university of groningen / faculty of science and engineering / artificial intelligence

The Importance of Filters: Using Shapley Value Pruning to Optimize Convolutional Neural Networks

Bachelor's Project Thesis

Romanas Munovas, s4004981 (r.munovas@student.rug.nl)
Supervisors: dr. C.P. Lawrence

Abstract: This paper presents an analysis of the importance of filters in Convolutional Neural Networks (CNNs) and the use of Shapley value pruning to optimize these architectures. CNNs have become the industry standard for computer vision tasks, but their growing depth and parameter size are demanding more resources for training and inference. To optimize these architectures, the practice of pruning is often used to remove redundant filters or layers. The current state-of-the-art criterion for pruning in a low-data regime approximates Shapley values via Monte Carlo sampling (Ancona et al., 2020). Computing the actual Shapley values would be optimal, however, calculating the Shapley value for a single image has a high computational complexity, which limits the application of this method. To solve this problem, this paper proposes a hybrid approach that uses a variation of Monte Carlo approximation and actual Shapley value calculation when the number of activations allows it. This approach is designed to tackle the issue of the dangers of pruning a wanted unit due to the inevitable variance created by Monte Carlo approximation in a low-sample setting. The results show that this hybrid approach significantly outperforms random pruning and slightly outperforms the exclusive use of Monte Carlo approximation. This paper also investigates the mechanisms behind the decision making of the Shapley criterion in order to gain more insight into how the scores for activations are attributed.

1 Introduction

Convolutional Neural Networks (CNNs) have become the industry standard for computer vision tasks involving object detection, recognition, and classification, and so the demand for state-of-the-art CNNs to be deployed on mobile platforms has never been higher. At the same time, CNNs are also becoming deeper, thus growing in parameter size and demanding significantly more resources for training and inference (Gu et al., 2015). It is often the case, however, that these architectures have parameters that do not contribute to the performance of the inference. In order to optimize the architecture, the redundant parameters are often removed in a process known as pruning.

Network pruning has been around for awhile. In LeCun, Denker, & Solla (1990), unstructured pruning was introduced with the goal to reduce the size of an architecture. Han et al. (2015) utilizes a salience heuristic to nullify single weights without affecting the architecture's structure; the process is illustrated in Figure 1.1. While this does reduce the number of parameters and size of the CNN, it does not actually decrease the computational cost because the network will still have to do the same amount of floating-point operations (FLOPs) during forward and backward passes as the weights are not removed per se. Instead, we will focus on structured pruning (further on, just pruning) — which, removes whole filters (as seen in Figure 1.2) and the in-out connections associated with them instead (Mittal et al., 2018) — as a means to also reduce the inference time.

Pruning of an architecture is done in a top-down manner, — starting with a complete model, — by ranking the chosen units and removing, or pruning, the ones with the smallest importance

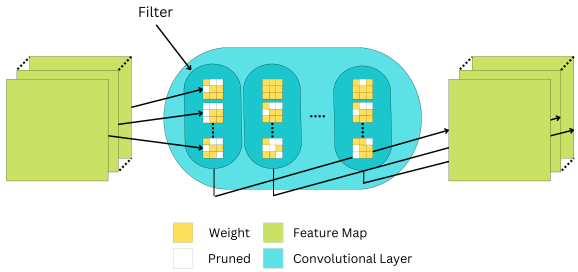

Figure 1.1: Graphic illustrating unstructured pruning. In this example, weights are being pruned but the number of the feature maps stays the same.

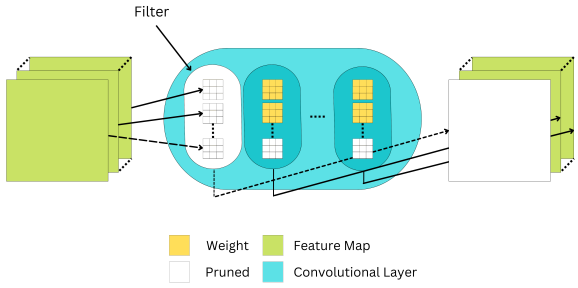

Figure 1.2: Graphic illustrating structured pruning. In this example, a filter is being pruned which results in a smaller number of feature maps in total.

score (Li et al., 2016). The pruned architecture is more compact, and if done successfully does not show a significant drop in performance (Hu et al., 2016).

