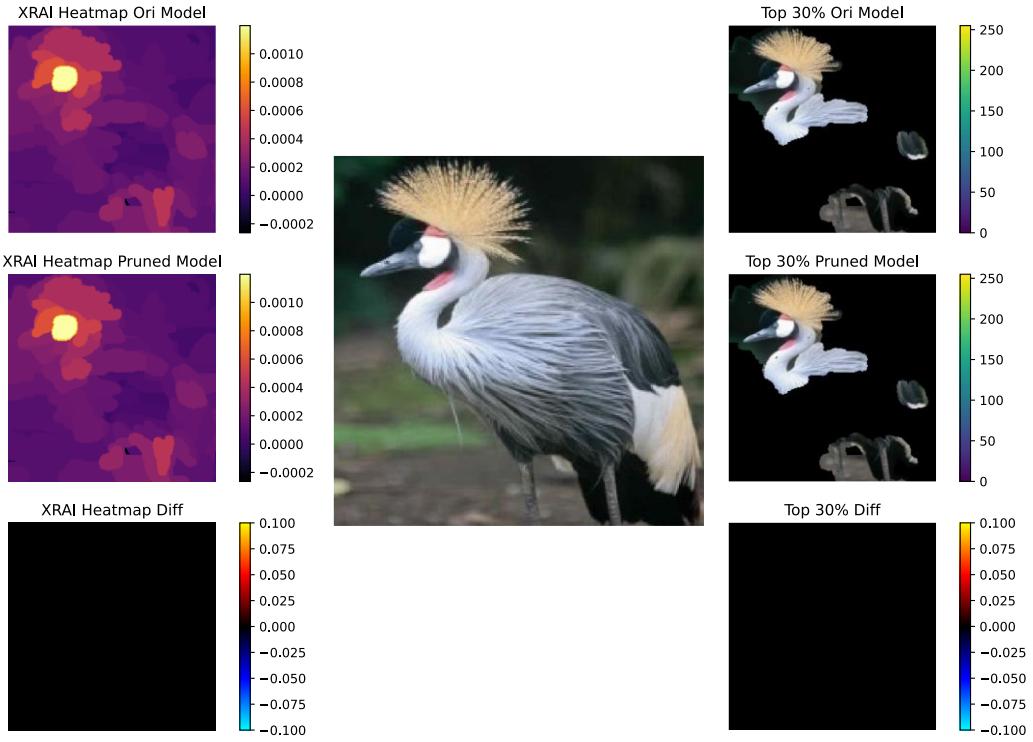


Figure 162: Both models predicting the correct class and the corresponding XRAI and the 30% most important pixel plot.

8.4.1.1.2 Saliency Plots

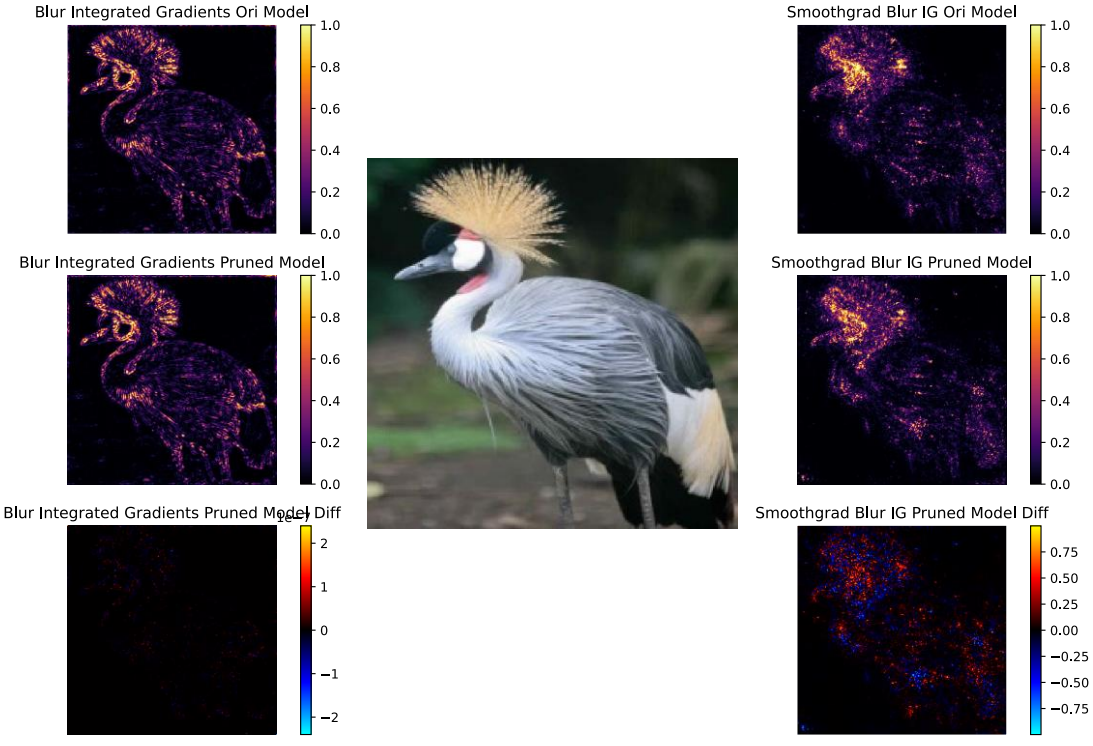


Figure 163: Both models predicting the correct class and the corresponding saliency plots

8.4.1.2 Both Wrong Classification – Same

8.4.1.2.1 Xrai Plots

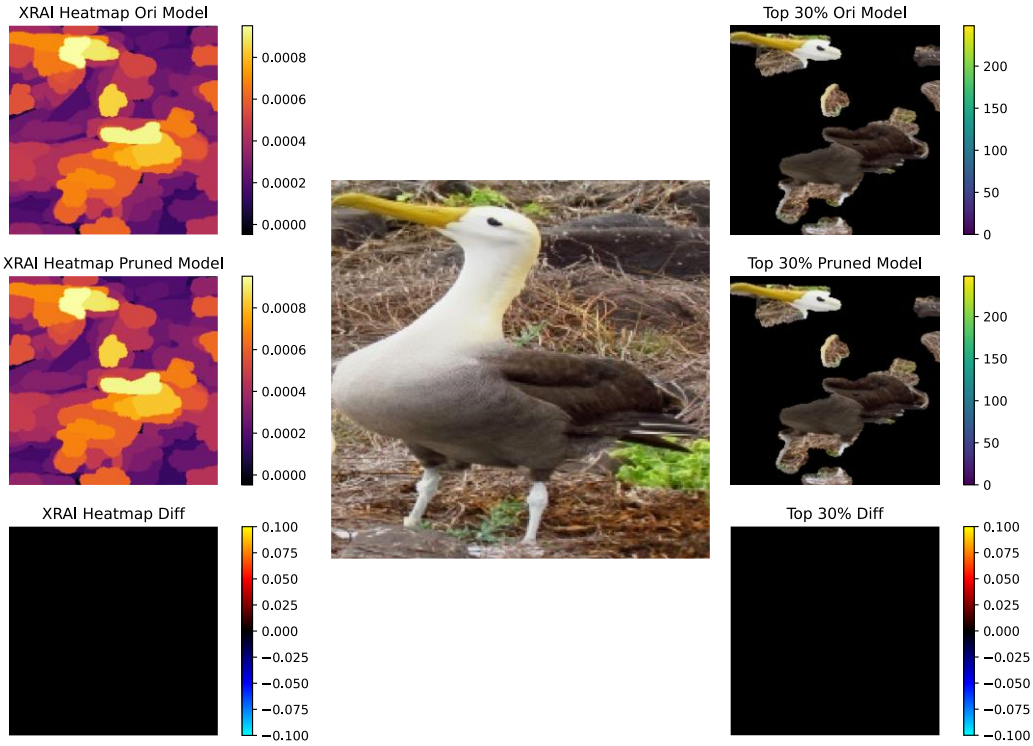


Figure 164: Both models predicting the incorrect class and the corresponding XRAI and the 30% most important pixel plot

8.4.1.2.2 Saliency Plots

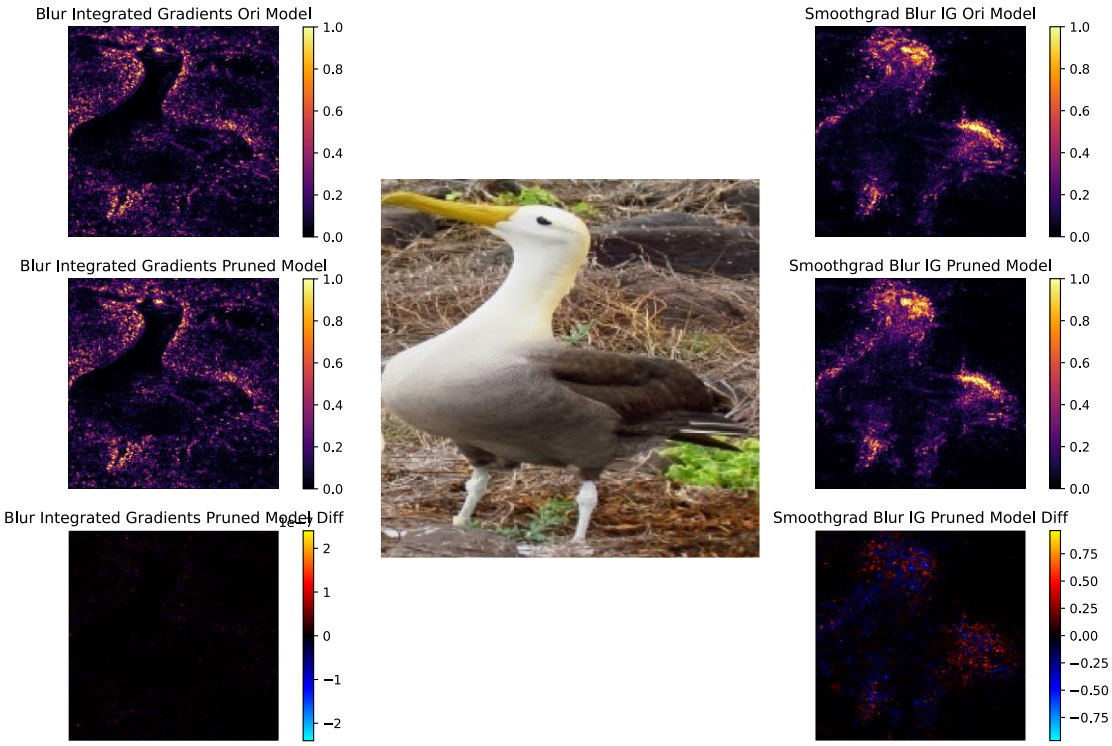


Figure 165: Both models predicting the incorrect class and the corresponding saliency plots

8.4.1.3 Correct Classification for original model

8.4.1.3.1 Xrai Plots

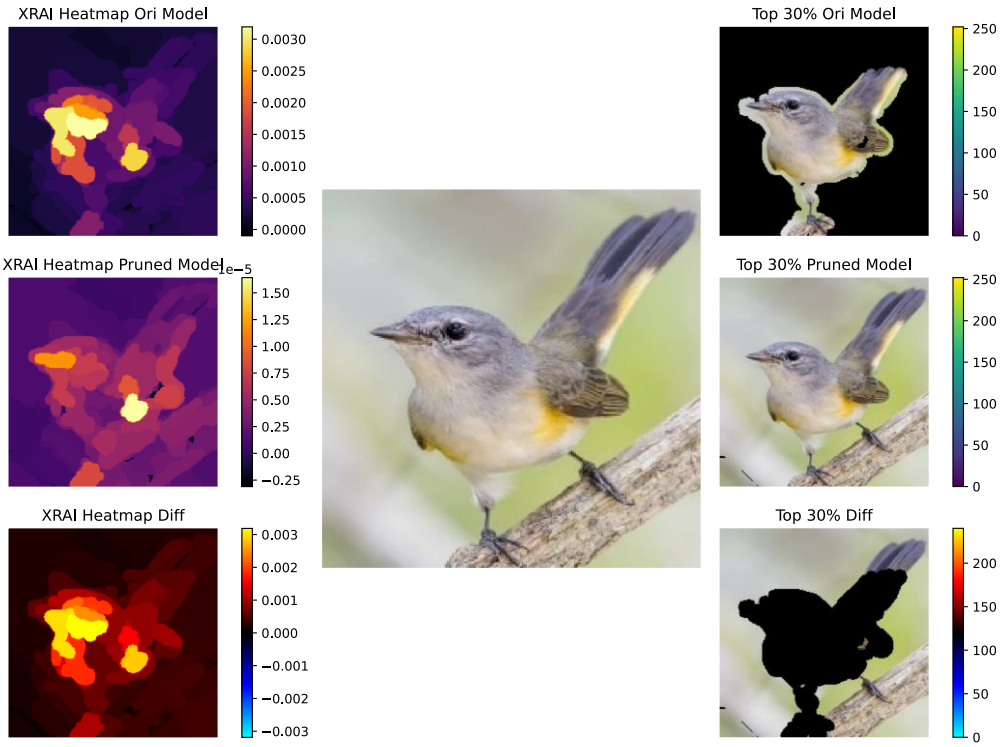


Figure 166: Original model predicting the correct class and the pruned model predicting the incorrect class and corresponding XRAI and the 30% most important pixel plot.

