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

The effect of unstructured pruning on the explanatory quality of heatmaps; a quantitative analysis

Bachelor's Project Thesis

Thijs Lukkien, s3978389, t.lukkien.1@student.rug.nl,
Supervisors: Matias Valdenegro-Toro; Marco Zullich

Abstract: The aim of this research was to quantitatively investigate how unstructured, magnitude-based, iterative pruning affected the visual explainability behavior of moderately-sized Convolutional Neural Networks (CNNs) in image classification. CNNs, and Deep Neural Networks (DNNs) in general, are black boxes that are expensive and environmentally demanding to use due to their computational complexity and large size. Unstructured pruning can be used to reduce network size by reducing density, and saliency maps can give a degree of explainability. However, the effect of pruning on such visual explanation methods is insofar done qualitatively. In this experiment, a small CNN was trained on MNIST and iteratively pruned, resulting in a series of models with sparsity percentages ranging from 100-0.09%. At each pruning level, saliency maps were generated and used to compute the intersection over union with the input image. This was then compared to the validation accuracy. It was hypothesized that unstructured pruning would have the same effect on the performance of gradient-based explainability methods as it has on model classification performance. The findings showed that there is a significant correlation between this explainability method and the validation accuracy. However, further research is needed to investigate the significance and implications of these findings.

1 Introduction

After the successful application of Convolutional Neural Networks (CNNs) by LeCun, Boser, et al. (1989), CNNs have been widely adopted for use in computer vision.

Since then, it has been shown that the performance of a Deep Neural Network (DNN) scales along with its size (Simonyan & Zisserman, 2014; Zagoruyko & Komodakis, 2016; He et al., 2015). CNNs are a type of DNN, and so general properties of DNNs can be applied to CNNs as well. As a result, modern CNNs can have over 100 million parameters (Simonyan & Zisserman, 2014), and over 100 layers of depth (Chollet, 2017; Szegedy et al., 2016).

Though efficient, such CNNs are very computationally expensive (Patterson et al., 2021). The high cost naturally leads to several problems with regard to creating, researching, or using large models. For example, any CNN-developing party with limited resources—think of developing countries, independent researchers, or small companies—will be at a disadvantage in the research or development of large models due to limited resources. Moreover, computationally expensive models require a lot of energy and can thus produce a lot of greenhouse emissions, as also shown by Patterson et al. (2021).

This goes to illustrate why large models should be reduced in size. One solution is to first train large CNNs and then apply compression techniques to reduce the model size. A commonly applied technique is called pruning, first introduced to neural networks by LeCun, Denker, & Solla (1989). Pruning is the process of removing DNN weights, nodes, channels, or filters (Wen et al., 2016). This can be subdivided into two categories: structured and unstructured pruning.

Unstructured pruning removes connections by setting specific weights to zero, whereas structured pruning removes groups of neurons. Hence,

structured pruning focuses on reducing model size, whereas unstructured pruning focuses on reducing DNN density.

As the model size directly influences how many computations need to be made, structured pruning is often looked to concerning the reduction of computational costs. However, there remain plenty of reasons to research unstructured pruning.

For example, recent developments in computer hardware have resulted in GPUs that have speed-ups on the multiplication of zero (Valero-Lara et al., 2018), allowing for efficiency increases by unstructured pruning. Moreover, unstructured pruning has been shown to increase CPU computational efficiency (Kurtic et al., 2022) and can be used to increase the performance of a model at high levels of sparsity in a variety of situations (S. Liu & Wang, 2023). Lastly, recent works (Chen et al., 2022) have proposed methods to convert unstructurally pruned DNNs into structurally pruned DNNs. Their methods were shown to reduce inference time on GPUs by up to 60% without a significant decrease in accuracy.

So far, the focus has mainly been on the performance of large models. However, it can be difficult to assess to what extent a model has truly learned the correct underlying concept using only model accuracy scores. For example, it is possible for a model to accurately classify images based on unimportant features. An example of this is the 'Clever Hans' behavior, where a model learns heuristics to correctly perform a task, using incorrect features (Lapuschkin et al., 2019).

A paper by Samek et al. (2017) argues that an explainable output is the best way to verify whether a model has truly learned the underlying concept that one desires. One such explainability method is the gradient-based Layer-wise Relevance Propagation (LRP). This method back-propagates through a model to find which input features (pixels, in this case) most strongly influenced the output (Bach et al., 2015). A commonly used implementation of this is called Grad-CAM (Selvaraju et al., 2017).

As unstructured pruning has a direct effect on the sparsity of connections in a CNN, it may have an effect on the quality of explainability analyses as well.

Previous research by Frankle & Bau (2019) used the dissection method to show that the number of neurons responding to specific concepts is directly linked to the model accuracy, regardless of the degree of unstructured pruning. Research by a (Li et al., 2020) has compared CNN visualization techniques with respect to quantification methods.

Saliency maps are often used as an explainability method, and pruning is often used to compress large models. However, the relationship of pruning on gradient-based methods, such as saliency maps, is still largely done qualitatively (Nauta et al., 2023).