The standard procedure for network pruning is:

1. Train a network
2. Prune n units with the lowest importance score
3. Fine-tune the network
4. Repeat steps 2-3 until a significant drop in accuracy

While there is not a unified criterion standard that has been accepted as the correct way of calculating the importance score, attempts have been made to introduce one. Molchanov et al. (2016) criterion involved approximating the changes of the cost function using first-order Taylor Expansion. Li et al. (2016) used L1-norm criterion to reduce the network size by 38% while retaining an accuracy similar to the original model. In Hu et al. (2016), they used percentage of zeros in a filter as an importance metric, allowing them to compress VGG-16 and LeNet-5 by 2-3 times without losing the accuracy. Surprisingly, Mittal et al. (2018) showed that random pruning is on par with other pruning strategies, managing to prune up to 50% of the network without losing accuracy given enough fine-tuning resources. This would mean that the importance score is a redundant metric. However, Ancona et al. (2020) showed that while random pruning may be effective in some cases, Shapley value criterion is the preferred importance score metric due to also working in low-data settings, where fine-tuning is either ineffective or not possible.

Most of the present pruning literature focuses on finding a criterion or method that is the most effective in reducing the size without significantly penalizing the accuracy of a network. But the reasoning behind the decisions to prune a particular filter over another remains widely unexplored. Understanding the mechanics behind the decision making of a criterion could help design better networks. One way to do so is by analyzing the feature maps, that is, intermediate representations of images after being passed through a filter. This allows us to inspect

what kind of features are being passed in-between layers and into the final inference.

1.1 Shapley values

Shapley values (Shapley, 1952) were introduced as a solution concept to cooperative games. In a cooperative game, coalitions of players compete and cooperate toward achieving a unified goal. The outcome of each game depends only on the subset of players that are found within that coalition.

Shapley (1952) provides a way to calculate how much each player contributes to the game. Assume $P = 1, 2, ..., n$ is a finite set of n players i called the grand coalition; if $v_x : S \subset P \to \mathbf{R}$ maps out the outcome of the game in which a subset of players $S \subset P$ participate, where $x_1, ..., x_k \in X$ is the variation of the game that is being played, the marginal contribution of a single player can be found by the difference of the game's outcome with and without the player:

$$m_x(S, i) := v_x(S) - v_x(S \setminus \{i\}) \tag{1.1}$$

The advantage of this solution is that it takes into account the interaction between players, as opposed to naively computing the contribution of a single player $v_x(i)$. However, the equation only calculates the player's contribution for a given coalition S. In order to calculate the Shapley value for player i and game x, we have to average the marginal contributions over all subsets that contain i:

$$\phi_x(i) := \frac{1}{n!} \sum_{S \subseteq P \setminus \{i\}} (|S|!(n - |S| - 1)!) \cdot m_x(S, i) \tag{1.2}$$

For each player, there are infinitely many solutions. However, some of them are considered to be more 'fair' than other ones. For Shapley value to be a unique solution it must satisfy the following properties of 'fairness' (that is, the worth of the goal $v_x(P)$ is split in a way that all players in P are considered upon calculation) as adapted from Strumbelj & Kononenko (2010):

1. **The Efficiency Axiom**: the sum of Shapley values is equal to the difference between the valuation of the grand coalition and an empty coalition

$$\sum_{i \in P} \phi_x(i) = v_x(P) - v_x(\emptyset) \tag{1.3}$$

2. **The Symmetry Axiom**: If for two players i and j $v_x(S \cup i) = v_x(S \cup j)$ holds for every S, where $S \subset P$ and $i, j \notin S$, then $\phi_x(i) = \phi_x(j)$.

3. **The Null Axiom**: If $v_x(S \cup i) = v_x(S)$ holds for every S, where $S \subset P$ and $i \not\subset S$, then $\phi_x(i) = 0$

4. **The Additivity Axiom**: For any two game-evaluation functions v_x and w_x: ${}_v\phi_x(S) + {}_w\phi_x(S) = {}_{(v+w)}\phi_x(S)$ for all S and any game x, where $v_x(S) + w_x(S) = (v_x + w_x)(S)$. Notation ${}_v\phi_x(S)$ denotes that the game-evaluating function v was used in the computation of the Shapley values ϕ_x for all S.

1.1.1 Parallels with Neural Networks

In the context of our problem, let us assume we have a simple feedforward CNN f with L layers:

$$f(x) = (f^{(1)} \circ f^{(2)} \circ ... \circ f^{(L)})(x) \tag{1.4}$$

where x is an input example, such as an image, fed through each layer for classification, and f^l is a transformation function used on the activation z^{l-1} of the previous layer:

$$z^{(l)} = f^{(l)} z^{(l-1)}; z^0 = x \tag{1.5}$$

If we assume that z_i^l is a filter on a particular layer l, the goal of the pruning task can be reduced to finding a set Z of filters to prune which will reach a goal of choice, such as maximizing the performance metrics.

It is important to note that the output of the model does not depend on the activations of previous layers, that is, the loss only depends on the units in z^l.