8.4.1.3.2 Saliency Plots

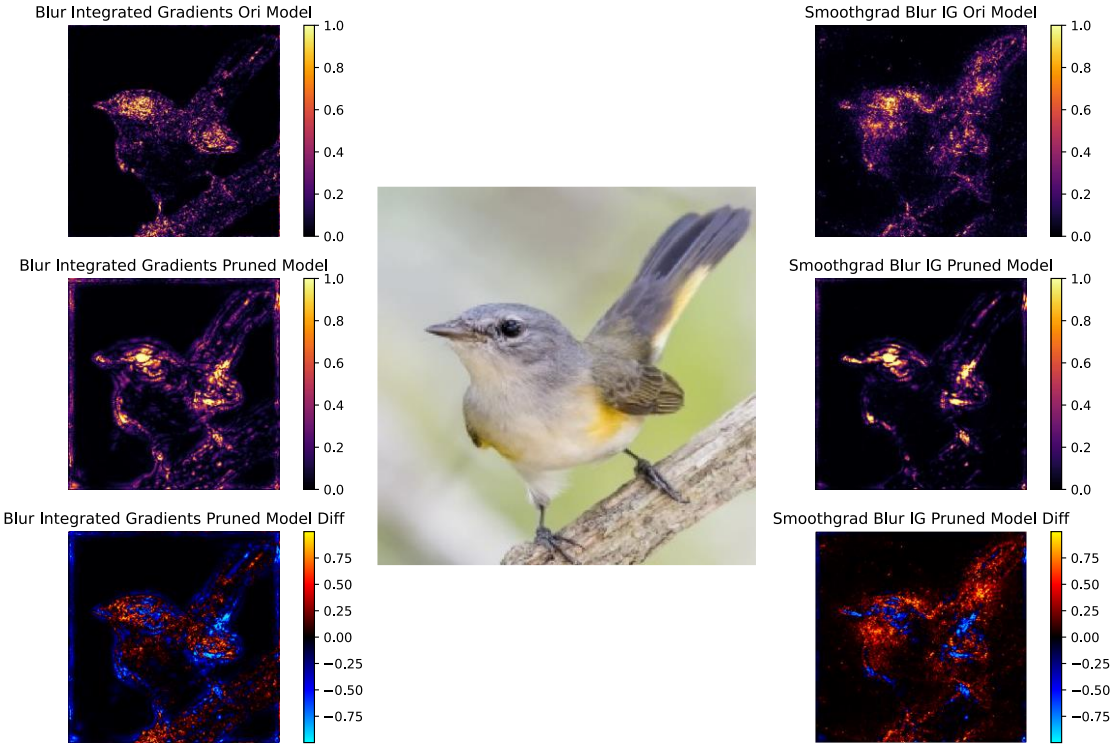


Figure 167: Original model predicting the correct class and the pruned model predicting the incorrect class and corresponding XRAI and the corresponding saliency plots.

8.4.1.4 Correct Classification for pruned model

8.4.1.4.1 Xrai Plots

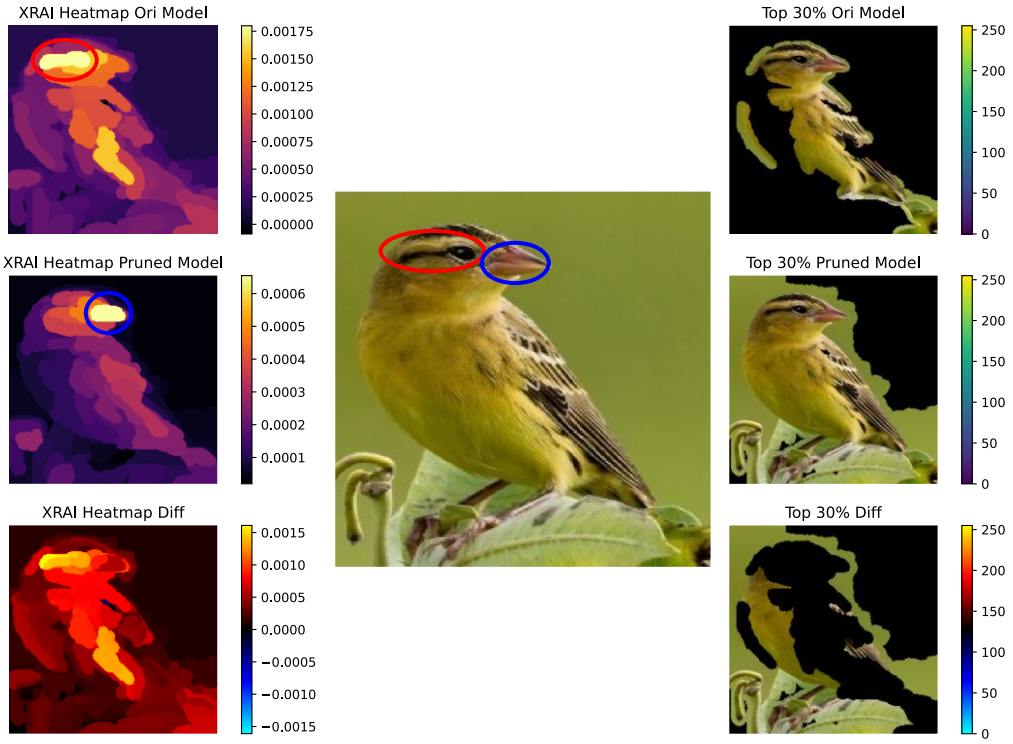


Figure 168: Original model predicting the incorrect class and the pruned model predicting the correct class and corresponding XRAI and the 30% most important pixel plot. Red and blue circles show a change in the models focus.

8.4.1.4.2 Saliency Plots

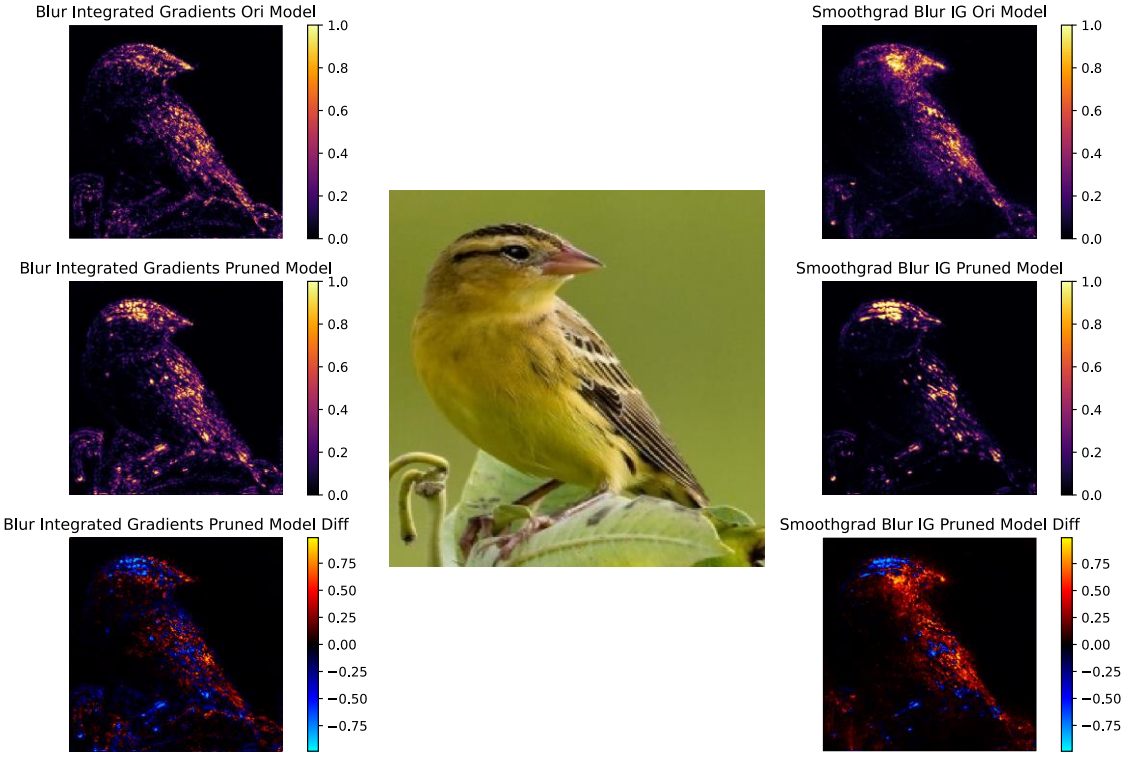


Figure 169: Original model predicting the incorrect class and the pruned model predicting the correct class and corresponding XRAI and the corresponding saliency plots.

8.4.1.5 Discussion

Firstly, focusing on the XRAI plots when the pruned and original model both produce the same correct prediction in Figure 164, the same behaviour is shown as the MNIST results. The models have an identical XRAI plot, and this translates to the same top 30% of pixels being selected too.

Additionally, the saliency plots are also so similar in Figure 165 that they produce no difference plot for the blur integrated gradients, and only have extremely small differences when looking at the SmoothGrad plot.

Focusing on the result where both models are incorrect in Figure 166, the results are very similar to the MNIST visualisations, both models seem to focus on the same parts of the image. Both models are also similar down to the saliency plots again shown in Figure 167, and the same conclusions can be drawn again, when the models are wrong in the same way they are wrong for similar reasons and when they are right, they are right for similar reasons.

In the correct prediction the XRAI plots show how both models are focusing on the bird in the image and specifically the defining features of the bird, in the face, crest on the top on the head, and legs shown in Figure 164. It's also obvious that when the models are incorrect, they have failed to identify key features of the bird, whilst the beak has been identified, the breast of the bird is ignored, and both models have heavily focused on a rock in the background shown in Figure 166. This shows that sometimes it may not be the ability of the model to find the correct bird features, that results in the correct class prediction, but also the ability to exclude non-bird information.

When the original model is right and the pruned model is wrong, the reason is clear, the model has failed to focus on the bird, or put another way, the model has become distracted by the background. This is most clearly seen in the XRAI heatmaps seen in Figure 168, although it is obvious that the pruned model is still generally focused on the bird slightly more than the rest of the image. You can see by the scale on the right of the image that the focus on the bird is orders of magnitude less than the original model.

The saliency maps shown in Figure 167 also show that the original model is more focused on the bird in the image shown by higher values across the whole area the bird takes up. The pruned model does seem to focus specifically on the bird's eye, this may indicate a shortcut that the pruned model has had to rely on to identify birds using fewer features. However as demonstrated with this result, if it is a shortcut, it doesn't always work.

