Technische Hochschule Ingolstadt

AImotion Bavaria

Master Thesis

im Studiengang Informatik (INF)
Fakultät Informatik

A Dual Study of Filter Capacity and Adversarial Robustness on CNN Models Using Pattern-Based Pruning

Vor- und Zuname: **Knauer Dominik**

ausgegeben am: 16.07.2024

abgegeben am: 14.08.2024

Erstprüfer: Prof. Dr.-Ing. Richard Membarth

Zweitprüfer: Tobias Pfaller

Affidavit

I hereby declare that this thesis is my own work, that I have not presented it elsewhere for examination purposes and that I have not used any sources or aids other than those stated. I have marked verbatim and indirect quotations as such.

Ingolstadt, _____
(Date, Signature)

Dominik Knauer

Acknowledgments

I would like to take this opportunity to thank everyone who supported and accompanied me during the creation of this thesis. First and foremost, I want to express my gratitude to my family, who have always supported me and provided me with the strength needed to get through this intense period. My special thanks go to my mother, Helene Knauer, my father, Viktor Knauer, and my brother, Maximilian Knauer, who met my 'questionable' behavior and inner turmoil during this time with great understanding.

A heartfelt thank you goes to my supervisor, Tobias Pfaller. I am immensely grateful for the tremendous dedication and extensive support you provided throughout my master's thesis. Our numerous discussions about progress, new approaches, and theoretical backgrounds were invaluable and significantly contributed to the success of this work.

I would also like to thank my friends who stood by me during this long journey, despite the social neglect, and showed great understanding. A special thanks to Armin Softic, Bence Dornai, Daniel Schuktuew, Fatih Gültekin, Jetmir Raimi, Kevin Kertesz, and Marcello Umali.

Another thank you goes out to my fellow students. It has been a long and often exhausting journey, and I am grateful to have shared my studies with you. Without your support, I probably wouldn't have made it this far. A big thank you to Ahmet Ucar, Benjamin Puskarevic, Burak Topuz, Dogan Günes, Gent Lushi, Marco Bäck, Oguzhan Topcu, Patrick Schenker, Waseem Krabler, and Yasin Eroglu.

Finally, I would like to thank all those who indirectly supported me during this work and whose support was more behind the scenes.

Thank you all!

Dominik Knauer

Abstract

This thesis investigates the adversarial robustness and filter capacity of Convolutional Neural Networks (CNN) using a pattern-based pruning approach. Modern CNNs, which are essential in image processing for safety-critical applications like autonomous driving, are susceptible to perturbations and targeted adversarial attacks that can weaken their accuracy and reliability. To address these challenges, an approach based on reducing specific filter weights selected by patterns was developed to increase the efficiency of neural networks without significantly compromising classification accuracy. The Alternating Direction Method of Multipliers algorithm was used to optimally apply this pruning technique while preserving the model's robustness against adversarial attacks. The results show that pattern-based pruning methods not only improve the memory and computational efficiency of CNNs but can also enhance their precision efficiency and resilience against adversarial attacks. Adversarial training as well as filter kernel pruning were integrated into the pattern pruning algorithm to further explore adversarial robustness and the possibility of higher computational efficiency through increased pruning rates, but were considered in a limited way as side information. Additionally, I developed a new evaluation method based on Channel-wise Increases in Confidence, which enables a fine-grained analysis of filter performances within a convolutional layer. This work provides important insights for the further development of robust and efficient CNN optimization techniques in safety-critical applications.