In such case, we can write the network loss function as a function of z^l (Ancona et al., 2020):

$$\overline{L}(z^l; y) = L((f^{(l+1)} \cdot ... \cdot f^{(L)})(z^{(l)}); y) = L(x; y) \tag{1.6}$$

If we consider the filters or layers of a neural network in a classification task as players working

together towards achieving a unified goal, such as maximizing accuracy or minimizing the loss, the whole process can be viewed as a cooperative game. In such cases, Shapley values also have an advantage over other methods as they provide a game-theoretic foundation and explanation behind the contribution of each player.

We can assume that every prunable filter $z_1^l, ..., z_n^l \in P$ is a player in a game of coalitions, where the loss function $\overline{L}_x : S \subset P- > R$ is a way to assign the contribution of each filter. In this case, $x_1, ..., x_k \in X$ would denote the input image, where X is the set of all images to be fed into the network. We can then rewrite (1.1) and (1.2) in the following way:

$$m_x(S, z^l) := \overline{L}_x(S) - \overline{L}_x(S \setminus \{z^l\})$$
$$\phi_x(z^l) := \frac{1}{n!} \sum_{S \subseteq P \setminus \{z^l\}} (|S|!(n - |S| - 1)!) \cdot m_x(S, z^l) \tag{1.7}$$

Note that viewing cooperative games through the lens of a neural network does satisfy all four Shapley value axioms and assumptions of 'fairness' of the solution ϕ (as adapted from Ancona et al. (2020)):

1. **The Efficiency Axiom**: The sum of attributions equals to the difference between the loss obtained when all activations are present and the loss obtained when all activations are removed. Similarly, in Shapley, the sum of the Shapley values for all the players in the grand coalition P will be equal to the difference between valuation function when all of the players are present and when none are.

2. **The Symmetry Axiom**: If swapping the values of two activations doesn't affect the loss, then those two activations are attributed equally; in Shapley, when the outcome of the game-evaluating function $v_x(S)$ when player i participates (i.e. $v_x(S \cup i)$) equals to the value of the function when player j participates instead, then the Shapley Values of those players will be the same, given $i, j \notin S$.

3. **The Null Axiom**: If a particular activation z^l has no impact on the loss, then its attribution is considered to be zero, similarly to how in Shapley, if the game-evaluating function v_x does not depend on a particular player i, then the Shapley value $\phi_x(i)$ for that player is 0.

4. **The Additivity Axiom**: Suppose the loss function f can be expressed as a linear combination of two sub-networks' loss functions (i.e., $f = a \times f_1 + b \times f_2$). In that case, any attribution computed on the overall network should also be a linear combination of the attributions computed on the sub-networks, with the same weights (a and b). From the perspective of Shapley, calculating the Shapley value $\phi_x(S)$ for any S using the combination of two game-evaluating functions v_x and w_x is the same as the sum of two Shapley value computations, one which uses v_x and another which uses w_x.

In the context of a neural network, using a single image x for calculating the loss of a forward-propagation algorithm moving from one layer to another will have high variance and, — if used in the context of calculating Shapley values — will likely not give a good representation of the contribution of each filter i in regards to the goal. For this reason, a set of k images $x \in X$ — independent of the images used for training, validating, or testing the network — should be used. The Shapley values for each image x are then aggregated and averaged for each filter i to gain the true contribution of each filter for the image set(1.8):

$$\texttt{Sh}(X, i) = \frac{1}{k} \sum_{x \in X} \phi_x(i) \tag{1.8}$$

1.1.2 The Problem with Shapley values

Intuitively, computing the Shapley value ϕ_x for filter z^l in a single layer l for a single image has the computational complexity of $O(2^{|z^l|})$ and is NP-complete (Deng & Papadimitriou, 1994). The computation for a single image requires evaluating the network for every single filter permutation and therefore is often limited by the computational power of the hardware at hand. Various methods of approximating Shapley values have been proposed (Strumbelj & Kononenko, 2010; Castro et al., 2009), but the most practical and easy to implement is that of Castro et al. (2009) which uses the Monte Carlo method to sample the permutations. This reduces the computational complexity

to just $O(C * |z^l|)$, were C is the number of samples taken. However, while Monte Carlo does have a convergence property, in cases where the number of samples is limited (again, due to computational power constraints), the approximation may suffer from high variance and introduce a strong bias (James, 1980) which may hinder the performance of the pruning algorithm.

When pruning filters from a single layer, the significance of the effect (as seen later in the result section) of each pruned filter on a chosen metric, — for example, accuracy — differs in magnitude. Intuitively, the trend is logarithmic; each consecutive filter pruned will have a bigger effect on the accuracy (i.e. it will degrade faster) because there are less filters to replace its role in classification. Hence, at some point during the pruning procedure, the choice of which filter to prune can become the difference between having a network that performs almost as well as its unpruned counterpart and a network that is not able to predict the classes correctly at all.