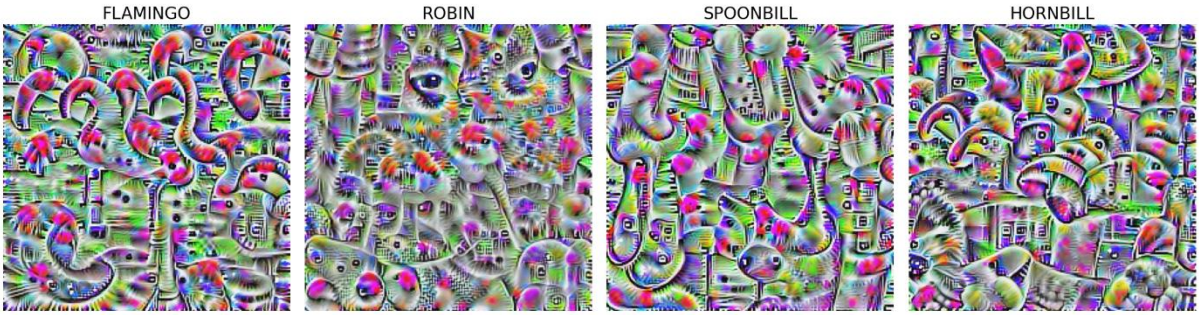
When the pruned model is correct the XRAI plots show in Figure 170 that the pruned model has a far lower value than the original model, meaning that the overall confidence is a lot lower. However, the strongest area of confidence is in the beak, circled in blue. Observing the unpruned model there is a highlighted region around the eye and the feather colouration around the eye highlighted with a red circle. This is another example of a potential shortcut being used, however this time the shortcut works. Additionally, this is an example of feature selection, the model has potentially learnt that it is more efficient to store beak shapes in filters than plumage patterns.

This makes intuitive sense as beaks often share similar general shapes, where small variations indicate big differences in species [109]. This is a much smaller information space to represent than the plumage patterns of all birds. It also makes intuitive sense as to why larger models would not rely solely on beak information, as there is no guarantee that a bird's beak would be visible in an image.

It seems that the pruned model is in this constant battle to maintain the most efficient filters at each pruning step. It also helps to answer the question as to why models that are architecturally identical, cannot be trained from scratch to achieve the same degree of accuracy. Some of the learnt filters must rely on previous filters before they can exist in a helpful way in the model. Smaller models don't have the capacity to learn these more refined "efficient" filters and therefore they cannot perform at the same level. The only way to have these "efficient" filters present in models of smaller sizes is to prune models in a gradual and deliberate fashion like the methods developed in this thesis. Data driven pruning ensures these efficient filters by focusing on filter utilization and removing the ones least used.

8.4.2 Maximising Neuron Output

Example Birds



Original Model

Model 2000

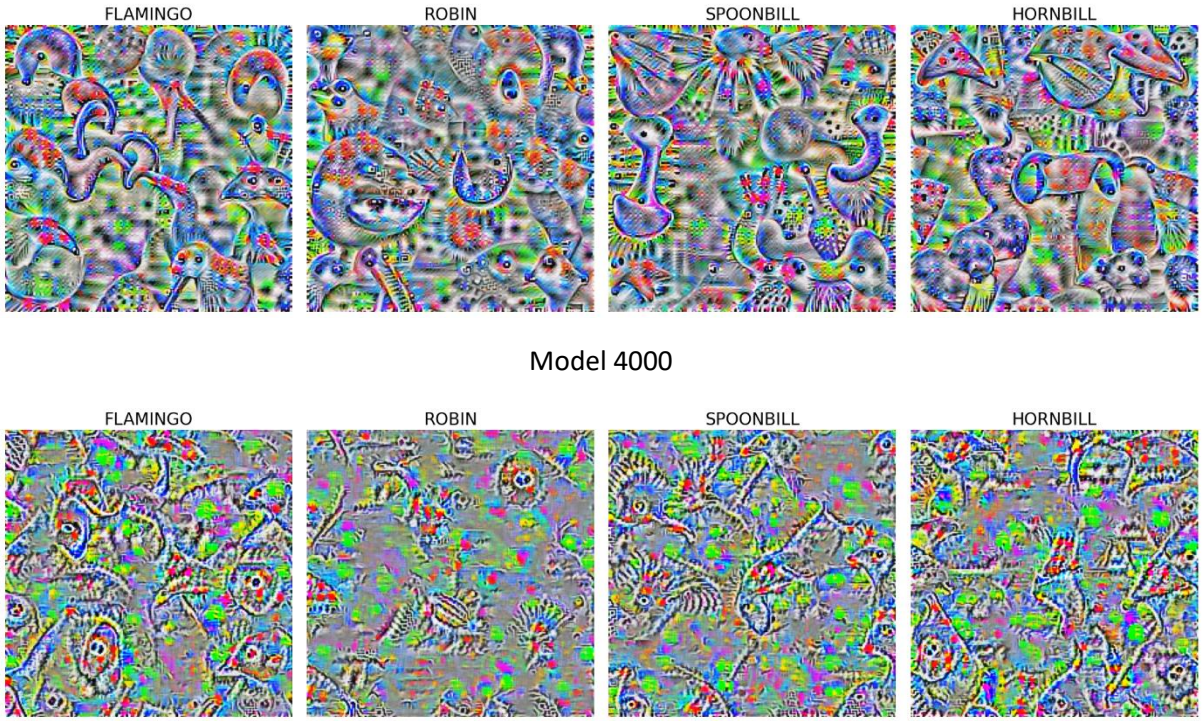


Figure 170: Maximisation maps and example inputs for the classes Flamingo, Robin, Spoonbill and Hornbill for the unpruned model, the model at pruning loop 2000,4000 and 6000.

8.4.2.1 Discussion

The maximisation task was undertaken focusing on 4 classes, Flamingo, Robin, Spoonbill and Hornbill. These classes were specifically selected as 3 represent extremely unique bird species, the fourth class of Robin was selected to represent a less unique species. All results are shown in Figure 172.

Focusing on the results of the original model, shapes, and colours important to the classifications of each species are clearly visible. This can be seen in the flamingo plots, with clear repetitions of the neck and bill throughout the image. The robin plots seem to show a high interest in the eye of the bird, this structure is repeated many times. The Spoonbill plots show many rounded circular shapes with long protrusions indicative of the head and bill arrangement of the species. Finally, the Hornbill once again has many repetitions of the beak and seems to have a few repetitions of the appendages both above and below the head of the bird.

The subsequent maximisation maps clearly show a gradual decline in the structures created through this process. The number of repetitions of the bird parts reduces, and the definition in these structures also decreases. There also seems to be a loss in the number of repeated elements, for instance the Hornbill appendages seem to disappear from the original model, even by model 2000. By model 4000 the flamingo plot has lost most representations of the neck of the flamingo. Interestingly the Robin plot stays the most similar from the original model until model 4000, this could be because the robin is a more “normal” bird, therefore it shares more features with other classes, and as such the internal model representation does not change much pruned model to pruned model.

By model 6000 the representations have become extremely sparse in information, this is caused by the optimisation process finding it easier to produce a “perfect” input. This might seem counter intuitive, but, as a model is pruned and information removed, the number of possible “perfect” inputs increases. This is simply due to the model being smaller, and having less filters and less constrained classes, it simply becomes easier to create the perfect class input.

This decrease in the difficulty needed to optimise for a specific class, results in regions where essentially the input does not matter. This is most pronounced in the Robin plot where 80% of the image looks like noise, the other classes at this stage retain more information in their plots. This is most probably caused by these classes having very specific features, due to the unique visual characteristics of the species.

8.4.3 Weight Analysis

The first analysis was to identify if this model showed the same characteristics as the previous when sparsity pruning is applied, so similarly below are plots of the first convolutional layer and the first dense layer’s weight distribution both before, and after, sparsity pruning was applied Figure 173.

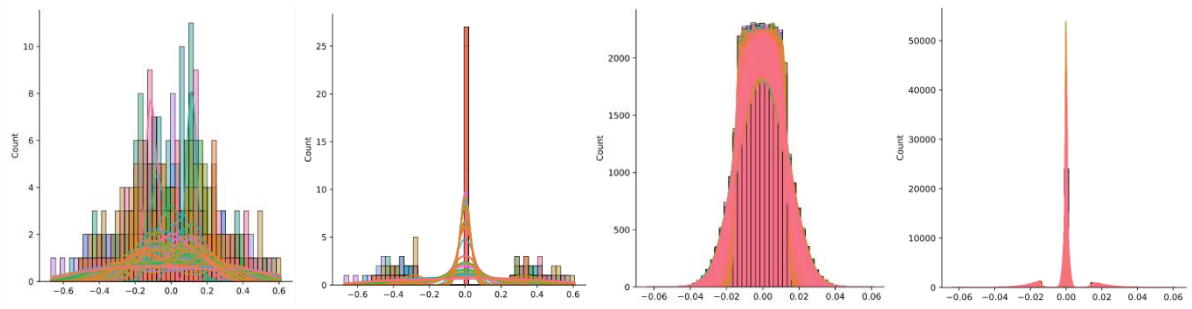


Figure 171: Here the weight distribution of the first convolutional layer of the unpruned model (left) the pruned version of this (left of centre). Similarly, the unpruned first fully connected layer at the end of the network (right of centre) and the pruned version of the (right).

Figure 173 shows that contrary to the previous set of results the convolutional layers seem to prune a similar amount to the fully connected layers. Both seem to double in the number of weights close to zero. This difference is most likely because in the previous weight analysis, the convolutional layers contained a very small proportion of the total weights of the model. Whereas in this model (VGG-16), the number of weights in each type of layer are much more balanced in their distribution, due to there being many more convolutional layers.

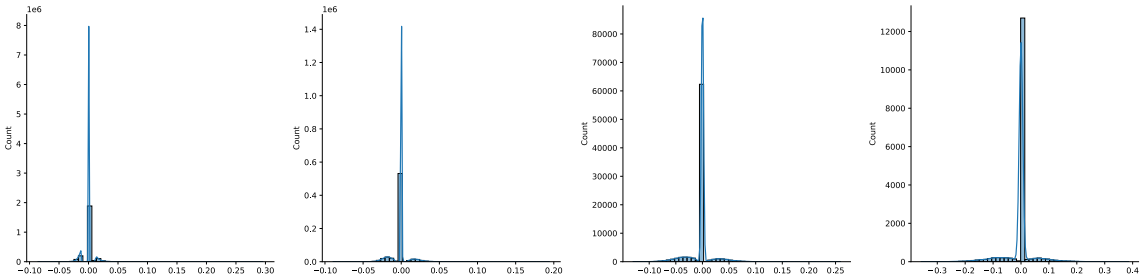


Figure 172: These plots show the weight distribution of the last convolutional layer of the network for the initially pruned model (left) pruned model 3000 (left of middle) pruned model 5500 (right of middle) pruned model 6500 (right).

Figure 174 shows similar characteristics to the previous results, as the pruning progresses the distribution of the weights becomes wider, and weights have more extreme values. Here increasing from -0.1 and 0.3 to -0.35 and 0.4. to show this phenomenon exists throughout the model convolution 10 is also shown in Figure 175.

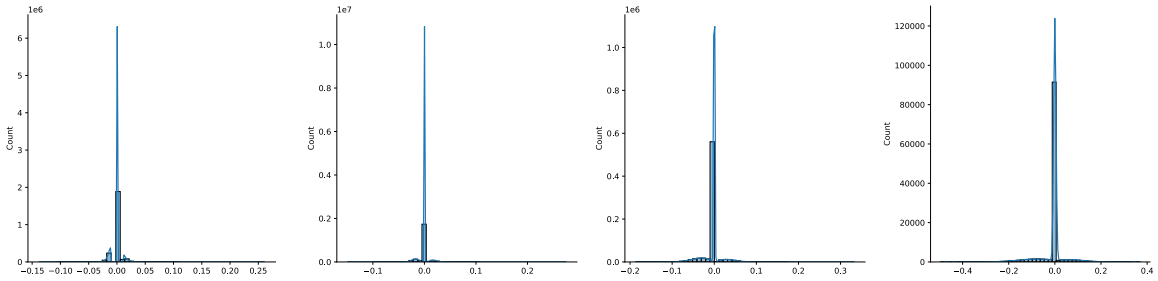


Figure 173: These plots show the weight distribution of convolutional layer 10 of the network for the initially pruned model (left) pruned model 3000 (left of middle) pruned model 5500 (right of middle) pruned model 6500 (right).