Table of contents

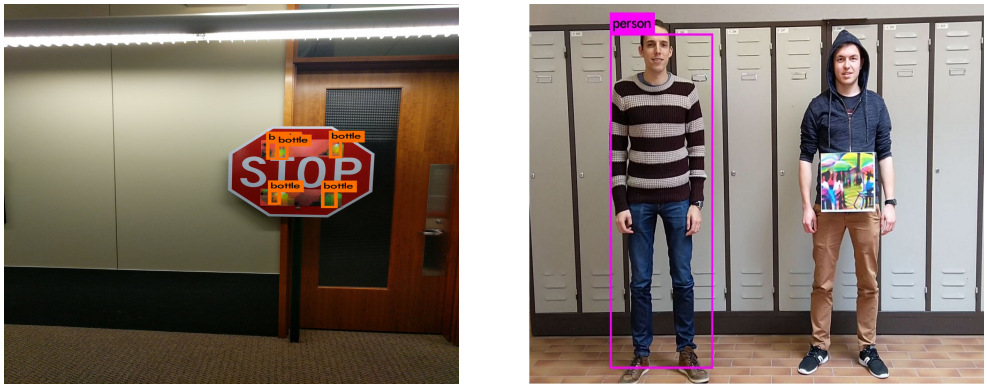
1. Introduction

In recent years, advancements in artificial intelligence (AI) have reached significant proportions. AI technologies, such as machine learning and deep learning, have revolutionized various industries from healthcare to finance. The advancement of neural networks and algorithms has significantly enhanced the performance of AI systems. These advancements have enabled the efficient processing of vast amounts of data, yielding valuable insights and forming the basis for numerous innovative applications. A particularly exciting application area for these AI developments is the automotive industry. In recent years, this industry has made significant strides toward autonomous driving. Modern vehicles are increasingly equipped with advanced driver assistance systems that automate various aspects of driving, thereby enhancing safety and comfort. The vision of a fully autonomous vehicle that can navigate without human intervention is drawing ever closer. Technologies such as lidar, radar and camera-based systems play a central role because they can analyze and respond to the vehicle's surroundings in real-time. Despite these impressive advancements, the implementation of fully autonomous vehicles is fraught with numerous technical challenges. Camera-based systems must be able to reliably detect and interpret traffic situations, which is a central challenge. These systems, which rely on deep learning and specifically *Convolutional Neural Networks (CNNs)*, are susceptible to various types of disturbances that can impair their accuracy and safety. Weather conditions such as rain or fog, poorly visible road markings and unexpected obstacles are just a few of the many issues that need to be addressed. Even though these points may seem minor, these minor disturbances can have serious consequences for people in the vicinity of autonomous vehicles. This is not just about natural disturbances. Vandalism, whether intentional or unintentional, can also cause disturbances. A simple sticker placed in the correct spot on a stop sign can lead to the most absurd misclassifications. Although a normal human can easily distinguish, a CNN requires some additional optimization to classify the big picture Fig. 1. The neural network could mistakenly classify a stop sign as a no-parking sign, a

Figure 1: False Classified Stop Sign by a Neural Network. Source: [1].

flag or a fish. Some more examples are provided in Sub-figures 2, where pictures were placed on the stop sign Fig. 2a or as in Fig. 2b, a disruptive image was held by the test person. In both cases, the neural network is unable to identify traffic-relevant objects. Damaged traffic signs can affect

the image processing systems in such a way that they misinterpret the situation and thus initiate potentially dangerous driving maneuvers. As described in the study by Silva and Najafirad, even slight changes in the input image can drastically influence the prediction outcome [2].



(a) Disruptive Images Catching Focus of the Neural Network. Source: [3]

(b) False Classification Caused by a Disruptive Image. Source: [4]

Figure 2: Snapshots of False Predictions for CNN-Based Yolo Object Tracking Algorithm

There are no limits to the misinterpretations made by neural networks, except for the number of programmed classification possibilities. Here, the term *adversarial attacks* is introduced. This term is the subject of its own research field, which seeks to identify vulnerabilities in neural networks, investigate them and develop countermeasures. In the following chapters, adversarial attacks will be further elaborated upon, as they are an important part of the evaluation in this research work. To more closely examine these vulnerabilities and measure the robustness of CNNs, adversarial images are generated. These are faulty inputs designed to deliberately expose weaknesses in the neural network. By analyzing these images, I aim to investigate where the vulnerabilities of the CNNs lie, how to mitigate them and how effective the optimizations are. A promising approach for increasing robustness is the so-called *pattern pruning*, which was introduced by Ma et al. [5]. According to the results from [5], targeted pruning of filters within the CNNs is expected not only to enhance the capacity of the filters but also to enable a more resource-efficient and effective use of CNNs on microcontrollers. These microcontrollers are often the central processing units in autonomous vehicles, whose computing power and memory are limited. Pattern pruning, as the term suggests, is a special form of weight pruning. Pruning cannot be conducted randomly. It must follow a certain methodology and be applied in a controlled manner, as otherwise, it would significantly impair the classification ability of the model. Mathematically, it represents a combinatorial optimization problem that must be solved. For this purpose, I applied *the alternating direction method of multipliers (ADMM)*. ADMM is a numerical optimization method and is often referred to as a state-of-the-art algorithm for solving convex optimization problems [6]. In contrast to random disturbances, which are nonspecific and may have only minor or random effects, adversarial attacks are specifically designed to test the robustness of a model under the worst-case scenarios. The optimization with pattern pruning may introduce new vulnerabilities, which can be appropriately tested based on adversarial samples.
Can pattern pruning enhance the robustness and filter capacity of convolutional neural networks against adversarial attacks?
To answer this question proper methods are needed to evaluate the pattern pruning optimization. My investigations into filter capacity were conducted using general examples, without spe-

cific scenarios from the automotive industry. Adversarial attacks were used as test examples for manipulated model inputs to evaluate the robustness and effectiveness of the filter optimizations. For evaluation purposes, I utilized techniques for visualizing activation maps such as *Grad-CAM++* [7] [8] *(-Class-Activation-Mapping)*, *Score-CAM* [9] and *feature maps*. Additionally, I developed an evaluation method, *Channel-wise Increase of Confidence*, based on Score-CAM's score assignment mechanism, to facilitate detailed benchmark comparisons of the filters and layers across various CNNs [9]. As mentioned earlier, I assess the impact of the optimizations on adversarial samples to examine how reliably the optimized models respond to artificial disturbances. Besides improving the filter capacity, it is also crucial to maintain the model's accuracy. For this reason, accuracy is also examined in-depth. As a bonus for broader investigations into pattern pruning optimization, I utilize *Adversarial Training* to incorporate another method known for improving the robustness of models [2].

2. Theoretical Foundations

In this chapter, fundamental concepts and theories are presented to build a basic understanding and overview of the topics covered in this work.