In Ancona et al. (2020), they use Monte Carlo approximation with a constant number of samples across all pruning iterations for DNNs (Deep Neural Networks). As discussed previously, the problem with using a Monte Carlo approximation of Shapley values is that if the number of samples is not high enough, the results will be exposed to high variance. Since the main reason for using Monte Carlo is that it does not require as many computational resources as calculating the actual Shapley value, the problem can be at least partially solved by introducing an amelioration hyper-parameter t (1.9):

$$C_{q+1} := C_q^t, \text{ where } t > 1 \tag{1.9}$$

where C_q is the number of samples taken at iteration q, which would increase the number of samples taken whenever a filter is pruned. A larger number of filters (and hence more voluminous permutations) requires a bigger number of forward passes for the calculation of the marginal contribution of each filter; hence, there exists a value t which increases the number of samples taken — reducing the variance in the approximation of the Shapley values, — without exceeding the computational power limits, assuming the C chosen initially encompasses the limitations.

Even then, while the variance may be reduced, it will most likely exist even when the number of samples is increased. To tackle the issue of the dangers of pruning a wanted filter due to inevitable variance created by Monte Carlo approximation in a low-sample setting, I propose a hybrid approach to calculating the Shapley values. Instead of reserving to just one method of Shapley value calculation, as was done in Ancona et al. (2020) (i.e. exclusively using Monte Carlo approximation in situations where calculating the actual Shapley value is unfeasible), we can use a mix of both Monte Carlo approximation and the actual Shapley value calculation when the number of activations allows it. Since each consecutive filter pruned has a more significant impact on the standard scoring metrics (accuracy and loss), approximating the Shapley values instead of computing the exact values is unreasonable.

1.2 Research Question

The goal of this work is to investigate whether the Hybrid Shapley value pruning pipeline performs better than the exclusive approximation of the Shapley values, and to examine the mechanisms behind the decision making mechanism of the Shapley pruning algorithm by analyzing the feature maps of the underlying architecture.

2 Methodology

2.1 Data and Architecture

The performance of the Hybrid Shapley value pruning algorithm in a low-data regime was evaluated and analyzed in the following experiments. Low-data is characterized by a setting where fine-tuning of the network is either not effective or not possible. Consequently, in-between iterations of the experiments we prune the filter with the lowest Shapley value, but the network is not fine-tuned. The Hybrid Shapley value pruning algorithm is compared to the Shapley value Monte Carlo Sampling, with the random pruning algorithm serving as the baseline. The feature maps produced by the filters are also later analyzed with their Shapley scores in mind.

2.1.1 MNIST

The MNIST dataset of 28x28 images of handwritten digits consisting of 70,000 examples was used. The MNIST dataset consists of ten classes for each digit from 0 to 9. Out of the given datset, 60,000 were used for training, 8,000 for testing, and 1,000 each for validation and Shapley. The MNIST dataset was chosen for the ease of visual analysis and minimizing the computational power that calculations of the Shapley values require, as it is small in dimensions and hence memory size, and requires no preprocessing for the chosen architecture.

2.1.2 LeNet-1

An architecture essentially identical to LeNet-1 (as described in LeCun, Boser, et al. (1990)) was used in the experiments. The experiments were implemented in PyTorch framework. LeNet-1 was chosen for this task as the simplest (to our knowledge) CNN capable of solving the MNIST dataset with a great performance. The architecture has five hidden layers, out of which two are convolutional, two averaging/subsampling, and a single output layer (Figure 5.1) It is important to note that only the convolutional layers are prunable due to being mutable. A detailed discussion of the architecture can be found in LeCun, Boser, et al. (1990).

It is generally not recommended to prune the filters in the first layer or the first layer as a whole, as it possesses the most important information (Molchanov et al., 2016) that is passed onto the next layers, — which is unrecoverable if pruned, — and can result in a significant drop in performance within a single iteration as it is divided across only four filters. On the other hand, most of the parameters are spread out between the connections of the second convolutional layer and the first fully connected layer. Consequently, we only prune the filters found in the second convolutional layer.

2.2 Procedure

For each pruning strategy we start with the fully trained LeNet-1 architecture, and prune only the 12 filters found in the second layer.

2.2.1 Monte Carlo pruning

For the Monte Carlo pruning algorithm, we used $C = 100$ samples throughout the experiment. Each iteration is repeated until there is a single filter left.