Once again, the extremes in Figure 175 increase from -0.15 and 0.25 to -0.45 and 0.4 respectively.

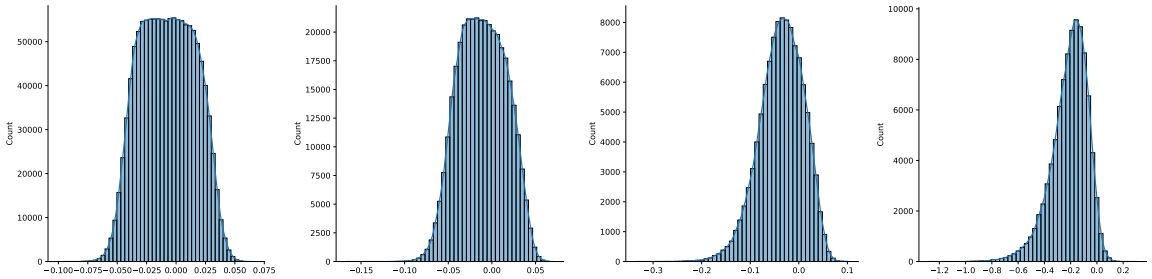


Figure 174: These plots show the weight distribution of the prediction layer of the network for the initially pruned model (left) pruned model 3000 (left of middle) pruned model 5500 (right of middle) pruned model 6500 (right).

Figure 176 shows how the distribution of the last layer of the model changes minimally in its weight distribution between each pruned model, however much like the deeper layers in the model the magnitude of the weights increases to much higher extremes than the original model.

8.5 Robustness Investigation Using t-SNE

T-distributed Stochastic Neighbor Embedding (t-SNE) is a machine learning algorithm for visualization developed by Laurens van der Maaten [161] and Geoffrey Hinton [160]. It is a nonlinear dimensionality reduction technique that is particularly well suited for embedding high-dimensional data into a space of two or three dimensions, which can then be visualized in a scatter plot. Specifically, it models each high-dimensional object by a two- or three-dimensional point in such a way that similar objects are modelled by nearby points and dissimilar objects are modelled by distant points.

The t-SNE algorithm accomplishes this by taking a set of high-dimensional objects and creating a probability distribution over the pairs of objects, such that the probability of picking a pair of objects is high if the objects are similar and low if they are dissimilar. It then creates a similar probability distribution over the pairs of resulting two- or three-dimensional points in the low-dimensional space and it minimizes the Kullback–Leibler divergence [162] between the two distributions with respect to the locations of the points in the map.

While the original algorithm measures the similarity between objects using the Euclidean distance, this can be replaced with any other metric that suits the data.

Because t-SNE can accurately represent the divergence in any embedding space it is possible to represent the embedding space of a neural network.

By comparing the embedding space of an unpruned model to a pruned version of the same model at multiple points we can ascertain how well the model can separate this embedding space given the pruning applied. The following visualisations focus on the model from section 7.6.2, this is an LE-NET-5 model trained on MNIST digits.

Figure 175:t-SNE plots for the original model (top left) the selected pruned model (97) (top right) an over pruned model (127) (bottom)

The results in figure 178 show that the initial model and the model that was selected from the pruning process have similarly sized clusters for each of the digits (this model was selected by analysis of accuracy and FLOPs/Parameters). t-SNE is created in such a way that the location of the groups does not matter, the relationship between the distance of the groups and the size of the groups is most important.

The group for digit 2 (Green) has grown slightly in the pruned model and so has 4 (purple). Overall, the groups show similar separation from each other meaning confusion between numbers has not increased dramatically.

The shapes of the clusters has changed between these two plots and this is expected to happen with a t-SNE plot, the pruned plot shows well defined shapes within groups (seen as an S for digit 7, or a ? for digit 4). The clusters in the plot for the original model are more spread-out, this would indicate that although the pruned model is still grouping similar digits the number of representations for each digit in the embedding space has decreased, resulting in coarser predictions.

Finally, the bottom plot shows what happens if you take pruning to the extreme, this model was the last model before accuracy fell below 2% of the original model. It can be observed that no clusters are represented by polygons anymore, the clusters are more representative of lines. This reflects the embedding space being reduced too far and the model no longer being able to represent enough variety of each digit. Because the clusters have become lines this means they are closer to other clusters which will be causing confusion between numbers being classified.

Finally looking at the groups 7 and 9 (cyan and grey) and 8 and 0 (yellowy green and blue) you can see they are almost touching, there are also multiple classifications that are incorrect at the edges of these clusters (represented by a minority colour in another cluster). It makes sense that these numbers are confused with each other before any others as they share a similar pixel space making them the hardest to differentiate given the reduced embedding space.

This result shows that pruning models will affect the ability of a model to deal with edge cases due to a shrinking embedding space. However, this behaviour does not become a significant issue until pruning is taken to the extreme.

8.6 Conclusions

Both sets of results show that there is an interesting phenomenon with the weights of a pruned model, weights seem to become larger in magnitude and weight values at the extremes seem to become more frequent.

Pruned model weights show that when convolutional layers represent a small proportion of the entire network, they are sparsely pruned less than the fully connected layers. This suggests a reason as to why limiting the number of convolutional layers being pruned, increases the accuracy of pruning overall. This is because, if the model is storing many of it's non-zero values in the convolutional layers, and these layers are removed. Not only does the model have to move the knowledge stored here to another layer, but it also must "wake up" connections in subsequent layers that had previously been turned off. This points towards very disruptive pruning, and as such, perhaps weight values should also be considered when pruning.

Saliency and XRAI maps show that pruned models change as minimally as possible in their internal focus from the original trained model.

Maximisation activation maps show that whilst the internal focus of the model stays similar, the overall representation of a specific class changes drastically. Whilst being pruned the number of definable features a model has about a given class decreases, and obviously if taken to the extreme would eventually reduce to nothing.

8.7 SUGGESTIONS FOR A FUTURE PARADIGM IN NEURAL NETWORKS

This work shows that a neural network requires a certain number of weights close to 0, this is consistent in original models, pruned models and sparsely pruned models. This can be observed either by a normal distribution around 0 in the unpruned case or a large spike in 0 weight values in the sparsely pruned case.

These findings suggest that 0 weight values should be built into model design from the start, deliberately limiting the number of connections to the previous layer to mimic these 0 values, and that this distribution of connections should broadly follow a normal distribution.

By building this into a network before training, the process of identifying filters and neurons and pruning can be circumvented.

This would also mean that the speed up due to pruning is immediately realised in the training process, as the connections are already removed reducing FLOPs and parameters at runtime.

Unfortunately, this method would need a model to be trained from scratch, and that does limit the applicability of this paradigm, however implementation would be much easier.

9 Summary Of Contributions And Future Work

9.1 Summary Of Contributions

9.1.1 MLI

This chapter outlines a process to visualise a simple neural network. These visualisations provide a deeper insight into the inner workings of the neural network, the visualisations indicate that some neurons may be considered redundant.

This chapter also concedes that this kind of visualisation is not possible for larger networks due to their sheer size. Because of this an “importance score” is suggested, this score is used to aid when pruning a neural network. The score is derived from the same data used for the visualisations and indicates the utilisation of a singular neuron or convolutional filter.

9.1.2 JPEG Restoration

In this chapter a network designed to remove video deblocking artefacts was applied to remove JPEG artefacts. This achieved results that were competitive with SOTA artefact removal CNN’s. This chapter showed that the deblocking network’s architecture is effective and transferable to other types of compression algorithms.

This chapter provides strong evidence that models trained on multiple compression strengths are more efficient in terms of the number of parameters. This process also creates a model that is more useful to the end user. There is also strong evidence that each image has an “ideal” decompression level. This “ideal” is not the actual compression level used to create the image and is instead within some range of the original compression level.

Finally, this chapter identifies whole convolutional filters that are universally underutilised by the neural network. This was achieved by manual inspection of the activation maps of the model, initially identified using the neuron importance score developed as a part of MLI.

9.1.3 Pruning a Deblocking Neural Network

This chapter begins by replicating SOTA video deblocking techniques, this is verified through accuracy metrics. The SOTA ML technique is compared against the vanilla VTM specification and was found to be superior. It was found that the ML techniques generally edit more pixel values in the filtering stages, and these pixels are edited at higher magnitudes.

The initial pruning algorithm was developed, experiments proved that sparsity pruning was required to further isolate neurons making it possible with the aid of structural pruning to guide the selection and removal of neurons from a neural network. This reduced run times and memory usage.

It was found that pruning the main branch of the network heavily reduced the performance of the filtering, because of this a sophisticated structural pruning technique was developed. This technique allowed for a large reduction of FLOPS and Parameters but preserved the performance of the network.

Ablation studies showed that a model that had been manually reduced in size could not reach the same performance as the original model and therefore also the pruned model.

This work vastly improved upon the SOTA ML deblocking technique, reducing inference time and VRAM usage significantly and was published in ISCAS 2022.

9.1.4 Pruning Generative Adversarial Networks

Pruning was applied to a GAN based model, aiming at comparing datasets against each other using the same model architecture for training. Overall, pruning was more successful when datasets were configured in such a way to produce a real-world image.

This suggests one of three things, either the metrics to measure the quality of real-world images are insufficient, Pix2Pix has a bias towards generating real-world images or real-world images are easier to produce than segmentation maps.

This section demonstrated that pruning is highly dependent on the dataset used to train the model. It also showed that reversing the modality of a dataset does not result in an equally complex task, each dataset should be considered individually and at length when pruning neural networks.

Additionally, the pruning algorithm was expanded, pruning based on a schedule was explored and found to be detrimental to the pruning process. This was because it was too aggressive and removed fundamental components of the neural network too early in the pruning process.

When training GAN networks mode collapse is a big issue, however experiments seem to show that a model can recover from mode collapse by undergoing a sparsity pruning step. Additionally, whilst some model diversity was observed due to differences dataset complexity, all models seemed to conform to a general “hourglass” shape. The hourglass shape is in reference to the number of filters in a layer. This is a different structure than currently proposed GAN models and perhaps hints at a better architecture for said neural networks.

9.1.5 Pruning Classification Networks

Finally, the efficacy of the pruning method needed to be compared against other SOTA pruning algorithms. To achieve this the pruning algorithm was tested on a comprehensive list of neural networks and datasets used in the field of neural network pruning.

Additionally, the algorithm was improved again. Sophisticated ways of simulating input data were added along with the ability to prune globally or locally. Noise-based pruning was also added and the role that optimisers have when training and pruning was explored. These additions finalised the pruning algorithm and it was named “Weight Action Pruning”, this was used to conduct all experiments in this section.

Issues were found with the proposed structural pruning of ResNet-50. This was due to the assumption that classifications networks would benefit from the same kind of structural pruning applied to the deblocking network in previous chapters, this was not the case. The difference in the dataset’s dimensionally is primarily to blame for this issue, WAP could be adapted to make exceptions for such cases.

Overall, when compared against SOTA methods the pruning algorithm achieves the lowest FLOP metrics when pruning LE-NET 300 and 5. Whilst also achieving the lowest number of parameters on LE-NET 5. Pruning VGG-16 shows comparable results when compared on the number of parameters but falls short on the number of FLOPS. ResNet-50 prunes extremely poorly, mainly due to the phenomena explained above.

As part of this work another network structure was discovered from the pruned model. This architecture was a VGG-16 structure but contained multiple “funnels”, or multiple layers of deeper compression. This achieved through an increase in the number of filters. This occurred at each subsequent layer before being seemly being reset every 3 to 4 layers and the process repeating.

Finally, a case study was performed for the BBC where a transfer learning step was added to the pruning process to help create a customised, prunable model for any dataset. This was envisioned to initially aid with metadata tagging of birds. However, this method could be used to help identify any number of objects in the BBC archive, in an efficient and controlled manner.

9.1.6 Impacts of Neural Network Pruning

The final chapter asks, “what is lost?” when pruning neural networks. The results initially suggest, not much! Models seem to change their internal hidden functions minimally; this is proved by inspecting saliency and XRAI maps of pruned and unpruned models.