- I explain the term filter capacity, describe convolutional filters, provide examples regarding filter capacity and demonstrate how a CNN with good filter capacity differs from one with poor filter capacity.
- Under the chapter Adversarial Attacks/Defenses, I explain relevant terms and metrics, the structures of the research field, types of adversarial attacks, types of adversarial defenses and the fundamental mechanisms behind them. I reference the parallels of my optimization and point out the broader spectrum of fundamentals in the context of this work.
- In the chapter on pruning, I explain the diversity of pruning, why pruning is a common approach in model training and compare the various types with their advantages and disadvantages.
- The chapter on ADMM provides a brief overview of the key features of the state-of-the-art optimization method. For simplicity's sake, I use unstructured magnitude pruning to show how ADMM finds a nearly optimal solution to the combinatorial pruning problem while training the model at the same time.

Where possible, connections are made to the project. These connections are kept brief and will be discussed in more detail in the main section.

2.1. Filter Capacity

In the scientific discipline of computer vision, filters and their properties constitute central elements. Filters can be used to smooth images, highlight edges and enhance image details. For example, box and Gaussian filters smooth images by reducing noise [10, chap. 17.2, 17.3], while Sobel and Laplacian filters detect edges and can better recognize edges in image regions with rapid brightness changes [10, chap. 18.7, 18.9]. The Retinex filter, on the other hand, separates reflection and illumination components, improving the perception of colors and brightness in images, especially under varying lighting conditions [10, chap. 18.12]. Filters can be represented as 2D matrices and can also be applied to images through convolution. An example of such a convolution applicable matrix is shown by the 2D-Binomial Filter below [10, chap. 18.5].

$$\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

Figure 3 shows a base image with a checkerboard pattern overlay. The box filter and the 2D binomial filter are applied to remove the disturbance caused by the checkerboard pattern. It is clearly visible how the application of the smoothing filters reduces the disturbance.

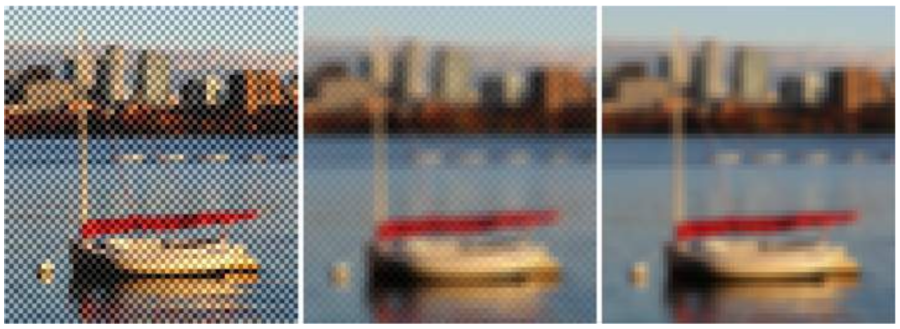


Figure 3: Left Base Image With Checkerboard Overlay Disturbance. Middle Box-Filter Applied. Right 2D Binomial Filter Applied. Source: [10, chap. 18.4].

In CNNs, the filters inside the convolutional layers are based on trained weights. Trained convolutional layers often contain filters that resemble well-known filter structures from computer vision [10, chap. 18.15]. This is not surprising, as neural networks need various features from the input image to correctly interpret it. Sobel operator-like structures enable the model to extract edges. Therefore, it can be asserted that CNNs have learned mechanisms through their training to suppress noise or extract features such as edges, color differences, etc., for their specific application.
The term *Filter Capacity* can be defined for a CNN. I define filter capacity as the measure of a model's ability to extract features and correctly classify an image. In this project, I start with a consistently non-zero-weighted model architecture and aim to enhance the model's filter capacity through pattern pruning. The model's behavior is examined using disturbed/noisy input images, so-called adversarial samples, to make statements about filter capacity.

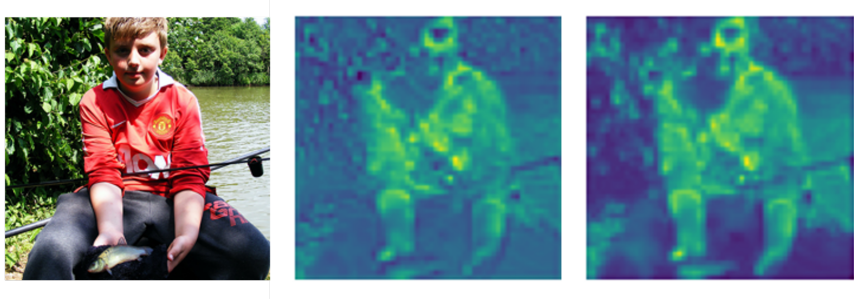


Figure 4: Base Image on the Left, Default Model Feature Map in the Middle and Pattern-Pruned Model Feature Map on the Right

An example of improved filter capacity can be seen in Figure 4. The figure shows the base image, a feature map extracted from a non-optimized model and the same extracted from a pattern-pruned model. At the depicted sample position, the kernel of the optimized convolutional layer extracts more detailed image information than the comparison layer. In the feature map of the pattern-pruned model, sharper edges can be observed. Additionally, the image shows more contrasting color differences, making the feature map more similar to the original image. The feature map of the base model contains softer edges and has a lower variance in color intensity, making it harder to recognize features. We could for example claim, that the optimized model has a greater filter capacity than the base model. However, this claim serves only as an example to illustrate filter

capacity. To make a meaningful statement about filter capacity, more detailed and comprehensive evaluations are needed.