At the start of each iteration, the original loss is computed by doing a forward-pass on a set of MNIST images (separate from the training, validation, and testing sets) when all filters are present. For each sample n we randomly permute the set of all filters Z. Each filter $z \in Z$ is sequentially masked and the loss for the updated model is computed through a forward-pass. The intermediate Shapley value is then found by computing the difference between the loss when the filter was present and when it was pruned, over the number of samples; this process is repeated until the intermediate Shapley value is calculated for each filter. After all sampling episodes, the intermediate Shapley values are aggregated over the samples to find the approximate Shapley value for each filter; the one with the lowest Shapley value is pruned, but the model is not fine-tuned.

2.2.2 Hybrid Shapley pruning

For Hybrid Shapley, we used the Monte Carlo Sampling with $C = 100$ samples and $t = 1.08$ amelioration parameter to approximate the Shapley values until there were 6 filters left (process described in section 2.2.1, except after a filter has been pruned, the number of samples at the next iteration increases as shown in Formula 1.9), at which point the algorithm would start computing the actual Shapley values. The only difference between computing Shapley values through Monte Carlo sampling and computing actual Shapley values is that instead of sampling random permutations, the latter instead uses all possible permutations.

The amelioration parameter was chosen to be high enough to create a significant increase in the number of samples (at 5 filters, the number of samples taken is $C = 859$, almost 9 times larger) without sacrificing the computational time by too much. We used accuracy and cross-entropy loss as the criteria for measuring performance of the algorithms. In addition to that, we used Area Under the Curve (AUC) of the loss and accuracy as a quantitative measure of the results; a lower value for loss and a higher value for accuracy generally indicated a

AUC	Random	Monte Carlo Shapley values	Hybrid Shapley values
Accuracy	0.65±0.03	0.78±0.00	0.79
Loss	0.83±0.02	0.39±0.00	0.37

Table 3.1: Area under the curve for accuracy and loss. A lower value for loss and a higher value for accuracy indicates a slower decay in performance. Standard deviation over 3 trials is reported where applicable.

slower degradation of the performance.

2.2.3 Feature Maps

For the feature maps, we passed the images through all the layers up to the last convolutional before visualizing them for each filter. The feature maps were normalized between 0 and 1 for better visual inference. One of each digit from the subset of images used for Shapley experiments was plotted.

2.3 Code Repository

The entire code repository along with the presentation in this paper are available on Github at https://github.com/Zarathustrai/Shapley-Pruning. The source directory contains two Jupyter notebooks; 'shapley-value-pruning.ipynb' can be used to reproduce the results, while 'graphing.ipynb' can be used to partially reproduce the illustrations used in this paper.

3 Results and Discussion

3.1 Algorithms

The experiment was repeated three times; the average results are seen in Table 3.1. Hybrid Shapley produces a significantly slower performance degradation compared to random pruning. In comparison to Monte Carlo sampling, Hybrid Shapley also performs better, but only by a minuscule amount. This can also be seen in Figures 3.1 and 3.2, where the loss and accuracy curves for Monte and Hybrid Shapley are almost identically the same.

For both of the Shapley value methods we can see that until 10 filters have been pruned the decrease in the performance seems to be consistent, and in some cases even improves the performance. In comparison to the random pruning algorithm, the decrease in the performance is significantly lower

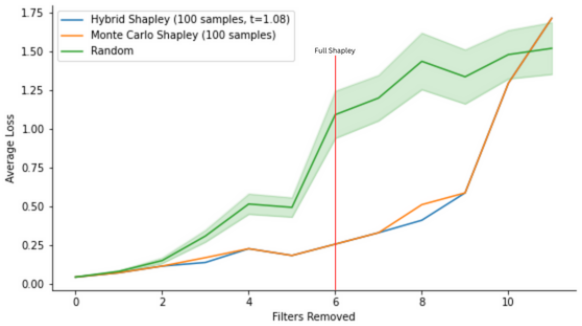

Figure 3.1: Average loss comparison between different Shapley value algorithms. The red line indicates at which point the Hybrid Shapley algorithm started computing the full Shapley value. Note that the standard deviation across 3 trials has been reported where applicable.

for the Shapley algorithms. It is also important to note that the Hybrid Shapley algorithm chooses to prune filters differently from Monte Carlo Shapley at two points in time as can be seen by the small deviations between the two graphs, namely at 3 and 8 filters removed. At 3 filters removed, the Hybrid Shapley still samples the values, meaning that the higher number of samples (due to the amelioration parameter) did make a difference when approximating, hence the difference. And again, at 8 filters, Hybrid Shapley outperformed Monte Carlo Shapley, meaning that the classic sampling algorithm failed to approximate the Shapley values correctly. However, the significant drop in accuracy only occurs when 10 filters have been removed, so the algorithm choice potentially has no difference depending on at which point the pruning is stopped.