However, whilst the internal functions used to identify an animal or number seem to remain similar, the overall “knowledge” in the system decreases. The pruning process selectively removes filters that do not affect accuracy. This work therefore provides evidence that pruning does indeed decrease the “knowledge” in the network overall, but redundant “knowledge” is targeted before all else, to preserve accuracy.

Finally, weight distributions for pruned networks are inspected, and it was found that the distributions stay similar in shape throughout pruning. However, as pruning progresses weights become spread across a wider range of values, and any small differences in the symmetry of the weight distributions is exacerbated.

Of note, values near zero are always the most prevalent in every step of the pruning process. This indicates a fundamental need for a neural network to have a majority of zero weight values irrelevant of the model’s size. This might suggest that models should have deliberately partially connected neurons throughout the model, to simulate zero weight values and provide a FLOP/Parameter reduction inherently before any training occurs.

9.2 Future Work

I believe that the best direction for future research should be based on the findings of the final sections of this thesis, that indicate, neural networks require a percentage of 0 weight connections to function.

Additionally, I believe that revisiting the pruning of ResNet50, but pruning in a parallel nature will yield much better results for WAP. Also adding the functionality to search for “similar” filters and pruning based on this will help with WAP’s efficacy.

Finally, testing WAP on other well established image translation tasks, like super-resolution, colourisation, and frame interpolation. And perhaps even newer tasks like text to image generation would be highly interesting and produce fruitful research.

10 BIBLIOGRAPHY

[1] Benenson, R., Petti, S., Fraichard, T. and Parent, M., 2008. Towards urban driverless vehicles.

[2] Silver, D., Huang, A., Maddison, C.J., Guez, A., Sifre, L., Van Den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M. and Dieleman, S., 2016. Mastering the game of Go with deep neural networks and tree search. nature, 529(7587), p.484.

[3] Arulkumaran, K., Cully, A. and Togelius, J., 2019. Alphastar: An evolutionary computation perspective. arXiv preprint arXiv:1902.01724.

[4] Wachter, S., Mittelstadt, B. and Russell, C., 2017. Counterfactual Explanations without Opening the Black Box: Automated Decisions and the GPDR. Harv. JL & Tech., 31, p.841.

[5] Molnar, C., 2019. Interpretable machine learning.

[6] Quinlan, J.R., 1987. Simplifying decision trees. International journal of man-machine studies, 27(3), pp.221-234.

[7] Gall, R. (2019). Machine Learning Explainability vs Interpretability: Two concepts that could help restore trust in AI - KDnuggets. Available at: https://www.kdnuggets.com/2018/12/machine-learning-explainability-interpretability-ai.html.

[8] Coglianese, C. and Lehr, D., 2016. Regulating by robot: Administrative decision making in the machine-learning era. Geo. LJ, 105, p.1147.

[9] Khalfa, Jean (ed.) (1994). What is Intelligence?. Cambridge University Press.

[10] Patel, K., Smith, B.C. and Rowe, L.A., 1993, September. Performance of a software MPEG video decoder. In Proceedings of the first ACM international conference on Multimedia (pp. 75-82).

[11] Sze, V., Budagavi, M. and Sullivan, G.J., 2014. High efficiency video coding (HEVC). In Integrated circuit and systems, algorithms and architectures (Vol. 39, p. 40). Berlin, Germany: Springer.

[12] Hu, Y., Yang, W. and Liu, J., 2020, April. Coarse-to-fine hyper-prior modeling for learned image compression. In Proceedings of the AAAI Conference on Artificial Intelligence (Vol. 34, No. 07, pp. 11013-11020).

[13] M. Wang, S. Wan, H. Gong and M. Ma, "Attention-Based Dual-Scale CNN In-Loop Filter for Versatile Video Coding," in IEEE Access, vol. 7, pp. 145214-145226, 2019.

[14] Ogura, M. (2019). Uncovering what neural nets “see” with FlashTorch. Available at: https://towardsdatascience.com/feature-visualisation-in-pytorch-saliency-maps-a3f99d08f78a.

[15] Bradford, J.P., Kunz, C., Kohavi, R., Brunk, C. and Brodley, C.E., 1998. Pruning decision trees with misclassification costs. In Machine Learning: ECML-98: 10th European Conference on Machine Learning Chemnitz, Germany, April 21–23, 1998 Proceedings 10 (pp. 131-136). Springer Berlin Heidelberg

[16] Quinlan, J.R., 1987. Simplifying decision trees. International journal of man-machine studies, 27(3), pp.221-234.

[17] Che, Z., Purushotham, S., Khemani, R. and Liu, Y., 2015. Distilling knowledge from deep networks with applications to healthcare domain. arXiv preprint arXiv:1512.03542.

[18] Marco Tulio Ribeiro, Sameer Singh and Carlos Guestrin. “Why should I trust you? Explaining the predictions of any classifier”. In: Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. ACM. 2016, pp. 1135–1144.

[19] Lundberg, S.M. and Lee, S.I., 2017. A unified approach to interpreting model predictions. In Advances in Neural Information Processing Systems (pp. 4765-4774).

[20] Verma, Sahil, Varich Boonsanong, Minh Hoang, Keegan E. Hines, John P. Dickerson, and Chirag Shah. "Counterfactual explanations and algorithmic recourses for machine learning: A review." arXiv preprint arXiv:2010.10596 (2020).

[21] Alaa, A.M. and van der Schaar, M., 2019. Demystifying Black-box Models with Symbolic Metamodels. In Advances in Neural Information Processing Systems (pp. 11304-11314).

[22] Hoefler, T., Alistarh, D., Ben-Nun, T., Dryden, N. and Peste, A., 2021. Sparsity in deep learning: Pruning and growth for efficient inference and training in neural networks. The Journal of Machine Learning Research, 22(1), pp.10882-11005.

[23] Tartaglione, E., Bragagnolo, A., Odierna, F., Fiandrotti, A. and Grangetto, M., 2021. Serene: Sensitivity-based regularization of neurons for structured sparsity in neural networks. IEEE Transactions on Neural Networks and Learning Systems, 33(12), pp.7237-7250.

[24] Anwar, S., Hwang, K. and Sung, W., 2017. Structured pruning of deep convolutional neural networks. ACM Journal on Emerging Technologies in Computing Systems (JETC), 13(3), pp.1-18.

[25] Bross, B., Wang, Y.K., Ye, Y., Liu, S., Chen, J., Sullivan, G.J. and Ohm, J.R., 2021. Overview of the versatile video coding (VVC) standard and its applications. IEEE Transactions on Circuits and Systems for Video Technology, 31(10), pp.3736-3764.

[26] Kilpeläinen, M., Nurminen, L. and Donner, K., 2011. Effects of mean luminance changes on human contrast perception: Contrast dependence, time-course and spatial specificity. PLoS One, 6(2), p.e17200.

[27] Atchison, D.A. and Smith, G., 2000. Optics of the human eye. Butterworth-Heinemann.

[28] De La Rocha Gomes-Arevalillo, A., 2017. Investigating the Adaptive Loop Filter in Next Generation Video Coding.

[29] Benjamin Bross, 2020. VVC tutorial at ICME 2020 together. Available at: https://www.slideshare.net/MathiasWien/vvc-tutorial-at-icme-2020-together-with-benjamin-bross

[30] Redmon, J., Divvala, S., Girshick, R. and Farhadi, A., 2016. You only look once: Unified, real-time object detection. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 779-788).

[31] Shi, W., Caballero, J., Huszár, F., Totz, J., Aitken, A.P., Bishop, R., Rueckert, D. and Wang, Z., 2016. Real-time single image and video super-resolution using an efficient sub-pixel convolutional neural network. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 1874-1883).

[32] Richardson, I.E., 2004. H. 264 and MPEG-4 video compression: video coding for next-generation multimedia. John Wiley & Sons.

[33] Bao, W., Lai, W.S., Ma, C., Zhang, X., Gao, Z. and Yang, M.H., 2019. Depth-aware video frame interpolation. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (pp. 3703-3712).

[34] Ma, D., Zhang, F. and Bull, D.R., 2020. MFRNet: a new CNN architecture for post-processing and in-loop filtering. IEEE Journal of Selected Topics in Signal Processing, 15(2), pp.378-387.

[35] Song, X., Yao, J., Zhou, L., Wang, L., Wu, X., Xie, D. and Pu, S., 2018, October. A practical convolutional neural network as loop filter for intra frame. In 2018 25th IEEE International Conference on Image Processing (ICIP) (pp. 1133-1137). IEEE.

[36] Jia, C., Wang, S., Zhang, X., Wang, S., Liu, J., Pu, S. and Ma, S., 2019. Content-aware convolutional neural network for in-loop filtering in high efficiency video coding. IEEE Transactions on Image Processing, 28(7), pp.3343-3356.

[37] C. Dong, C. C. Loy, K. He, and X. Tang, ‘‘Learning a deep convolutional network for image super-resolution,’’ in Proc. Eur. Conf. Comput. Vis., 2014, pp. 184–199.

[38] E. Agustsson and R. Timofte, ‘‘Ntire 2017 challenge on single image super-resolution: Dataset and study,’’ in Proc. IEEE Conf. Comput. Vis. Pattern Recognit. Workshops, vol. 3, Jul. 2017, pp. 126–135. Accessed: Oct. 31, 2018. [Online]. Available: https://data.vision.ee. ethz.ch/cvl/DIV2K/

[39] Lehtinen, J., Munkberg, J., Hasselgren, J., Laine, S., Karras, T., Aittala, M. and Aila, T., 2018. Noise2Noise: Learning image restoration without clean data. arXiv preprint arXiv:1803.04189.

[40] Pan, J., Dong, J., Liu, Y., Zhang, J., Ren, J., Tang, J., Tai, Y.W. and Yang, M.H., 2020. Physics-based generative adversarial models for image restoration and beyond. IEEE transactions on pattern analysis and machine intelligence, 43(7), pp.2449-2462.

[41] Zamir, S.W., Arora, A., Khan, S., Hayat, M., Khan, F.S., Yang, M.H. and Shao, L., 2021. Multi-stage progressive image restoration. In Proceedings of the IEEE/CVF conference on computer vision and pattern recognition (pp. 14821-14831).

[42] Blog, Netflix Technology (2016-06-06). "Toward A Practical Perceptual Video Quality Metric". Netflix TechBlog. Retrieved 2017-07-15.

[43] Olah, et al., "The Building Blocks of Interpretability", Distill, 2018. Available at: https://distill.pub/2018/building-blocks/

[44] Wallace, G.K., 1992. The JPEG still picture compression standard. IEEE transactions on consumer electronics, 38(1), pp.xviii-xxxiv.

[45] Dong, C., Deng, Y., Loy, C.C. and Tang, X., 2015. Compression artefacts reduction by a deep convolutional network. In Proceedings of the IEEE international conference on computer vision (pp. 576-584).

[46] Tai, Y., Yang, J., Liu, X. and Xu, C., 2017. Memnet: A persistent memory network for image restoration. In Proceedings of the IEEE international conference on computer vision (pp. 4539-4547).

[47] Zhang, Y., Li, K., Li, K., Zhong, B. and Fu, Y., 2019. Residual non-local attention networks for image restoration. arXiv preprint arXiv:1903.10082.

[48] Agarwal, S. and Farid, H., 2017, December. Photo forensics from JPEG dimples. In 2017 IEEE workshop on information forensics and security (WIFS) (pp. 1-6). IEEE.

[49] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[50] Harris, C.R., Millman, K.J., van der Walt, S.J. et al. Array programming with NumPy. Nature 585, 357–362 (2020). DOI: 10.1038/s41586-020-2649-2.

[51] Isola, P., Zhu, J.Y., Zhou, T. and Efros, A.A., 2017. Image-to-image translation with conditional adversarial networks. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 1125-1134).

[52] M. Cordts, M. Omran, S. Ramos, T. Rehfeld, M. Enzweiler, R. Benenson, U. Franke, S. Roth, and B. Schiele, “The Cityscapes Dataset for Semantic Urban Scene Understanding,” in Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2016.

[53] A. Yu and K. Grauman. "Fine-Grained Visual Comparisons with Local Learning". In CVPR, 2014.