This paper will focus on the quantitative correlation between the degree of sparsity and the performance of gradient-based explainability methods in image classification.

In other words: how does unstructured, magnitude-based, iterative pruning affect the visual explainability of moderately-sized CNNs in image classification?

It is argued by Samek et al. (2017) that a "good" model—one that has learned correct features—should perform equally well on explainability methods as on inference accuracy, as both performance measures are based on the extent of learned concepts. Research from Frankle & Bau (2019) has shown that this notion holds for the dissection method.

Therefore, it is hypothesized that unstructured pruning will have the same effect on the performance of gradient-based explainability methods as it has on model classification performance.

2 Materials and Methods

The techniques used in the experiment will be discussed. For each technique, a brief explanation as well as a formal definition, is given in the Materials subsection. In the Methods subsection, this information will then be placed in the context of the experiment.

The replicability details are discussed in the Appendix.

2.1 Materials

2.1.1 CNN

A Convolutional Neural Network (CNN) is a type of artificial neural network that works with the mathematical concept of convoluting two sets without flipping the final product. This is also known as

cross-correlation but more often simply referred to as convolution.

Each convolutional layer consists of filters. Each filter contains a number of kernels equal to the number of input channels. Kernels are 2D matrices, where each item represents a learned weight. If there is a number of channels c, kernel width w, and kernel height h, then the filter shape is $c \times w \times h$. As this experiment used gray-scaled images, there was one input channel, hence all input filters contained one kernel.

The process of convolution is performed by sliding the aforementioned kernels over an input image while computing the dot product at each step as shown in Equation 2.1.

$$y_{[m,n]} = x_{[m,n]} * h_{[m,n]} = \sum_{j=-\infty}^{\infty} \sum_{i=-\infty}^{\infty} x_{[i,j]} \cdot h_{[m-i,n-j]} \tag{2.1}$$

Here, '$y_{[m,n]}$' is the output value at pixel coordinates 'm' and 'n'. 'x' represents an input matrix, '$*$' the convolution operator, and 'h' the kernel. The convolution process results in a 2D output matrix for each kernel, an example of which can be seen in Figure 2.1.

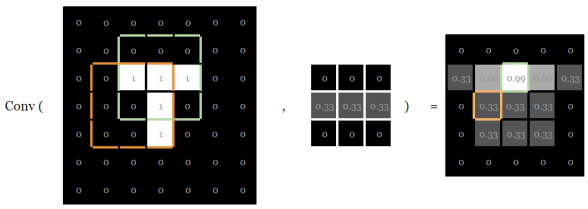

Figure 2.1: An example of the convolution process with one kernel. The orange and green boundaries show two dot products and their output values.

By changing the weights of a kernel, one can influence the produced outputs, also called feature maps. This name stems from the process in which a CNN works; kernel weights are adjusted in order to extract features from an input signal, and mapped onto the output signal. Hence, the output signal is referred to as the feature map. For example, Figure 2.1 shows the feature extraction of a horizontal bar which may be useful when trying to recognize the letters 'T', as shown in the example, or 'L'. The output feature maps are then passed through an activation layer before being used as input for the next layer. Often, the convolution process is combined with other layers such as maxpooling or flattening, which are outside the scope of this section.

2.1.2 ReLU

The Rectified Linear activation Unit (ReLU) is used to create a divide between which input signals are passed on to the next layer, and which are not. While being linear for values greater than zero, it creates non-linearity in the network by clamping all negative values to zero. The formula for the ReLU function can be seen in Equation 2.2.

$$f(x) = \max(0, x) \tag{2.2}$$

Where x is an input signal.

2.1.3 Optimizer

The optimizer used in this experiment is Stochastic Gradient Descent with Momentum (SGDM) to which weight decay was applied. Momentum takes previous gradients into account, in order to reduce oscillations in the loss optimization process. Weight decay adds a penalty to the loss value for large weights, resulting in smaller weights after training. This way, it prevents overfitting by regularization of weight size. It is described here as it is relevant for the weight update rule, although usually it is implemented with the loss function rather than the optimizer.

The weight update rule with these optimizer settings can be seen in Equation 2.3.

$$w_{t+1} \leftarrow w_t - \alpha(\nabla_{w_t}\mathcal{L} - \mu v - \lambda w_t) \tag{2.3}$$

Here, 'w_t' represents the set of weights 'w' at time step t. 'α' represents the learning rate. The SGD algorithm is implemented by '$w_t - \nabla_{w_t}\mathcal{L}$' where '$\nabla$' is the gradient of the loss function '$\mathcal{L}(X, Y)$' for output prediction X and ground truth Y. This way, the model takes a step towards a local minimum in the loss landscape for each iteration. The size of the step is determined by the learning rate. The loss function used in this research is cross-entropy, which is commonly used for categorical output. Momentum is implemented with the addition of μv, where μ represents the momentum coefficient, and

v represents the velocity from the previous iteration. As mentioned earlier, this addition allows the weight update rule to use the outcomes of previous weight update iterations, reducing oscillations and increasing the chances of finding better optima. Weight decay is implemented with λw_t, where λ is the weight decay coefficient and is used as a regularization method.