3.2 Feature Maps

The plotted feature maps of each digit for every convolutional layer can be seen in Figure 5.2. While our first intention was to inspect the feature maps produced by the second convolutional layer, we soon realized that the features are far too abstract for analysis and discussion. This is because deeper layers tend to focus on more complex features such as edges, while shallow layers focus on the shape of the objects as a whole. Instead, we will focus on the feature maps produced by the first convolutional layer, specifically on the filters with the

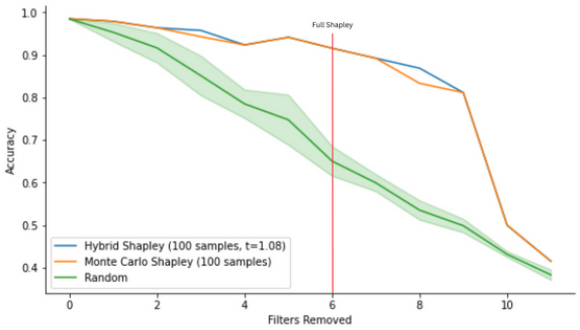

Figure 3.2: Average accuracy comparison between different Shapley value algorithms. The red line indicates at which point the Hybrid Shapley algorithm started computing the full Shapley value. Note that the standard deviation across 3 trials has been reported where applicable.

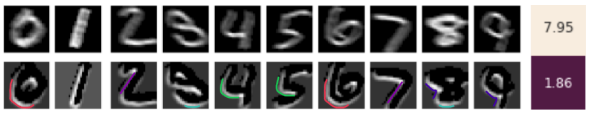

Figure 3.3: The comparison of feature maps between the filter with the highest and lowest Shapley value of the first convolutional layer. Edges highlighted with the same color are identical to one another. Best seen in digital format.

highest and lowest Shapley values. Note that the feature maps from the first convolutional layer are passed onto the second one, and hence are of equal value in the discussion of their mechanisms.

In Figure 3.3, the first thing to notice is that the digits in the feature maps produced by the filter with the highest Shapley value are 'complete', that is, the digits are presented as if drawn with a single stroke. On the other hand, the feature maps with the lowest Shapley values are only recognizable due to the phenomenon known as Gestalt's closure principle (Wertheimer, 1938) which states that humans tend to fill out gaps to perceive objects as being whole, even when fragments are missing. Even then, if the white color were to be isolated it would be hard to recognize the digits as they are fragmented. For the architecture, it might be even more difficult, as the feature maps are completely different compared to the input images and may not capture all the unique features that distinguish the digits from one another. And while these are not the feature maps that are passed on to the final inference, they are passed onto the next convolutional layer which infers more complex features from them. Hence, the low score could partially be explained by the idea that features which do not contribute or contribute negatively to the inference are passed on to the next convolutional layer.

In addition to that, some of the feature maps for the low contribution score filter share some identical features, which can be seen in Figure 3.3, highlighted by different colors. While the same identical features are also found in the feature maps of the filter with the highest Shapley score, since the feature maps are a somewhat accurate and whole representation of the digits, the features extracted by the second convolutional layer have a high probability to be useful in the final inference stage. Since some of the highlighted edges in the low contribution filter feature maps make up a significant part of the whole feature map, the feature maps after being passed through the second convolutional layer could share a significant amount of similarities. These similarities could make it difficult for the architecture to distinguish the differences between different digits during the final inference, which could explain the low Shapley value.

3.3 Limitations

Shallow CNNs such as LeNet-1 are no longer used for image classification as more modern architectures are available which can achieve better performance on such tasks. These state-of-the-art architectures are also deeper and have many more filters. In such scenarios, reaching the point of being able to compute full Shapley values during the pruning process instead of approximating them using Monte Carlo is most likely unfeasible; this is because the percentage of filters to be pruned before reaching the breaking point of computing actual Shapley values increases in a logarithmic manner.

In addition to that, the discussion of the feature maps is purely speculative; since there is no standardized way to report the results, we reserve to qualitative analysis. We attempt to interpret the decision making of the architecture using human reasoning and pattern recognition. Consequently, the conclusions derived from the discussion of the

mechanisms behind the decision making of the Shapley algorithm should be taken with a grain of salt.

3.4 Future directions

While computing the actual Shapley values of filters may not be feasible for most architectures due to the presence of a high number of filters, it could be possible in a layer-wise pruning scenario, as CNNs always have significantly less layers than filters. Modern architectures such as VGG-16 (Simonyan & Zisserman, 2014), which consists of 16 layers, could benefit from the ability of the Hybrid Shapley to compute actual Shapley values in later stages of pruning, although further investigation on the effects would be needed.