[54] Zhu, Jun-Yan “Generative Visual Manipulation on the Natural Image Manifold”. In Proceedings of European Conference on Computer Vision (ECCV) 2016.

[55] Tyleček, R. and Šára, R., 2013. Spatial pattern templates for recognition of objects with regular structure. In Pattern Recognition: 35th German Conference, GCPR 2013, Saarbrücken, Germany, September 3-6, 2013. Proceedings 35 (pp. 364-374). Springer Berlin Heidelberg.

[56] Zhu, Jun-Yan and Park, Taesung and Isola, Phillip and Efros, Alexei A, 2017. Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks. Available at : https://github.com/junyanz/pytorch-CycleGAN-and-pix2pix/blob/master/docs/datasets.md

[57] Laffont, P.Y., Ren, Z., Tao, X., Qian, C. and Hays, J., 2014. Transient attributes for high-level understanding and editing of outdoor scenes. ACM Transactions on graphics (TOG), 33(4), pp.1-11.

[58] Moles, P. and Terry, N., 1997. The handbook of international financial terms. OUP Oxford.

[59] J. Long, E. Shelhamer, and T. Darrell. Fully convolutional networks for semantic segmentation. In CVPR, 2015

[60] Woody Bayliss, 2022. Automatic pruning of PIx2Pix on multiple datasets. Available at : https://github.com/Woodyet/Pix2Pix_Auto_Prune

[61] Xiao, X., Wang, Z. and Rajasekaran, S., 2019. Autoprune: Automatic network pruning by regularizing auxiliary parameters. Advances in neural information processing systems, 32.

[62] Mussay, B., Osadchy, M., Braverman, V., Zhou, S. and Feldman, D., 2019. Data-independent neural pruning via coresets. arXiv preprint arXiv:1907.04018.

[63] Hu, B., Zhao, T., Xie, Y., Wang, Y., Guo, X., Cheng, J. and Chen, Y., 2021, July. MIXP: Efficient Deep Neural Networks Pruning for Further FLOPs Compression via Neuron Bond. In 2021 International Joint Conference on Neural Networks (IJCNN) (pp. 1-8). IEEE.

[64] LeCun, Y., Bottou, L., Bengio, Y. and Haffner, P., 1998. Gradient-based learning applied to document recognition. Proceedings of the IEEE, 86(11), pp.2278-2324.

[65] Preprocessing techniqu Pal, K.K. and Sudeep, K.S., 2016, May. Preprocessing for image classification by convolutional neural networks. In 2016 IEEE International Conference on Recent Trends in Electronics, Information & Communication Technology (RTEICT) (pp. 1778-1781). IEEE.es

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

[67] He, K., Zhang, X., Ren, S. and Sun, J., 2016. Deep residual learning for image recognition. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 770-778).

[68] Deng, L., 2012. The mnist database of handwritten digit images for machine learning research. IEEE Signal Processing Magazine, 29(6), pp.141–142.

[69] Xiao, H., Rasul, K. and Vollgraf, R., 2017. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. arXiv preprint arXiv:1708.07747.

[70] Krizhevsky, A. and Hinton, G., 2009. Learning multiple layers of features from tiny images.

[71] Birds Kaggle, 2023. Available at: https://www.kaggle.com/datasets/gpiosenka/100-bird-species/code

[72] Deng, J. et al., 2009. Imagenet: A large-scale hierarchical image database. In 2009 IEEE conference on computer vision and pattern recognition. pp. 248–255.

[73] Vadera, S. and Ameen, S., 2022. Methods for pruning deep neural networks. IEEE Access, 10, pp.63280-63300.

[74] Han Xiao and Kashif Rasul and Roland Vollgraf 2017. Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms. Available at: https://github.com/zalandoresearch/fashion-mnist

[75] George A. Miller (1995). WordNet: A Lexical Database for English. Communications of the ACM Vol. 38, No. 11: 39-41.

[76] Dai, X., Yin, H. and Jha, N.K., 2019. NeST: A neural network synthesis tool based on a grow-and-prune paradigm. IEEE Transactions on Computers, 68(10), pp.1487-1497.

[77] LeCun, Y., Bottou, L., Bengio, Y. and Haffner, P., 1998. Gradient-based learning applied to document recognition. Proceedings of the IEEE, 86(11), pp.2278-2324.

[78] Gavrikov, Paul, 2020. VisualKeras. Available at: https://github.com/paulgavrikov/visualkeras

[79] Chollet, F. & others, 2015. Keras. Available at: https://github.com/fchollet/keras

[80] Build fast, sparse on-device models with the new TF mot pruning API (2021) The TensorFlow Blog. Available at: https://blog.tensorflow.org/2021/07/build-fast-sparse-on-device-models-with-tf-mot-pruning-api.html (Accessed: 20 June 2023).

[81] Xiao, X., Wang, Z. and Rajasekaran, S., 2019. Autoprune: Automatic network pruning by regularizing auxiliary parameters. Advances in neural information processing systems, 32.

[82] Mussay, B., Osadchy, M., Braverman, V., Zhou, S. and Feldman, D., 2019. Data-independent neural pruning via coresets. arXiv preprint arXiv:1907.04018.

[83] Hu, B., Zhao, T., Xie, Y., Wang, Y., Guo, X., Cheng, J. and Chen, Y., 2021, July. MIXP: Efficient Deep Neural Networks Pruning for Further FLOPs Compression via Neuron Bond. In 2021 International Joint Conference on Neural Networks (IJCNN) (pp. 1-8). IEEE.

[84] Dai, X., Yin, H. and Jha, N.K., 2019. NeST: A neural network synthesis tool based on a grow-and-prune paradigm. IEEE Transactions on Computers, 68(10), pp.1487-1497.

[85] Dai, X., Yin, H. and Jha, N.K., 2020. Incremental learning using a grow-and-prune paradigm with efficient neural networks. IEEE Transactions on Emerging Topics in Computing, 10(2), pp.752-762.

[86] Han, S., Pool, J., Tran, J. and Dally, W., 2015. Learning both weights and connections for efficient neural network. Advances in neural information processing systems, 28.

[87] Wu, T., Li, X., Zhou, D., Li, N. and Shi, J., 2021. Differential evolution based layer-wise weight pruning for compressing deep neural networks. Sensors, 21(3), p.880.

[88] Wu, T., Shi, J., Zhou, D., Lei, Y. and Gong, M., 2019, June. A multi-objective particle swarm optimization for neural networks pruning. In 2019 IEEE Congress on Evolutionary Computation (CEC) (pp. 570-577). IEEE.

[89] Babaeizadeh, M., Smaragdis, P. and Campbell, R.H., 2016. Noiseout: A simple way to prune neural networks. arXiv preprint arXiv:1611.06211.

[90] Sarvani, C.H., Ghorai, M., Dubey, S.R. and Basha, S.S., 2022. Hrel: Filter pruning based on high relevance between activation maps and class labels. Neural Networks, 147, pp.186-197.

[91] Ding, X., Zhou, X., Guo, Y., Han, J. and Liu, J., 2019. Global sparse momentum sgd for pruning very deep neural networks. Advances in Neural Information Processing Systems, 32.

[92] Zhao, C., Ni, B., Zhang, J., Zhao, Q., Zhang, W. and Tian, Q., 2019. Variational convolutional neural network pruning. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (pp. 2780-2789).

[93] Lin, S., Ji, R., Yan, C., Zhang, B., Cao, L., Ye, Q., Huang, F. and Doermann, D., 2019. Towards optimal structured cnn pruning via generative adversarial learning. In Proceedings of the IEEE/CVF conference on computer vision and pattern recognition (pp. 2790-2799).

[94] Chen, Y., Wen, X., Zhang, Y. and He, Q., 2022. FPC: Filter pruning via the contribution of output feature map for deep convolutional neural networks acceleration. Knowledge-Based Systems, 238, p.107876.

[95] Kang, M. and Han, B., 2020, November. Operation-aware soft channel pruning using differentiable masks. In International Conference on Machine Learning (pp. 5122-5131). PMLR.

[96] Lin, R., Ran, J., Wang, D., Chiu, K.H. and Wong, N., 2022. EZCrop: Energy-Zoned Channels for Robust Output Pruning. In Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision (pp. 19-28).

[97] Lin, L., Chen, S., Yang, Y. and Guo, Z., 2022, August. AACP: Model compression by accurate and automatic channel pruning. In 2022 26th International Conference on Pattern Recognition (ICPR) (pp. 2049-2055). IEEE.

[98] Lin, M., Ji, R., Li, S., Wang, Y., Wu, Y., Huang, F. and Ye, Q., 2021. Network pruning using adaptive exemplar filters. IEEE Transactions on Neural Networks and Learning Systems, 33(12), pp.7357-7366.

[99] Lin, M., Ji, R., Wang, Y., Zhang, Y., Zhang, B., Tian, Y. and Shao, L., 2020. Hrank: Filter pruning using high-rank feature map. In Proceedings of the IEEE/CVF conference on computer vision and pattern recognition (pp. 1529-1538).

[100] Basha, S. H., Farazuddin, Mohammad, Pulabaigari, Viswanath, Dubey, Shiv Ram, & Mukherjee, Snehasis (2021). Deep model compression based on the training history. arXiv preprint arXiv:2102.00160.

[101] Sarvani, C.H., Ghorai, M., Dubey, S.R. and Basha, S.S., 2022. Hrel: Filter pruning based on high relevance between activation maps and class labels. Neural Networks, 147, pp.186-197.

[102] Lin, M., Ji, R., Zhang, Y., Zhang, B., Wu, Y. and Tian, Y., 2020. Channel pruning via automatic structure search. arXiv preprint arXiv:2001.08565.

[103] Ganesh, M.R., Corso, J.J. and Sekeh, S.Y., 2021, January. Mint: Deep network compression via mutual information-based neuron trimming. In 2020 25th International Conference on Pattern Recognition (ICPR) (pp. 8251-8258). IEEE.

[104] Hewahi, N.M., 2019. Neural network pruning based on input importance. Journal of Intelligent & Fuzzy Systems, 37(2), pp.2243-2252.

[105] He, Y., Kang, G., Dong, X., Fu, Y. and Yang, Y., 2018. Soft filter pruning for accelerating deep convolutional neural networks. arXiv preprint arXiv:1808.06866.

[106] Shaopeng Guo, Yujie Wang, Quanquan Li, and Junjie Yan. Dmcp: Differentiable markov channel pruning for neural networks. Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pages 1539–1547, 2020.

[107] Liu, Z., Mu, H., Zhang, X., Guo, Z., Yang, X., Cheng, K.T. and Sun, J., 2019. Metapruning: Meta learning for automatic neural network channel pruning. In Proceedings of the IEEE/CVF international conference on computer vision (pp. 3296-3305).

[108] Kapishnikov, A., Bolukbasi, T., Viégas, F. and Terry, M., 2019. Xrai: Better attributions through regions. In Proceedings of the IEEE/CVF International Conference on Computer Vision (pp. 4948-4957).

[109] Grant, B.R. and Grant, P.R., 1996. High survival of Darwin's finch hybrids: effects of beak morphology and diets. Ecology, 77(2), pp.500-509.

[110] X. Dai, H. Yin and N. K. Jha, "Incremental Learning Using a Grow-and-Prune Paradigm With Efficient Neural Networks," in IEEE Transactions on Emerging Topics in Computing, vol. 10, no. 2, pp. 752-762, 1 April-June 2022, doi: 10.1109/TETC.2020.3037052.

[112] Build fast, sparse on-device models with the new TF mot pruning API (2021) The TensorFlow Blog. Available at: https://blog.tensorflow.org/2021/07/build-fast-sparse-on-device-models-with-tf-mot-pruning-api.html (Accessed: 20 June 2023).

[115] Zhu, L., Chen, Y., Yuille, A. and Freeman, W., 2010, June. Latent hierarchical structural learning for object detection. In 2010 IEEE computer society conference on computer vision and pattern recognition (pp. 1062-1069). IEEE.