2.1.4 Grad-CAM and Grad-CAM++

Earlier, it was discussed how convolutional layers produce feature maps that contain spatial information about features of the input image. In a CNN, this spatial information is used to predict the class of an input image.

The final convolutional layer has the most finely-processed features and it is closest to the output of the model. Therefore, its feature maps are assumed to contain the most informative features for classification. For this reason, the feature maps of the final convolutional layer are used to generate saliency maps, from here on referred to as heatmaps.

The idea of Grad-CAM is to use these feature maps to display which features are most relevant for predicting the output class.

To do this, each feature map needs to be weighed. This is done with the partial derivative of output prediction with respect to the feature maps, which is called the gradient. This is different from the gradient that is usually referred to in neural networks, which is the partial derivative of the loss function with regard to the weights.

Computing this gradient results in a gradient matrix for each feature map. Global average pooling is then applied to find the average value describing the gradient matrix. This value will act as the weight for its corresponding feature map.

The weighted feature maps are then summed such that the result is one image containing all weighted features. This can be interpreted as the collection of all 'evidence' for the output class. To only allow positive 'evidence', features supporting the prediction of the output class, the summed feature maps are passed through the ReLU function.

The weights applied to each feature map are calculated with Equation 2.4.

$$\alpha_k^c = \frac{1}{Z} \sum_i \sum_j \frac{\partial y^c}{\partial A_{ij}^k} \tag{2.4}$$

Here, 'α_k^c' is the weight for output class 'c', and feature map '$_k$'. Gradient maps of the output prediction are calculated with the gradient '$\frac{\partial y^c}{\partial A_{ij}^k}$' where '$y^c$' is the model output and 'A_{ij}^k' is the set of each feature map '^{k}'. This results in a set of gradient maps that are summed by global average pooling '$\frac{1}{Z} \sum_i \sum_j$'.

The final heatmap is then calculated using Equation 2.5.

$$L_{\text{Grad-CAM}}^c = \text{ReLU}\left(\sum_k \alpha_k^c A^k\right) \tag{2.5}$$

Here, the feature map weights 'α_k^c' are applied to the feature maps 'A^k'. This is then summed, resulting in a heatmap containing all weighted features. Finally, this heatmap is passed through the *ReLU* to only allow positive evidence for features.

Grad-CAM++ extends the idea of Grad-CAM by using the gradients of all possible class predictions, instead of only the output prediction. Only positive gradients are used and each gradient is weighed individually before being averaged and summed similarly to the final steps of Grad-CAM. This can be seen in Equation 2.6.

$$L_{\text{Grad-CAM++}}^{kc} = \sum_i \sum_j \alpha_{ij}^{kc} \cdot \text{ReLU}\left(\frac{\partial Y^c}{\partial A_{ij}^k}\right) \tag{2.6}$$

2.1.5 Pruning

The pruning process used in this research is iterative pruning with weight reset. A pseudocode example is given in Algorithm 2.1.

Here, 'W_{start}' represents the weights matrix before pruning and retraining, and 'W_t^p' represents the weight matrix that was pruned ('^{p}') in the current iteration 't'. 'M' represents the pruning mask, and '$M_{t[i,j]}$' represents the mask value at row 'i' and column 'j'. The multiplication of the weights 'W' and mask 'M' is done element-by-element, also known as the Hadamard product. 'ϕ_t' represents the threshold value below which neurons will be pruned. 'μ' represents the pruning depth, determining what percentage of neurons should be pruned each iteration. However, this depends on how many connections can still be pruned without disabling the network. In the current experiment, the percentage of pruned neurons in each iteration is 20%

Algorithm 2.1 Pruning

$W_{start} \Leftarrow Pretrain(W_{random})$
$M_{start}[0, .., n] \Leftarrow 1$
while $t \neq t_{final}$ **do**
  $W_t^p \Leftarrow W_{start} \times M_{t-1}$
  $\phi_t \Leftarrow \mu \times \phi_{t-1}$
  $Model_t^p \leftarrow Retrain(W_t^p)$
  for all $(W_{t[i,j]}^p \in W_t^p) > 0$ **do**
    if $W_{t[i,j]^p} < \phi_t$ **then**
      $M_{t[i,j]} \Leftarrow 0$
    end if
  end for
end while

of prunable weights, resulting in a μ of 0.8. 'Model_t^p' represents the pruned ('^{p}') output model at iteration 't'.

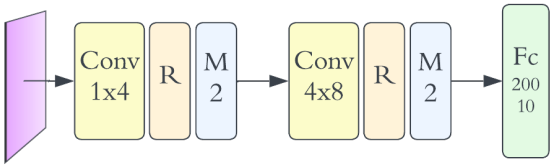

Figure 2.2: A visual representation of the model structure. 'Conv 1x4' represents a convolutional layer with 1 input channel and 4 output channels. 'R' represents a *ReLU* layer, 'M 2' represents a Maxpooling layer with a kernel size of 2, and 'Fc 200 10' represents a Fully connected layer with 200 input nodes and 10 output nodes. The input of the 'Fc' layer is flattened by a Flattening layer.