2.2. Adversarial Attacks/ Defenses

In this work, a basic understanding of adversarial attacks as well as defenses is essential. The following section highlights the key fundamentals that are part of this research topic. Adversarial attacks are approaches that attempt to trick deep neural networks by adding minor, often unnoticeable modifications to input data, resulting in inaccurate model predictions. These attacks can be classified based on the attacker's knowledge of the model, timing, targets and attack frequency, while exploiting vulnerabilities in neural networks [2, pp. 2–3]. Adversarial defenses, on the other hand, are strategies implemented to improve the robustness and reliability of deep neural networks against such attacks [2, pp. 3–5].
From the attacker's perspective, adversarial attacks can be broadly categorized into two types: *blackbox* and *whitebox*. In whitebox scenarios, the attacker has comprehensive access to model information such as weights and gradients, while in blackbox scenarios, the attacker has limited or no access to this information [2, pp. 2–3]. Whitebox attacks utilize detailed knowledge of the model to create precise perturbations, whereas blackbox attacks rely solely on the model's outputs to find similar perturbations. Additionally, in a blackbox attack, transfer models can be used to generate samples through whitebox attacks on a structurally similar model and apply them to the blackbox model [2, pp. 5–8]. All test examples for evaluating the models are generated through whitebox attacks. The attack methods fully exploit model information to precisely identify and exploit the model's vulnerabilities. For research purposes, the focus is mostly on whitebox attacks, while in real-world scenarios, referenced as blackbox cases, little is usually known about the model.
In a whitebox scenario, adversarial samples are predominantly generated from the model's gradient information to make precise modifications to the input data. These modifications are based on calculating gradients with respect to the input data to determine the direction in which the input needs to be altered to achieve the desired false classification [2, pp. 5–7]. Mathematically, the attacker attempts to maximize the loss function of the model inference while simultaneously limiting the added perturbations. The goal of this limitation is to generate images that show little to no difference to humans, as well as neural networks. Equation (1) illustrates the mathematical principle behind adversarial attacks explained before [2, pp. 2–3].

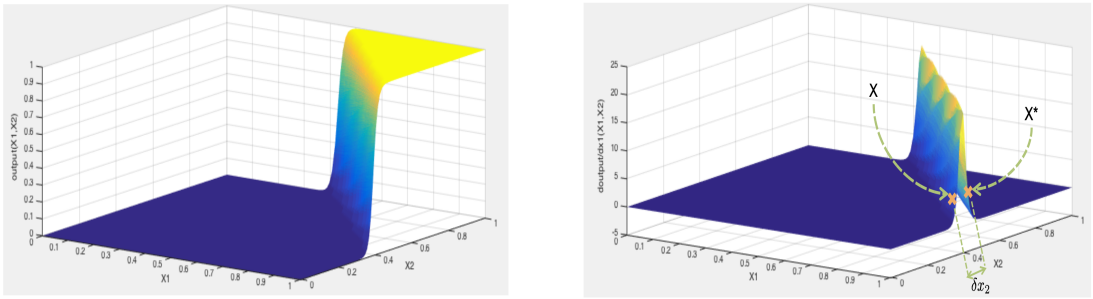
$$\max_{\delta \leq \Delta} \mathcal{L}(\theta, x + \delta, y) \tag{1}$$

such that:

$$f(\theta, x + \delta) = y' \quad \text{and} \quad y' \neq y$$

$\mathcal{L}$ describes the loss function, θ represents the model parameters, $x + \delta$ is the base input image combined with a perturbation δ and y corresponds to the ground truth label. To maximize the loss function through a modified input image, a limiting constraint Δ is provided. This serves as a threshold for the maximum disturbance with which the input data can be altered. Without this constraint, images could be randomly transformed, which would be equivalent to simply swapping images.

Decision boundaries are imaginary boundaries in a model's input space that separate regions where a model distinguish it's class predictions. These boundaries are defined by the model's internal parameters and represent the thresholds at which the model switches from one class to another [11, pp. 2–3]. Gradient information is directly related to the decision boundaries, as it indicates how the input data needs to be altered to cross these boundaries. Calculating the gradients enables to determine the direction and extent of input adjustments required to shift the model's prediction to a different class, thereby forming the basis for generating adversarial samples. [2] [11]. In Figure 5a, the output values of a simple *MLP (Multi Layer Perceptron)* model are presented as a graph. The model takes 2 inputs with values between [0,1] and predicts an output in the binary range $0, 1$, allowing it to classify between two classes. It can easily be observed that only very small changes to the input values are needed to change the output value. The area where the output changes it's state is referred to as the decision boundary [12, p. 4]. Figure 5b displays the MLP's gradient information. A steep spike around the decision boundary area is visible. Adversarial attacks can use that information to precisely target the weaknesses of models. In Figure 5b, two points x and x^* are marked, and the distance/perturbation between them is drawn as δ. Adversarial attacks recognize that a small perturbation $\delta = |x - x^*| \ll x$ to input neuron 2 at axis x_2 results in a significant change in classification [12, pp. 4–5].