[116] Alzubaidi, L., Zhang, J., Humaidi, A.J., Al-Dujaili, A., Duan, Y., Al-Shamma, O., Santamaría, J., Fadhel, M.A., Al-Amidie, M. and Farhan, L., 2021. Review of deep learning: Concepts, CNN architectures, challenges, applications, future directions. Journal of big Data, 8, pp.1-74.

[117] Holcombe, A.O., 2009. Seeing slow and seeing fast: two limits on perception. Trends in cognitive sciences, 13(5), pp.216-221.

[118] Webster, M.A., 1996. Human colour perception and its adaptation. Network: Computation in Neural Systems, 7(4), pp.587-634.

[119] Lisney, T.J., Rubene, D., Rózsa, J., Løvlie, H., Håstad, O. and Ödeen, A., 2011. Behavioural assessment of flicker fusion frequency in chicken Gallus gallus domesticus. Vision research, 51(12), pp.1324-1332.

[120] Siniscalchi, M., d'Ingeo, S., Fornelli, S. and Quaranta, A., 2017. Are dogs red–green colour blind?. Royal Society Open Science, 4(11), p.170869.

[121] Anwar, S., Khan, S. and Barnes, N., 2020. A deep journey into super-resolution: A survey. ACM Computing Surveys (CSUR), 53(3), pp.1-34.

[122] Parihar, A.S., Varshney, D., Pandya, K. and Aggarwal, A., 2022. A comprehensive survey on video frame interpolation techniques. The Visual Computer, pp.1-25.

[123] Creswell, A., White, T., Dumoulin, V., Arulkumaran, K., Sengupta, B. and Bharath, A.A., 2018. Generative adversarial networks: An overview. IEEE signal processing magazine, 35(1), pp.53-65.

[124] Carvalho, D.V., Pereira, E.M. and Cardoso, J.S., 2019. Machine learning interpretability: A survey on methods and metrics. Electronics, 8(8), p.832.

[125] DeRose, J.F., Wang, J. and Berger, M., 2020. Attention flows: Analyzing and comparing attention mechanisms in language models. IEEE Transactions on Visualization and Computer Graphics, 27(2), pp.1160-1170.

[127] N. Biggs, E. Lloyd, and R. Wilson, Graph Theory 1736-1936, Clarendon Press Oxford, 1976 (ISBN 0-19-853901-0)

[128] Bilal, A., Jourabloo, A., Ye, M., Liu, X. and Ren, L., 2017. Do convolutional neural networks learn class hierarchy?. IEEE transactions on visualization and computer graphics, 24(1), pp.152-162.

[129] Linardatos, P., Papastefanopoulos, V. and Kotsiantis, S., 2020. Explainable ai: A review of machine learning interpretability methods. Entropy, 23(1), p.18.

[130] Gu, J. and Dong, C., 2021. Interpreting super-resolution networks with local attribution maps. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (pp. 9199-9208).

[132] Fyles, M. (2023) Inside an AI ‘brain’ - what does machine learning look like?, Graphcore. Available at: https://www.graphcore.ai/posts/what-does-machine-learning-look-like (Accessed: 28 June 2023).

[134] High Efficiency Video Coding (HEVC) Family, H.265, MPEG-H Part 2 (Preliminary draft). Sustainability of Digital Formats. Washington, D.C.: Library of Congress. Available at: https://www.loc.gov/preservation/digital/formats/fdd/fdd000530.shtml

[135] D'Amario, V., Srivastava, S., Sasaki, T. and Boix, X., 2022. The Data Efficiency of Deep Learning Is Degraded by Unnecessary Input Dimensions. Frontiers in Computational Neuroscience, 16, p.1.

[136] R. Timofte, V. De Smet, and L. Van Gool. Anchored neighborhood regression for fast example-based super-resolution. In The IEEE International Conference on Computer Vision (ICCV), December 2013.

[137] Wang, W., Guo, R., Tian, Y. and Yang, W., 2019. Cfsnet: Toward a controllable feature space for image restoration. In Proceedings of the IEEE/CVF international conference on computer vision (pp. 4140-4149).

[139] Liu, B., Wang, M., Foroosh, H., Tappen, M. and Pensky, M., 2015. Sparse convolutional neural networks. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 806-814).

[140] Philipp, G., Song, D. and Carbonell, J.G., 2017. The exploding gradient problem demystified-definition, prevalence, impact, origin, tradeoffs, and solutions. arXiv preprint arXiv:1712.05577.

[141] Masters, D. and Luschi, C., 2018. Revisiting small batch training for deep neural networks. arXiv preprint arXiv:1804.07612.

[142] Kingma, D.P. and Ba, J., 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

[144] Ronneberger, O., Fischer, P. and Brox, T., 2015. U-net: Convolutional networks for biomedical image segmentation. In Medical Image Computing and Computer-Assisted Intervention–MICCAI 2015: 18th International Conference, Munich, Germany, October 5-9, 2015, Proceedings, Part III 18 (pp. 234-241). Springer International Publishing.

[145] Mirza, M. and Osindero, S., 2014. Conditional generative adversarial nets. *arXiv preprint arXiv:1411.1784*.

[147] Python multiprocessing pickling error:. Available at: https://stackoverflow.com/questions/8804830/python-multiprocessing-picklingerror-cant-pickle-type-function (Accessed: 28 June 2023).

[148] Yang, C., Shen, Y., Xu, Y., Zhao, D., Dai, B. and Zhou, B., 2022. Improving gans with a dynamic discriminator. arXiv preprint arXiv:2209.09897.

[149] Salimans, T., Goodfellow, I., Zaremba, W., Cheung, V., Radford, A. and Chen, X., 2016. Improved techniques for training gans. Advances in neural information processing systems, 29.

[150] Srivastava, A., Valkov, L., Russell, C., Gutmann, M.U. and Sutton, C., 2017. Veegan: Reducing mode collapse in gans using implicit variational learning. Advances in neural information processing systems, 30.

[151] Barratt, S. and Sharma, R., 2018. A note on the inception score. arXiv preprint arXiv:1801.01973.

[152] Liu, Y., Shu, Z., Li, Y., Lin, Z., Perazzi, F. and Kung, S.Y., 2021. Content-aware gan compression. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (pp. 12156-12166).

[154] Chavolla, E., Zaldivar, D., Cuevas, E. and Perez, M.A., 2018. Color spaces advantages and disadvantages in image color clustering segmentation. Advances in soft computing and machine learning in image processing, pp.3-22.

[155] Hochreiter, S., 1998. The vanishing gradient problem during learning recurrent neural nets and problem solutions. International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems, 6(02), pp.107-116.

[156] Mission, values and public purposes - about the BBC (no date) BBC News. Available at: https://www.bbc.com/aboutthebbc/governance/mission (Accessed: 28 June 2023).

[157] Samuel Danzon-Chambaud, December 2021. Automated news at the BBC. BBC Research & Development White Paper, BRITISH BROADCASTING CORPORATION. Available at: https://www.bbc.co.uk/rd/publications/automated-news-at-bbc-algorithmic-journalism

[158] Setiono, R., Baesens, B. and Mues, C., 2009. A note on knowledge discovery using neural networks and its application to credit card screening. European Journal of Operational Research, 192(1), pp.326-332.

[159] Chang, J., Lu, Y., Xue, P., Xu, Y. and Wei, Z., 2022. Global balanced iterative pruning for efficient convolutional neural networks. Neural Computing and Applications, 34(23), pp.21119-21138.

[160] Hinton, G.E. and Roweis, S., 2002. Stochastic neighbor embedding. Advances in neural information processing systems, 15.

[161] Van der Maaten, L. and Hinton, G., 2008. Visualizing data using t-SNE. Journal of machine learning research, 9(11).

[162] Kullback, S. and Leibler, R.A., 1951. On information and sufficiency. *The annals of mathematical statistics*, *22*(1), pp.79-86.

[164] Podell, D., English, Z., Lacey, K., Blattmann, A., Dockhorn, T., Müller, J., Penna, J. and Rombach, R., 2023. Sdxl: Improving latent diffusion models for high-resolution image synthesis. *arXiv preprint arXiv:2307.01952*.

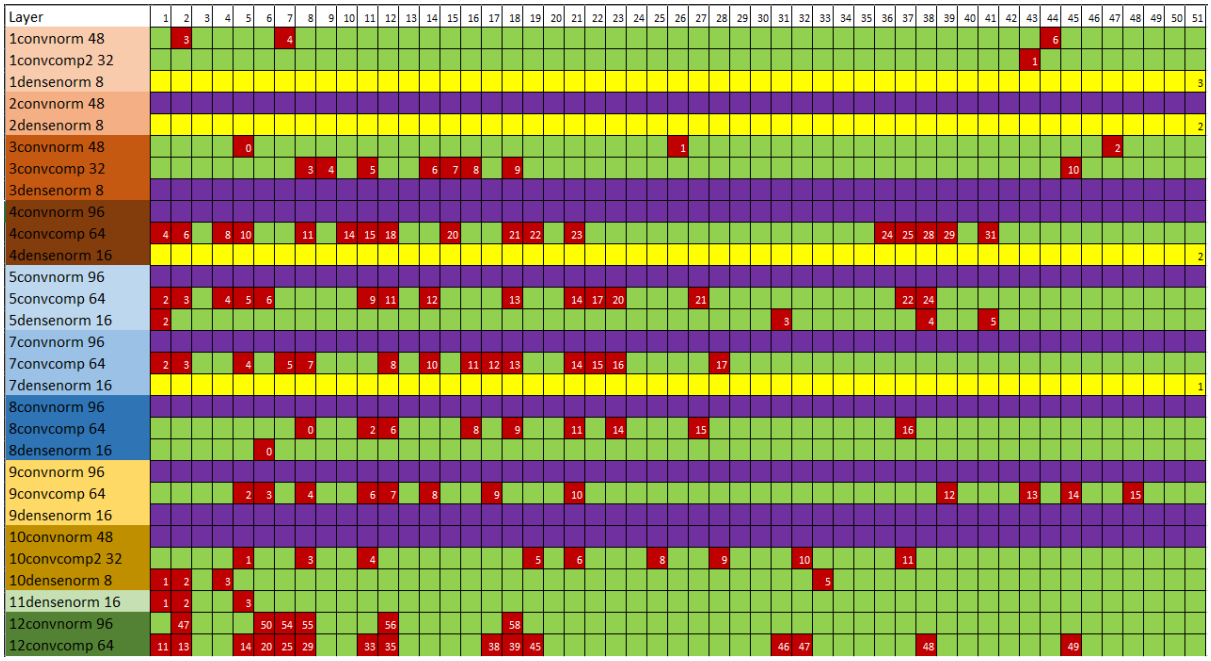
[165] Achiam, J., Adler, S., Agarwal, S., Ahmad, L., Akkaya, I., Aleman, F.L., Almeida, D., Altenschmidt, J., Altman, S., Anadkat, S. and Avila, R., 2023. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*.

[167] Dar, Y., Muthukumar, V. and Baraniuk, R.G., 2021. A farewell to the bias-variance tradeoff? an overview of the theory of overparameterized machine learning. *arXiv preprint arXiv:2109.02355*.

[168] Liebenwein, L., Baykal, C., Carter, B., Gifford, D. and Rus, D., 2021. Lost in pruning: The effects of pruning neural networks beyond test accuracy. *Proceedings of Machine Learning and Systems*, *3*, pp.93-138.

11 APPENDIX

11.1 A



Evolution of the layer pruning over time, red blocks indicate a channel was pruned, the number in the box shows how many filters have been removed at that point.

On the left is the name of the layer and its type, additionally it has the number of nodes associated with the layer, these are normally of the magnitude 8/16/32/48/64/96. The red boxes show where nodes/channels have been removed from the network, and the white number inside shows the number of layers removed. Purple lines indicated layers that did not have any channels or nodes removed, even though it was possible for the algorithm to remove them. Yellow lines indicate layers where layers were removed on the first iteration of the loop but were then left unpruned for the rest of the pruning loop.

11.2 B

Below is a list of animations relevant to the thesis.