2.2 Methods

2.2.1 Data

For the experiment, data from the *MNIST* dataset (Deng, 2012) was. *MNIST* is an image classification dataset that contains 10 classes, 60 000 training images, and 10 000 validation images. The images are gray-scale, with a size of 28×28 pixels.

The dataset was augmented by applying light noise to every batch. The noise was only applied to background pixels—represented as zero—as increasing pixel values larger than zero risked these values overflowing the maximum value, thereby changing or removing input patterns. The motivation behind applying noise is given in Section 2.2.4.

The applied noise had a magnitude that ranges between 0% and 20% of the maximum pixel value of the input image. This percentage was arbitrarily chosen, although higher percentages were found to cause the network to classify images based on noise patterns surrounding the pixels that would otherwise characterize the input class, hence influencing the original classification task.

2.2.2 Model

The model was created using the OpenLTH library (Frankle, 2020) and makes use of the Pytorch (Paszke et al., 2019) CNN module. A detailed description of CNNs is given in Section 2.1.

The CNN model structure consists of two 2D convolutional layers, a flattening layer, and one fully connected layer. Each convolutional layer is followed by a Rectified Linear activation Unit (ReLU) and Maximum Pooling (Maxpooling) layer respectively. The first convolutional layer accepts input from 1 channel and has an output of 4 channels. The second convolutional layer accepts input from 4 channels and has an output of 8 channels. Moreover, both convolutional layers have a kernel size of 3, a stride of 1, and no padding. The Maxpooling layers have a kernel size of 2, and the fully connected layer has an input size of 200 and an output size of 10. The model structure is visually represented in Figure 2.2.

2.2.3 Training and pruning

The training, pruning, and noiseless validation processes were performed with the *openLTH* library. A visual representation of the pruning pipeline can be seen in Figure 2.3.

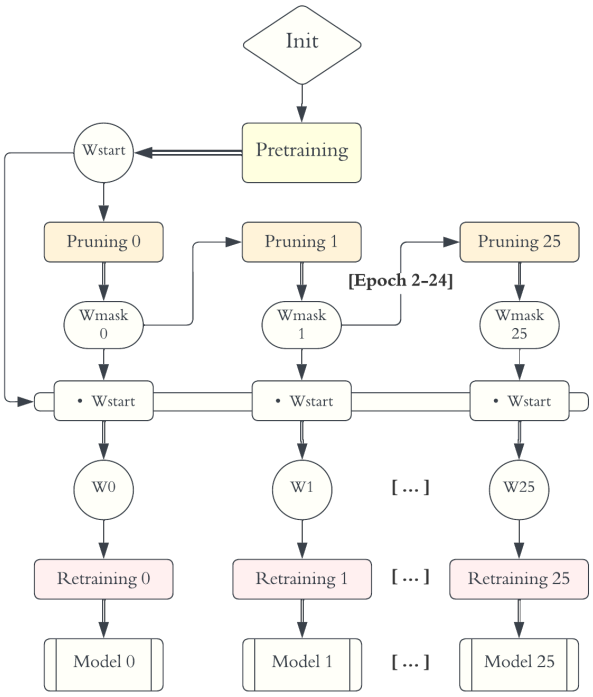

Figure 2.3: A visualization of the pruning pipeline. 'Init' stands for the initialization process of the model. 'Wstart' is the set of weights after pretraining. 'Pruning0' stands for pruning iteration 0. 'Wmask' is a mask, generated by the pruning process, that can be applied to the weight matrix. 'W0' is the set of pruned weights after pruning iteration 0. 'Model0' is the final model with pruned weights at iteration 0. Finally, '[...]' represents all the models and processes between iterations 1 and 25.

The model was trained using Stochastic Gradient Descent with Momentum, to which weight decay was applied. The training and pruning details are discussed in Section 2.2.5 and the pruning process is formally described in Section 2.1.

2.2.4 Explainability

The approach used to quantify explainability, was to compute the overlap between a generated heatmap and its input image. To generate such heatmaps, Gradient-weighted Class Activation Mapping Plus Plus (Grad-CAM++) was used. Grad-CAM++ was introduced by Chattopadhay et al. (2018) as an extension of Grad-CAM by Selvaraju et al. (2019). The technical descriptions of both methods can be found in Section 2.1. In this experiment, the implementation from (Gildenblat & contributors, 2021) was used.

The output of this method is a heatmap of the same size as the input image. This heatmap highlights certain areas, based on where important features were localized on the input image. This could then be compared to a segmentation map of the input image in order to calculate the overlap between the explainability method and ground truth.

Earlier it was discussed how explainability is quantified as the overlap between a heatmap and the ground truth. A common metric for quantifying such overlap between two areas is called the Intersection over Union (IoU) method (Zhu et al., 2019; Rahman & Wang, 2016). The IoU formula can be seen in Equation 2.7, where A and B are the two images being compared.