In order to further analyze the mechanics behind the decision making of the Shapley Criterion, additional information would have to be derived for discussion. One such way is to compute image-specific class salience maps (Simonyan et al., 2014) which show what features the architecture pays attention to during inference. The salience maps might provide more insight into what features the CNN deems to be important which could partially confirm whether the speculations drawn from subsection 3.2 are true.

4 Conclusion

The present study explored the effectiveness of the Hybrid Shapley Value pruning pipeline in a low-data setting in comparison to other Shapley methods for convolutional neural networks (CNNs) and examined the underlying decision-making mechanisms. The experiments were conducted using the MNIST dataset and a CNN architecture identical to LeNet-1, with the second convolutional layer pruned. The results showed that the Hybrid Shapley method performed significantly better than random pruning and slightly better than Shapley Value Monte Carlo Sampling. The feature maps produced by the filters were also analyzed to understand the network's decision-making process. Although the feature maps of the second convolutional layer were too abstract for any concrete analysis, the feature maps from the first convolutional layer potentially provide insights into the mechanisms behind the decision making of Shapley Value pruning algorithms. Overall, this work contributes to the ongoing research on improving the efficiency and interpretability of Shapley Value pruning algorithms.

References

Ancona, M., Öztireli, C., & Gross, M. (2020). *Shapley value as principled metric for structured network pruning.* arXiv. doi: 10.48550/ARXIV.2006.01795

Castro, J., Gómez, D., & Tejada, J. (2009). Polynomial calculation of the shapley value based on sampling. *Computers Operations Research*, *36*(5), 1726-1730. doi: https://doi.org/10.1016/j.cor.2008.04.004

Deng, X., & Papadimitriou, C. H. (1994). On the complexity of cooperative solution concepts. *Mathematics of Operations Research*, *19*(2), 257–266.

Gu, J., Wang, Z., Kuen, J., Ma, L., Shahroudy, A., Shuai, B., ... Chen, T. (2015). *Recent advances in convolutional neural networks.* arXiv. doi: 10.48550/ARXIV.1512.07108

Han, S., Pool, J., Tran, J., & Dally, W. J. (2015). *Learning both weights and connections for efficient neural networks.* arXiv. doi: 10.48550/ARXIV.1506.02626

Hu, H., Peng, R., Tai, Y.-W., & Tang, C.-K. (2016). *Network trimming: A data-driven neuron pruning approach towards efficient deep architectures.* arXiv. doi: 10.48550/ARXIV.1607.03250

James, F. (1980, sep). Monte carlo theory and practice. *Reports on Progress in Physics*, *43*(9), 1145. doi: 10.1088/0034-4885/43/9/002

LeCun, Y., Boser, B., Denker, J. S., Howard, R. E., Habbard, W., Jackel, L. D., & Henderson, D. (1990). Handwritten digit recognition with a back-propagation network. In *Advances in neural information processing systems 2* (p. 396–404). San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.

LeCun, Y., Denker, J. S., & Solla, S. (1990). Optimal brain damage. In D. Touretzky (Ed.), *Advances in neural information processing systems* (Vol. 2). Morgan-Kaufmann.

Li, H., Kadav, A., Durdanovic, I., Samet, H., & Graf, H. P. (2016). *Pruning filters for efficient convnets.* arXiv. doi: 10.48550/ARXIV.1608.08710

Mittal, D., Bhardwaj, S., Khapra, M. M., & Ravindran, B. (2018). *Studying the plasticity in deep convolutional neural networks using random pruning.* arXiv. doi: 10.48550/ARXIV.1812.10240

Molchanov, P., Tyree, S., Karras, T., Aila, T., & Kautz, J. (2016). *Pruning convolutional neural networks for resource efficient inference.* arXiv. doi: 10.48550/ARXIV.1611.06440

Shapley, L. S. (1952). *A value for n-person games.* Santa Monica, CA: RAND Corporation. doi: 10.7249/P0295

Simonyan, K., Vedaldi, A., & Zisserman, A. (2014). Deep inside convolutional networks: Visualising image classification models and saliency maps. In *Workshop at international conference on learning representations.*

Simonyan, K., & Zisserman, A. (2014, 09). Very deep convolutional networks for large-scale image recognition. *arXiv 1409.1556*.

Strumbelj, E., & Kononenko, I. (2010, mar). An efficient explanation of individual classifications using game theory. *Journal of Machine Learning Research*, *11*, 1–18.

Wertheimer, M. (1938). Laws of organization in perceptual forms. In *A source book of gestalt psychology* (p. 71-88). London.