1 - https://youtu.be/csCM9fInltk - Birds VGG16 pruned model over time

2 - https://youtu.be/Lw-WSmSzTJc - MLI first visualisations

3 - https://youtu.be/X7QL8ZWhIbo - Pix2Pix Pruning Over time

4 - https://youtu.be/4BSp1deTwgs - Vanilla VTM Vs NN aided VTM

5- https://tinyurl.com/2sk2y9j9 - Pruning algorithm explained visually

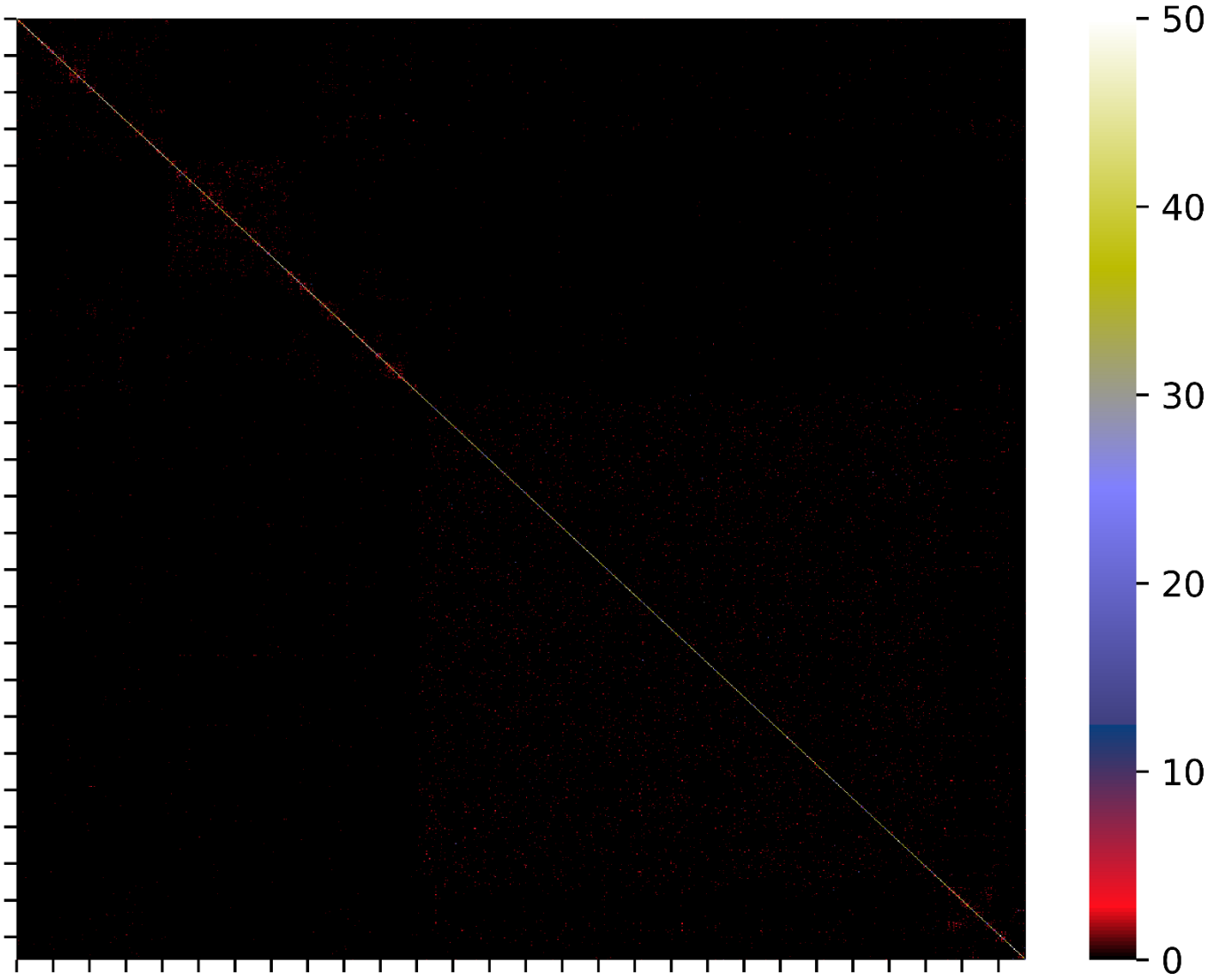
11.3 C

Repository of all code used for the creation of the thesis.

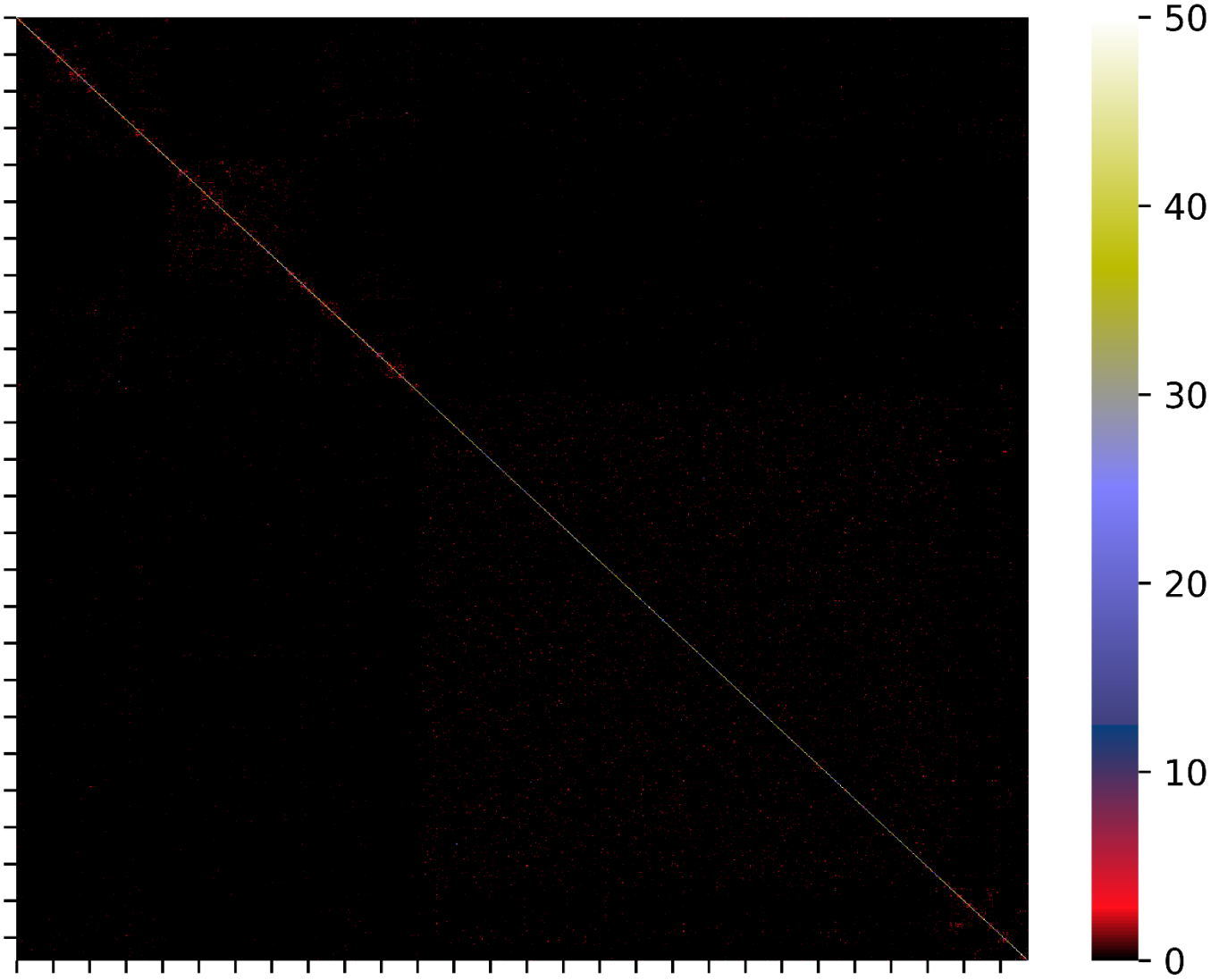
https://github.com/Woodyet/Thesis_Code_Dump

To request access please email woody.bayliss@sky.com

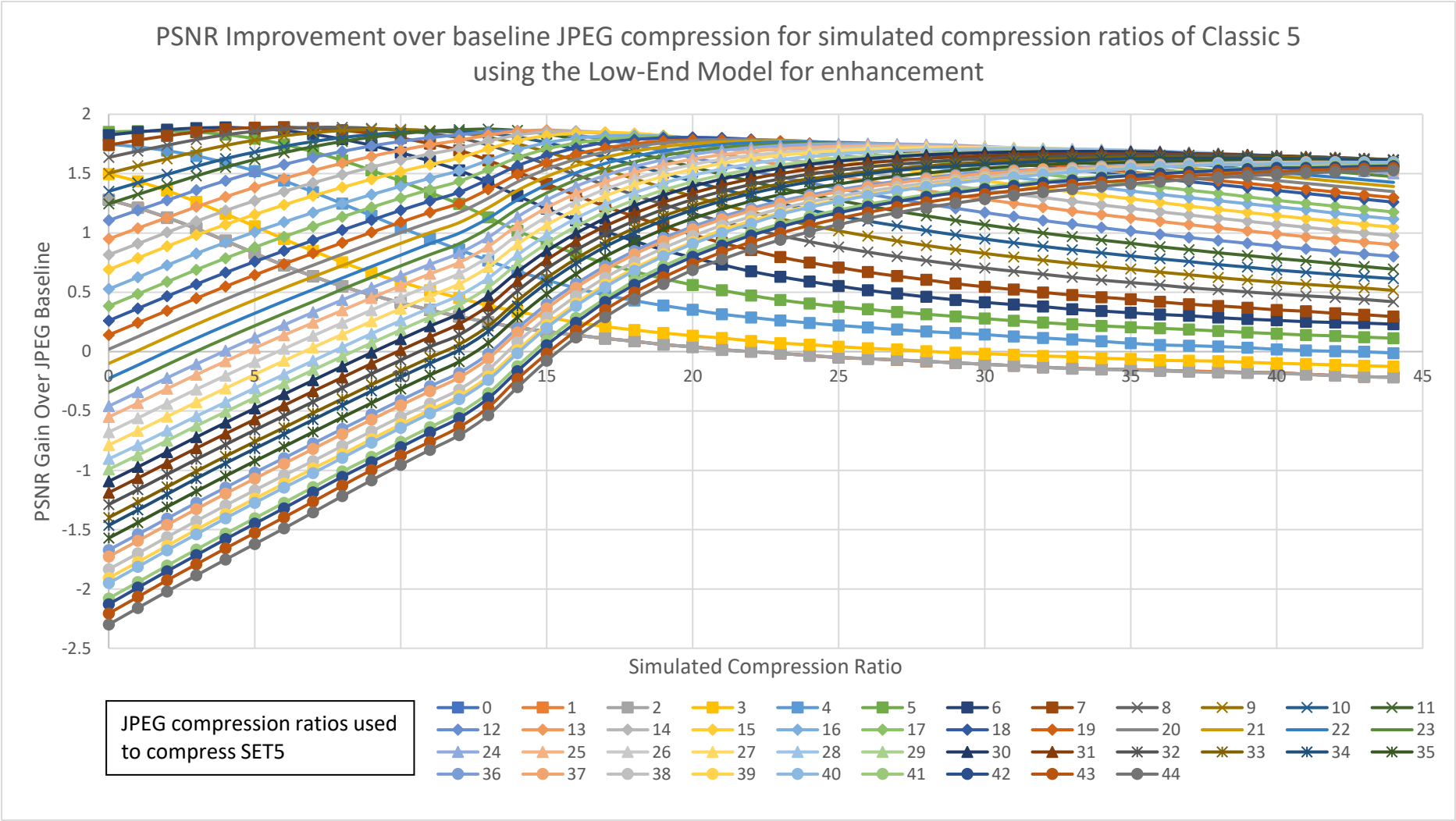
11.4 D – Unpruned Confusion Matrix



11.5 E – Pruned confusion Matrix



11.6 G. JPEG SET5 ENCODED AT ALL COMPRESSION RATIOS



PSNR of processed images at all compression ratios.

11.7 Full list of figures

11.8 Full list of Tables