As A and B were in the range $[0, 1]$, the binary operations from Equation 2.7 were replaced with the *min* and *max* operators. The adapted formula can be seen in Equation 2.8.

$$IoU = \frac{A \cup B}{A \cap B} \tag{2.7}$$

$$IoU = \frac{\min(A, B)}{\max(A, B)} \tag{2.8}$$

The Grad-CAM heatmap was produced using the *GradCamPP* function from the aforementioned Pytorch-grad-cam library (Gildenblat & contributors, 2021). An example output sample can be seen in Figure 2.4.

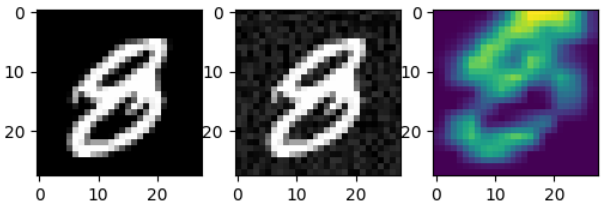

Figure 2.4: A sample of the data. The original input image (left), with noise applied (center), and its resulting heatmap (right).

As mentioned in section 2.2.1, noise was applied in data augmentation. The reason for this was that heatmaps are generated from the gradient-weighed feature maps. If there were only zero-valued pixels, the convolution process would always result in a

zero-valued output. In other words: a black background would never result in features being detected there. This meant that the network would be unable to display whether it had truly learned to discriminate between features of an input image and other input signals. As mentioned earlier, it was believed that adding noise could help navigate this issue.

2.2.5 Experimental settings

Before pruning, the model was pretrained for 10 epochs. The training was done using Stochastic Gradient Descent with Momentum (SGDM), with a learning rate of 0.01 over epochs 0 to 8, and a learning rate of 0.001 over epochs 9 and 10. Lastly, the momentum was set to 0.9, and a weight decay of 0.0001 was applied. A detailed description of SGDM and weight decay is given in Section 2.1.

The pruning method used in this experiment was iterative pruning. More specifically; iterative pruning with weight reset. This meant that each iteration of the pruning process started with the weight-pruning mask from the previous pruning iteration and the weights that resulted from pretraining. The masks were applied to the weight matrices throughout the retraining phase, such that a weight of zero—a pruned weight—would not be updated to a non-zero value. Each iteration pruned 20% of weights from the previous iteration, and the model was pruned for 25 iterations. A formal introduction of this pruning method is given in 2.1.

After pruning the result of pretraining, retraining continued at epoch 11 with the same hyperparameters as pretraining. However, the learning rate was set to 0.01 at epochs 11 to 15, 0.001 at epochs 16 to 18, and 0.0001 at epochs 19 and 20.

At full density, pruning iteration 0, the model achieved a mean validation accuracy of 98.17% on input data without noise, and 98.18% on input data with noise.

The validation accuracy was measured on data with and without the noise transformation to ensure that the augmentation pipeline did not affect the original classification task. This was crucial, as this experiment aimed to find a correlation between the original classification task, pruning, and visual explanation methods.

Data was loaded in batches of 128 and for each iteration of pruning, the validation accuracy was computed on the same dataset as the explanation accuracy.

2.2.6 Encompassing

The model was trained on data with light noise, to a validation accuracy of 98.17%. It was then pruned at 25 levels using iterative, magnitude-based pruning with weight reset. The 26 resulting models were then used to generate heat maps from data with noise. Lastly, the heatmaps were compared to the input images without noise using the intersection over union method.

3 Results

The goal of this paper was to investigate whether there is a quantitative correlation between the degree of sparsity and the performance of gradient-based explainability methods in image classification.

It was hypothesized that unstructured pruning would have the same effect on the performance of gradient-based explainability methods as it has on model classification performance.

To find out, the average validation accuracy was compared to the IoU scores for each iteration of pruning. The results can be found in Figures 3.1 and 3.2.

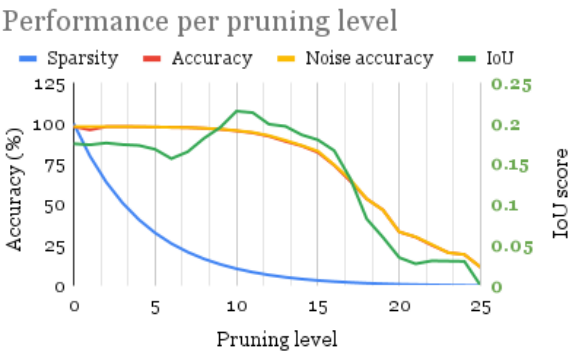

Figure 3.1: The resulting accuracies and IoU scores per level of pruning. All data points are spaced equally from one another on the horizontal axis, as can be seen from the decreasing sparsity series.