(a) Output. Source: [12]

(b) Decision Boundary. Source: [12]

Figure 5: Output and Gradient of the MLP Model

For more complex models, a simple representation of decision boundaries is difficult. The dimensionality of decision boundaries scales to $n - 1$ dimensions for n input neurons. As a metric for measuring the perturbation intensity on the inputs, l_p-norms are used. Commonly used l_p-norms in the field of adversarial attacks are l_0-, l_2- and l_∞-norms [2, p. 3]. Equation (2) provides the general definition of the l_p-norm [13, p.52].

$$\|\mathbf{x}\|_p = \left(\sum_{i=1}^{n} |x_i|^p \right)^{\frac{1}{p}}, \quad \text{for } p > 1 \tag{2}$$

Norm Type	Description	Equation	Ref.
l_0-norm	Represents a special case as it does not fulfill all mathematical criteria of a norm and is therefore referred to as a "pseudo-norm" [14, p. 2] [15, p. 2–4]. The l_0-norm counts the number of non-zero values.	$\|\mathbf{x}\|_0 = \sum_{i=1}^{n} \mathbb{I}(x_i \neq 0)$	(3)
l_2-norm	Also called Euclidean norm, this value is used to measure the square root of the summed squared elements/pixels. In case of this thesis, the square root of the summed differences between the adversarial- and the base sample. The l_2-norm is characterized by its weighting of outliers more heavily due to the squaring [13, p. 52].	$\|\mathbf{x}\|_2 = \left(\sum_{i=1}^{n} \|x_i\|^2\right)^{1/2}$	(4)
l_∞-norm	The l_∞-norm measures the elementwise/ pixelwise single maximum value. [13, p. 52].	$\|\mathbf{x}\|_\infty = \max_{1 \leq i \leq n} \|x_i\|$	(5)

Table 1: Descriptions and Equations for Different l_p-Norms

The listed norms are important specifications for the generation and defenses of or against adversarial attacks. To scientifically show the success rate of the created adversarial dataset under a certain l_p-norm for a single-pixel attack, the right norm needs to be set. In the case of adversarial defenses, models are defined as robust under l_p-norm constraints [16]. Figure 6 shows adversarial samples for increasing l_2-norm.

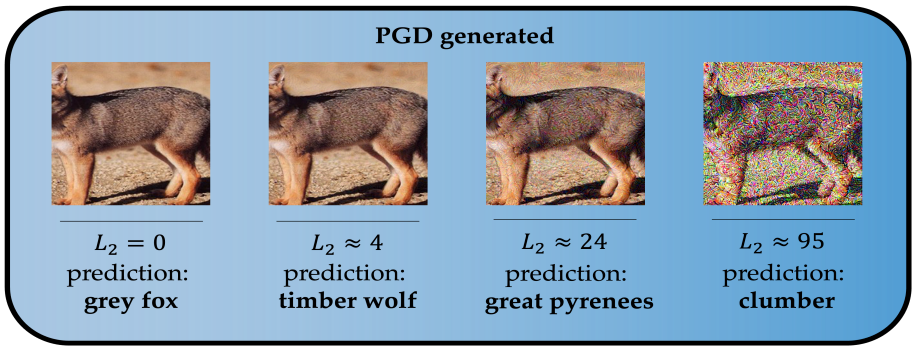


Figure 6: Adversarial Samples With Different l_2-Norm Limitations

Adversarial defenses can be divided into various categories. Figure 7 illustrates differenet defense strategies and their subcategories [2, p. 1] [16, pp. 2–3].
Defenses based on *gradient masking/obfuscation* exhibit gradients that are useless to attackers. Various techniques can be used to smooth the loss function, making it numerically unstable or non-differentiable. The network or the inputs can also be randomly transformed to achieve this de-

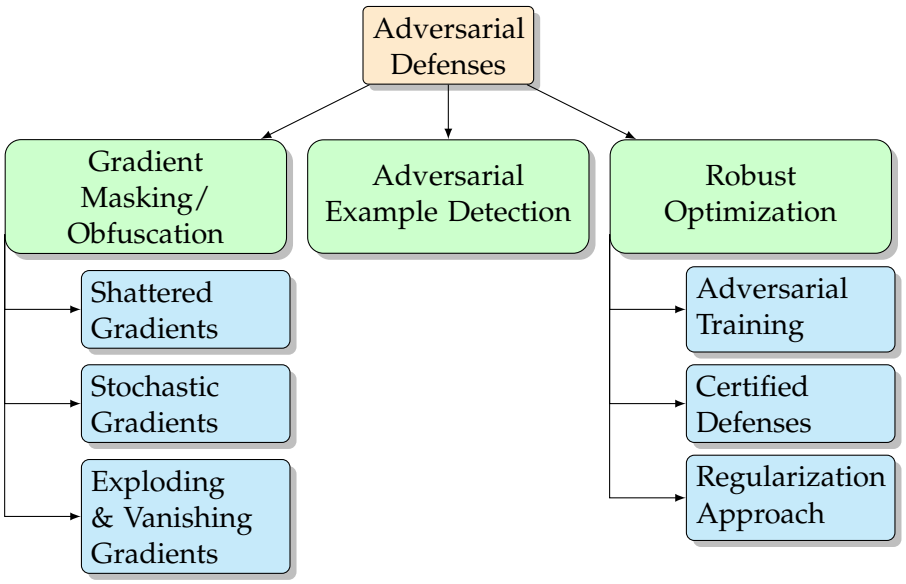


Figure 7: Hierarchical Illustration of Various Adversarial Defense Strategies

fense type. Additionally, recurrent loops within the network can cause gradients to vanish or explode [16, pp. 2–3]. *Adversarial example detection* is a mechanism to detect adversarial attacks by identifying statistical outliers [2, p. 3]. *Robust optimization* aims to adjust the model parameters during the training process to make them more robust against adversarial attacks. This strategy is the focus of this thesis. Robust optimization can be mathematically defined by the min-max optimization problem [2, p. 4]:

$$\min_{\theta} \max_{\delta \in \Delta} \mathcal{L}(f(x+\delta), y) \tag{6}$$

Similar to Eq. (1), the attacker attempts to maximize the loss function $\mathcal{L}$ by adding a perturbation δ. The perturbation is within the boundary Δ. The basic formula for robust optimization extends this equation by minimizing the maximized loss function by optimizing the model parameters θ. To establish a connection to the project, I assume pattern pruning optimization is a *Regularization Approach*, a special case of robust optimization. Pattern pruning is integrated into the training process via the ADMM method and functions as a regularization term. According to Silva et al., the regularization approach is defined by modifying the learning algorithm to improve generalization. Overfitting, especially concerning noise effects, should be avoided and penalized. Appropriate methods should optimize the model so that small changes in the input data do not cause a change in the classification [2, pp. 4–5].

I briefly explain a few of the attacks used for evaluation, including JSMA, DeepFool, PGD and BPDA. *Jacobian-based saliency map attack (JSMA)* identifies the most influential gradient points using a Jacobian matrix to create saliency maps exhibiting these specific points [12]. *DeepFool* uses the decision boundaries to efficiently cross these thresholds iteratively through minimal perturbation steps [11]. *Projected gradient descent (PGD)* iteratively inserts bounded perturbations on the whole input sample toward the model vulnerabilities exploited through gradients [17]. Lastly, the *backward pass differentiable approximation attack (BPDA)* is stated to be a strong attack which can

even approximate non-differentiable gradients in the backward pass. By iteratively improving the approximation it is able to beat gradient masking & obfuscation defenses [16].

As an additional bonus, I apply *Adversarial Training* to the models alongside pattern pruning to explore whether combining these methods can further enhance adversarial robustness. However, this approach is not the primary focus of this thesis. Equation (7) for adversarial training describes the minimization of the loss function over an adversarial dataset and is also a special case of robust optimization [2, p. 4].

$$\min_{\theta} \frac{1}{|D|} \sum_{(x,y)\in D} \max_{\delta\in\Delta} L(f(x+\delta;\theta),y) \tag{7}$$

2.3. Pruning

In this chapter, we will focus on the topic of pruning. It will outline the various types of pruning and the opportunities that arise from their application. Pruning in the context of machine learning describes the technique of reducing and simplifying neural networks. Pruning can affect the connections within a DNN or the weights inside [18, p. 131]. Pruning can be applied to achieve various goals. On one hand, pruning is used to regularize models and prevent overfitting. On the other hand pruning can also increase the accuracy, reduce the memory consumption or decrease the computational effort and inference time of models. Pruning can be applied in various ways, including by integrating it into the training process, which ensures that weights or connections naturally converge to zero, preventing any harm to the models during the final pruning phase. The model training can also be divided into training phases and pruning phases, which undergo retraining after pruning to stabilize the model again. Independent of the training process, pruning can also be applied in a single step, either layer-wise or globally [18, p. 132]. Depending on the application, an appropriate approach can be selected. In general, pruning can be divided into two categories, structured- and unstructured pruning [18, pp. 131–134].

2.3.1. Unstructured Pruning

Convolution Layer:

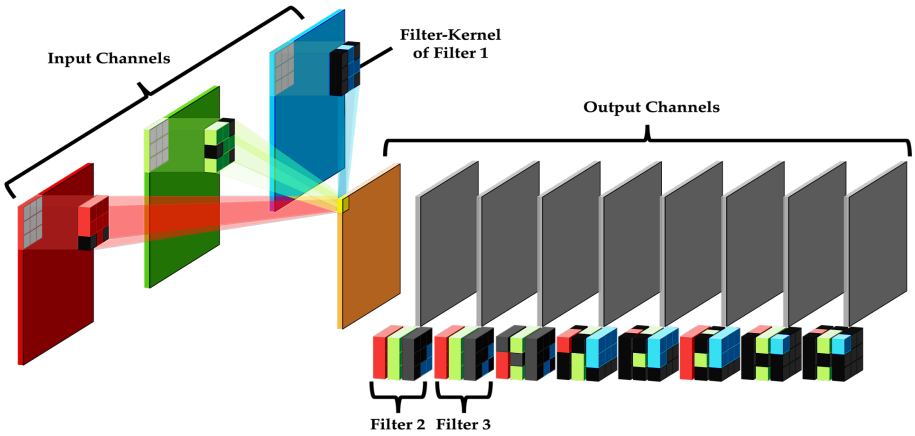


Figure 8: Illustration of Unstructured Pruning on a Convolutional Layer

Fig. 8 shows a pruned convolutional layer. Black-marked fields represent weights that have been removed or set to zero within the convolutional layer's filters. Unstructured pruning does not exhibit any patterns in the way weights are removed. It is completely unpredictable which weights will remain after the pruning process. Because of this uncertainty, it is extremely difficult to exploit the resulting redundancies for hardware acceleration.