5 Appendix

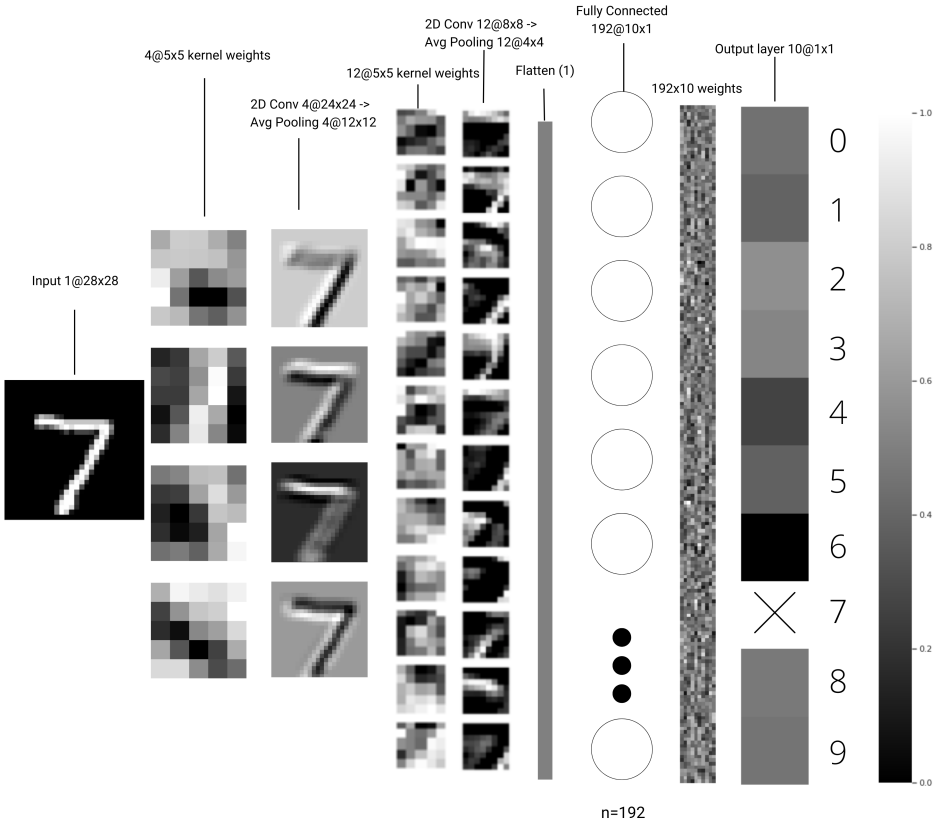

Figure 5.1: An illustration of the full LeNet-1 architecture inference process for a randomly selected sample of digit '7'. In our experiments, only the 12 filters in the second convolutional layer are pruned. The first layer remains untouched.

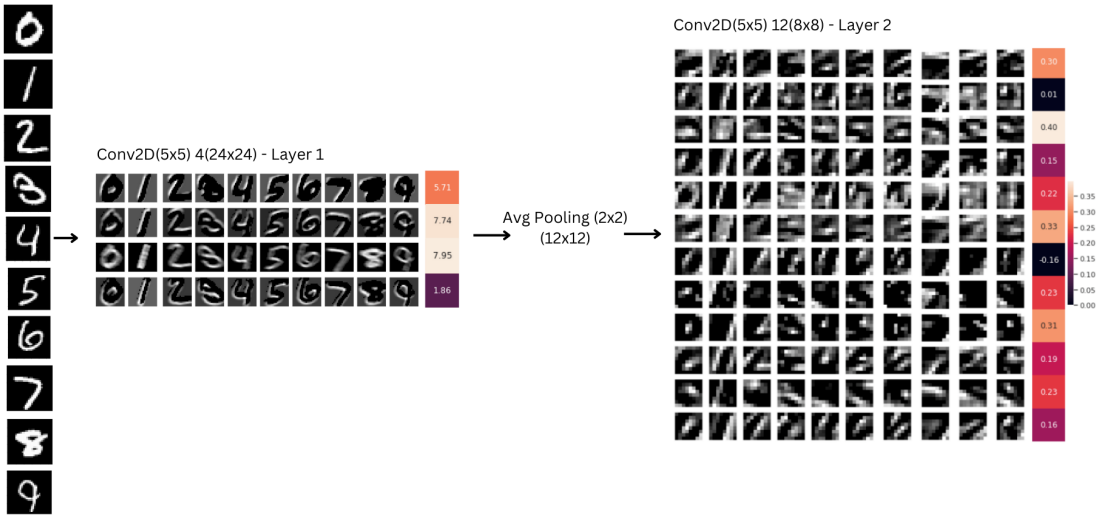

Figure 5.2: A figure illustrating the process of obtaining feature maps for each convolutional layer from the given inputs and their respective Shapley values obtained through Monte Carlo algorithm. Best seen in digital format.