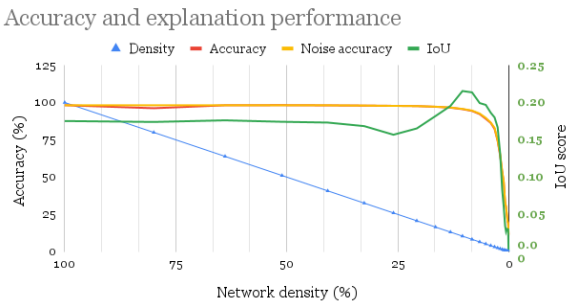

Figure 3.2: The resulting accuracies and IoU scores against network density. The density series shows that most data points were located nearer to zero percent density.

To find whether the IoU score truly followed the same pattern as the validation accuracy, the correlation was calculated on the aggregated means per pruning level. In other words: the experiment was run three times, resulting in three series of 26 accuracies and IoU scores. The final results consist of the averaged three runs, resulting in a final series of 26 averaged accuracies and IoU scores. The average scores are then tested for correlation using the Pearson tests. The results can be seen in Table 3.1

	ρ	p-value	df
Pearson correlation	0.960	$< .001$	25

Table 3.1: Results of the Pearson Correlation test. 'df' is the degrees of freedom and 'ρ' is the Pearson correlation coefficient.

As can be seen from Table 3.1, there is a significant correlation between the IoU-score and the validation accuracy of the model.

It should be noted that the IoU and Accuracy series were not normally distributed.

4 Conclusion and discussion

This paper focused on the quantitative correlation between the degree of sparsity and the performance of gradient-based explainability methods in image classification.

It was hypothesized that the behavior of gradient-based explainability would be similar to that of predictive accuracy.

The findings of this study revealed a significant similarity between the effect of pruning on predictive accuracy and explainability accuracy. Based on these results, the null hypothesis can be rejected.

This is in alignment with the findings of Frankle & Bau (2019) and the argumentation of Samek et al. (2017).

While these results are promising, some limitations must be acknowledged.

Limitations Firstly, as can be seen from Figures 3.2 and 3.1, the IoU score reached a maximum average of 0.21. That is 21% of the maximum score it could have achieved with a perfect fit to the input sample. It could be argued that such a low IoU score is insufficiently representative of explanatory behavior. For example, a heatmap with a single pixel could potentially show similar behavior to the current IoU series, at lower scores. Reversely, a qualitatively good explanatory heatmap can produce a relatively low IoU. In Figure 2.4 it can be seen how the heatmap shows a clearly distinguishable number eight. However, this picture results in an IoU of 0.18 as it is skewed by a few pixels. This shows the possible discrepancy between a heatmap-segmentation analysis and explanation accuracy.

Connected to the previous issue, is that the mapping of a heatmap onto the segmentation map can influence the IoU as much as the explanatory quality of the heatmap itself.

This way, more precise heatmaps could be slightly mismapped to the segmentation map, resulting in a lower IoU score than expected. This effect is likely what caused the explanation score to start lower before peaking after the first six pruning levels.

To investigate this phenomenon, heatmaps from pruning iteration 10 were generated. An example of those heatmaps, as seen in Figure 4.1, shows how heatmaps generated by a sparse network are more blurred out and thus easier to map, resulting in higher IoU scores. This would align with the difference in the standard deviation of heatmaps between iterations 0 and 10: 0.218 and 0.252 respectively.

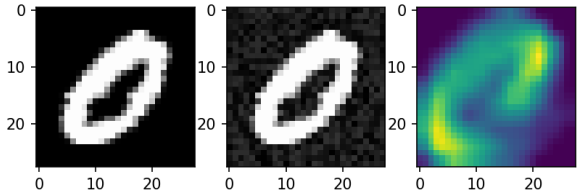

Figure 4.1: A sample of the data on iteration 10. A raw input image (left), the input image with noise (center), and its corresponding heatmap (right). This heatmap has an IoU score of 0.33**.**

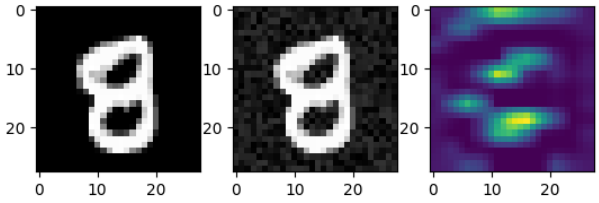

Figure 4.2: A sample of the data on iteration 0. A raw input image (left), the input image with noise (center), and its resulting heatmap (right). Note how the heatmap shows activation only around the figure of the number. This heatmap had an IoU score of 0.15**.**

Another possibility is that more 'negatives' were generated in early iterations, which would explain the difference in means between iterations 0 and 10: 0.179 and 0.236 respectively.

Furthermore, one could argue that the MNIST dataset is an over-simplistic dataset. Its small images, gray-scale and textureless surfaces, and easily recognizable features can each affect the quality of this experiment.

As the images are relatively small—28 by 28 pixels—it is relatively difficult for the heatmap to predict pixels that miss the segmentation map completely. This means that the Grad-CAM model method is not given much room to perform poorly. Input image size was found to influence CNN validation accuracy by Richter et al. (2021), perhaps influencing the performance of Grad-CAM as well.