Fully Connected Layer:

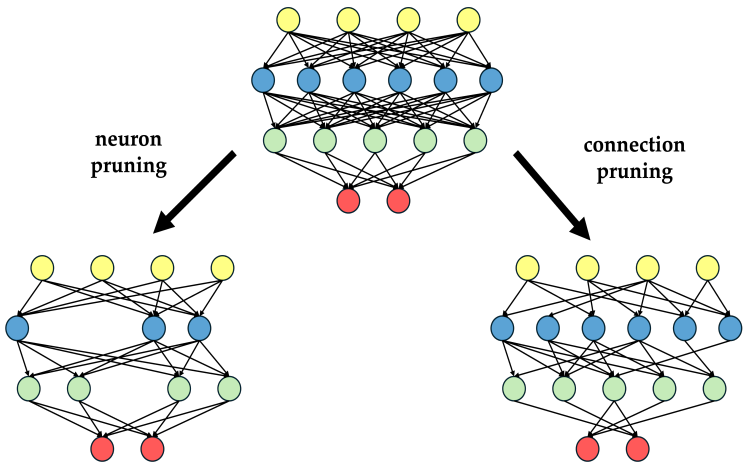


Figure 9: Illustration of Unstructured-Pruned -Neurons and -Connections Inside a Fully Connected Layer

In the case of fully connected layers, two different types of unstructured pruning can be applied. Neuron pruning selectively removes neurons and, consequently, all input and output connections toward the removed neuron. This approach represents a coarser change in the computational graph of the model compared to connection pruning. Connection pruning removes finer con-

nections/weights. The properties of a model are more significantly affected by the removal of a neuron than by a connection.

2.3.2. Structured Pruning

The following examples of structured pruning are exclusively related to convolutional layers and CNNs. Fully connected layers have been excluded on purpose, as their computational effort is relatively small compared to the convolution operations inside the convolutional layers. Efficiency and speed are the driving forces behind pruning in a structured manner. For this reason, structured pruning related to convolutional layers is the primary focus of this chapter.

Channel Pruning:

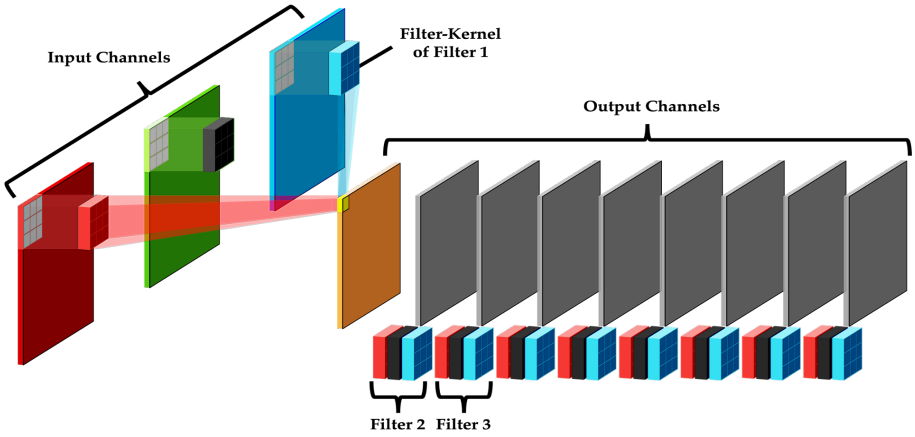


Figure 10: Illustration of Channel Pruning on a Convolutional Layer

Channel pruning removes entire channels from the convolutional layer's computation. In Fig. 10, three input channels are shown. These correspond to a standard JPEG image format with channels for the colors red, green and blue. Channel pruning removes the convolution operation over one of these input channels. The figure clearly shows that the green color channel is not part of the output channel/feature map computation. Only the red and blue channels are included in the calculations.

Filter Pruning:

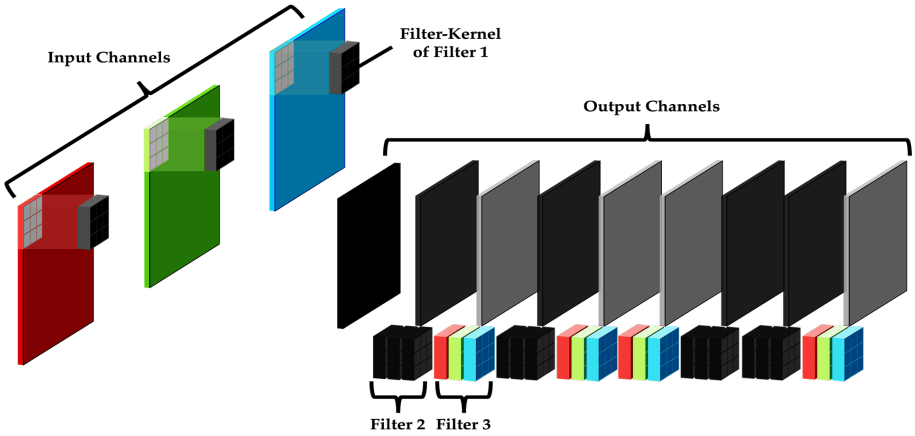


Figure 11: Illustration of Filter Pruning on a Convolutional Layer

Fig. 11 shows the pruning of all filter kernels within specific filters. As a result, individual output channels become unusable or are removed. A convolution through these pruned filters won't identify any features and is therefore redundant.