Normally, color and texture play a significant role in the learned patterns of a CNN (L. Liu et al., 2019). As the numbers displayed in input images do not have colored or textured surfaces, the ability of the CNN to recognize classes based on their environment may be more prominent than is realistic in a normal use case. For example, the model may learn to recognize when the noise starts and the number ends. While this edge is a usable pattern, it would be preferable if the model recognized numbers based on the pixels that actually constitute the number. Recognizing classes based on their environment may also influence the explanatory quality of heatmaps, an example of which can be seen in Figure 4.2.

Implications The findings of this experiment imply that one can prune a CNN up to a point 98% sparsity without losing accuracy or explanatory quality. Previously, it was discussed how pruned networks could save memory, reduce inference costs, speed up CPU computation, potentially speed up GPU computation, reduce emissions, and allow for efficient model conversion to a structurally pruned model to directly reduce GPU computational costs and emissions. Such consequences would be relevant for anyone aiming to work with explainable CNNs.

Suggestions for future research The first suggestion would be to further investigate this topic using other datasets.

A dataset with larger input images should allow more room for error with the heatmap method, while also being more realistic compared to real use cases. Extending that idea, images with more patterns than solely the target class pattern would be preferable as such patterns could 'distract' the model.

As discussed earlier in this section, colored or textured input images could allow the CNN to learn patterns besides spatial features, such as the stripes on a tree. This could help negate the issue of background pixels—with a value of zero—being fundamentally unable to generate heatmaps. This may especially be relevant to prevent negative heatmaps such as the example in Figure 4.2.

Another option would be to experiment with different methods of heatmap-segmentation fitting. However, such processes could be at risk of overfitting. One could learn how to fit qualitatively bad

predictions to an input image, hence producing an unrepresentative amount of overlap for the explanation score. However, the opposite could also be true: if done correctly, it could allow for an IoU score to better represent the true explanatory quality of a heatmap.

Furthermore, it has been discussed in the Introduction how pruning could be beneficial for, for example, storage space saving, reducing inference costs and emissions, and more. One could investigate the extent to which the benefits of a pruned model apply while retaining inference and explanatory quality. An example would be converting an unstructurally pruned model to a structurally pruned model. This would improve traditional GPU inference costs, though may also decrease performance.

Moreover, One could further investigate the correlation between inference and heatmap quality. An example of this would be to investigate whether correct inferences are more likely to result in heatmaps of better quality than incorrect inferences. This could provide further evidence of a correlation between inference accuracy and the quality of produced explanations.

Finally, the correlation was computed using a parametric test, on data that was not normally distributed. A suggestion for future research is thus to investigate the current findings using a different set of experiments and tests.

References

Bach, S., Binder, A., Montavon, G., Klauschen, F., Müller, K.-R., & Samek, W. (2015, July). On Pixel-Wise Explanations for Non-Linear Classifier Decisions by Layer-Wise Relevance Propagation. *PLOS ONE*, *10*(7), 1–46. (Publisher: Public Library of Science) doi: 10.1371/journal.pone.0130140

Chattopadhay, A., Sarkar, A., Howlader, P., & Balasubramanian, V. N. (2018, mar). Grad-CAM: Generalized gradient-based visual explanations for deep convolutional networks. In *2018 IEEE winter conference on applications of computer vision (WACV)*. IEEE. doi: 10.1109/wacv.2018.00097

Chen, T., Chen, X., Ma, X., Wang, Y., & Wang, Z. (2022, 17–23 Jul). Coarsening the granularity: Towards structurally sparse lottery tickets. In K. Chaudhuri, S. Jegelka, L. Song, C. Szepesvari, G. Niu, & S. Sabato (Eds.), *Proceedings of the 39th international conference on machine learning* (Vol. 162, pp. 3025–3039). PMLR.

Chollet, F. (2017). *Xception: Deep learning with depthwise separable convolutions.*

Deng, L. (2012). The mnist database of handwritten digit images for machine learning research. *IEEE Signal Processing Magazine*, *29*(6), 141–142.

Frankle, J. (2020, 5). *OpenLTH: A Framework for Lottery Tickets and Beyond.* Retrieved from `https://github.com/facebookresearch/open_lth`

Frankle, J., & Bau, D. (2019). Dissecting Pruned Neural Networks. doi: 10.48550/ARXIV.1907.00262

Gildenblat, J., & contributors. (2021). *Pytorch library for cam methods.* (commit 2183a9c)

He, K., Zhang, X., Ren, S., & Sun, J. (2015). Deep residual learning for image recognition. *CoRR*, *abs/1512.03385*.

Kurtic, E., Campos, D., Nguyen, T., Frantar, E., Kurtz, M., Fineran, B., ... Alistarh, D. (2022, December). The optimal BERT surgeon: Scalable and accurate second-order pruning for large language models. In *Proceedings of the 2022 conference on empirical methods in natural language processing* (pp. 4163–4181). Abu Dhabi, United Arab Emirates: Association for Computational Linguistics.

Lapuschkin, S., Wäldchen, S., Binder, A., Montavon, G., Samek, W., & Müller, K.-R. (2019, March). Unmasking Clever Hans predictors and assessing what machines really learn. *Nature Communications*, *10*(1), 1096. doi: 10.1038/s41467-019-08987-4

LeCun, Y., Boser, B., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W., & Jackel, L. D. (1989). Backpropagation applied to handwritten zip code recognition. *Neural Computation*, *1*(4), 541-551. doi: 10.1162/neco.1989.1.4.541

LeCun, Y., Denker, J., & Solla, S. (1989). Optimal brain damage. *Advances in neural information processing systems*, *2*.

Li, X.-H., Shi, Y., Li, H., Bai, W., Song, Y., Cao, C. C., & Chen, L. (2020). Quantitative Evaluations on Saliency Methods: An Experimental Study.
doi: 10.48550/ARXIV.2012.15616

Liu, L., Chen, J., Fieguth, P., Zhao, G., Chellappa, R., & Pietikäinen, M. (2019, January). From BoW to CNN: Two Decades of Texture Representation for Texture Classification. *International Journal of Computer Vision*, *127*(1), 74–109. doi: 10.1007/s11263-018-1125-z

Liu, S., & Wang, Z. (2023). *Ten lessons we have learned in the new "sparseland": A short handbook for sparse neural network researchers.*

Nauta, M., Trienes, J., Pathak, S., Nguyen, E., Peters, M., Schmitt, Y., ... Seifert, C. (2023, February). From Anecdotal Evidence to Quantitative Evaluation Methods: A Systematic Review on Evaluating Explainable AI. *ACM Computing Surveys*, 3583558. doi: 10.1145/3583558

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., ... Chintala, S. (2019). Pytorch: An imperative style, high-performance deep learning library. In *Advances in neural information processing systems 32* (pp. 8024–8035). Curran Associates, Inc.

Patterson, D., Gonzalez, J., Le, Q., Liang, C., Munguia, L.-M., Rothchild, D., ... Dean, J. (2021). Carbon emissions and large neural network training. *arXiv preprint arXiv:2104.10350*.

Rahman, M. A., & Wang, Y. (2016). Optimizing intersection-over-union in deep neural networks for image segmentation. In *International symposium on visual computing* (pp. 234–244).

Richter, M. L., Byttner, W., Krumnack, U., Wiedenroth, A., Schallner, L., & Shenk, J. (2021). (input) size matters for CNN classifiers. In *Lecture notes in computer science* (pp. 133–144). Springer International Publishing. doi: 10.1007/978-3-030-86340-1$_1$1

Samek, W., Binder, A., Montavon, G., Lapuschkin, S., & Muller, K.-R. (2017, November). Evaluating the Visualization of What a Deep Neural Network Has Learned. *IEEE Transactions on Neural Networks and Learning Systems*, *28*(11), 2660–2673. doi: 10.1109/TNNLS.2016.2599820

Selvaraju, R. R., Cogswell, M., Das, A., Vedantam, R., Parikh, D., & Batra, D. (2019, oct). Grad-CAM: Visual explanations from deep networks via gradient-based localization. *International Journal of Computer Vision*, *128*(2), 336–359. Retrieved from

Selvaraju, R. R., Das, A., Vedantam, R., Cogswell, M., Parikh, D., & Batra, D. (2017). *Grad-cam: Why did you say that?*

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

Szegedy, C., Ioffe, S., Vanhoucke, V., & Alemi, A. (2016). *Inception-v4, inception-resnet and the impact of residual connections on learning.*

Valero-Lara, P., Martínez-Pérez, I., Sirvent, R., Martorell, X., & Peña, A. J. (2018). NVIDIA GPUs Scalability to Solve Multiple (Batch) Tridiagonal Systems Implementation of cuThomasBatch. In R. Wyrzykowski, J. Dongarra, E. Deelman, & K. Karczewski (Eds.), *Parallel Processing and Applied Mathematics* (pp. 243–253). Cham: Springer International Publishing.

Wen, W., Wu, C., Wang, Y., Chen, Y., & Li, H. (2016). *Learning structured sparsity in deep neural networks.* doi: https://doi.org/10.48550/arXiv.1608.03665

Zagoruyko, S., & Komodakis, N. (2016). Wide residual networks. *arXiv preprint arXiv:1605.07146*.

Zhu, Y., Sapra, K., Reda, F. A., Shih, K. J., Newsam, S., Tao, A., & Catanzaro, B. (2019). *Improving semantic segmentation via video propagation and label relaxation.*

A Appendix

A.1 Reproducibility and replicability

The process of training, validation, heatmap generation, and explainability computation was performed three separate times on seeds 333, 334, and 335. During each experiment, one seed was used for all pseudo-random methods mentioned in this research. The experiment was repeated to reduce the risk of false positive findings. The final results were produced by averaging the results of these three experiments.

For the current dataset, the maximum pixel value was 1.0, so the applied noise was pseudo-randomly sampled from the range $[0, 0.2]$. The pseudo-random sampling was done with the *randint* method of Pytorch (Paszke et al., 2019).

B Multiple appendices