Filter Kernel Pruning:

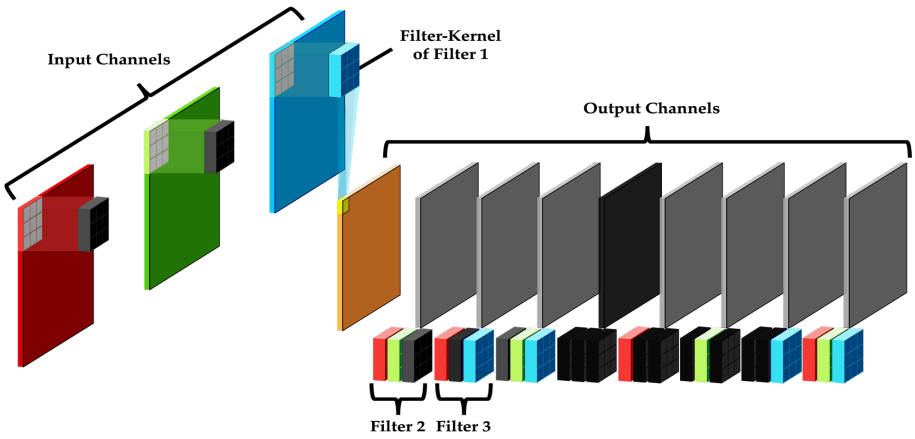


Figure 12: Illustration of Filter Kernel Pruning on a Convolutional Layer

Unlike channel or filter pruning, filter kernel pruning does not select a specific input or output channel to be removed. Filter kernel pruning is used to remove individual connections between input and output channels. The pruning structures across the channels can exhibit irregularities by removing unpredictable amounts of filter kernel over each filter. However, this method also exhibits regular structures by exclusively removing entire filter kernels. This pruning type is also known as *connectivity pruning* [5].

Layer Pruning:

Layer pruning removes weights at the architectural level. Individual layers within a model are identified and removed. In the context of this work, a deeper understanding of this concept is not required [18, p. 133].

Block Pruning:

Block pruning, like layer pruning, removes weights at the architectural level. Unlike layer pruning, block pruning removes entire blocks of multiple layers from a model instead of individual layers [18, p. 133].

Structured Sparsity:

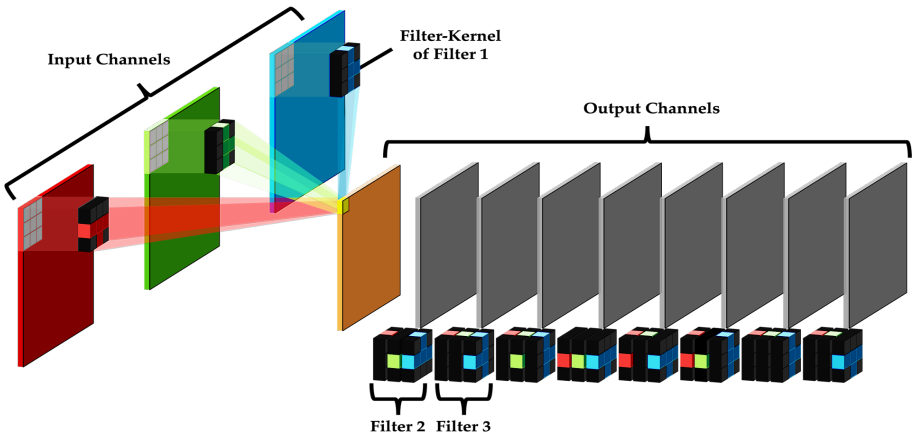


Figure 13: Illustration of Structured Sparsity Pruning on a Convolutional Layer as Pattern-Based Pruning

The regular structure of sparsity in specific weight matrices removed within the convolutional layer classifies structured sparsity as a structured pruning variant. In case of pattern pruning, the fixed number of weights to be removed limits the pattern variations [18, p. 133] [19] [5] [20]. Pattern pruning focuses on the individual filter kernels inside the convolutional layer. The filter kernels are pruned through specific patterns with fixed positioned zero and non-zero weights. Therefore specific structures are enforced within weight matrices. In Figure 13 you can see only 4 different patterns of non-zero weights applied through pruning on all filters. To accelerate a structured-sparsity-pruned model you can make use of the limited amount of patterns and restructure, group, or rather, reformat the weights [19, pp. 914–917].

2.3.3. Pruning Type Comparison

The following Table 2 summarizes the advantages and disadvantages of structured and unstructured pruning. It is hard not see a certain relationship between the model properties depending on the chosen pruning category.

	Unstructured Pruning	Structured Pruning
Description	• removes individual weights or connections • doesn't alter architecture	• removes whole structures • can alter the architecture
Advantages	• accuracy can be increased or maintained • lower accuracy drop while high compression • offers flexiblity in model compression, caused by fine grained weight removal	• possibility for hardware acceleration • reduces model size
Disadvantages	• no to little hardware acceleration possible	• higher decrease in accuracy for increasing compression

Table 2: Comparison of Unstructured and Structured Pruning. Source [18, pp. 131–134]

As listed in Table 2, there is a tradeoff between the properties precision efficiency, memory efficiency and computation efficiency for each pruning type. A magic triangle in Figure 14 illustrates the tradeoff between the pruning techniques. The triangle expresses that it is possible to deliberately try to enhance one or two of these properties by approaching towards the edges or corners, but in return deteriorate another property. This illustration is just for demonstration purposes and doesn't fit the real impact of the pruning effects.

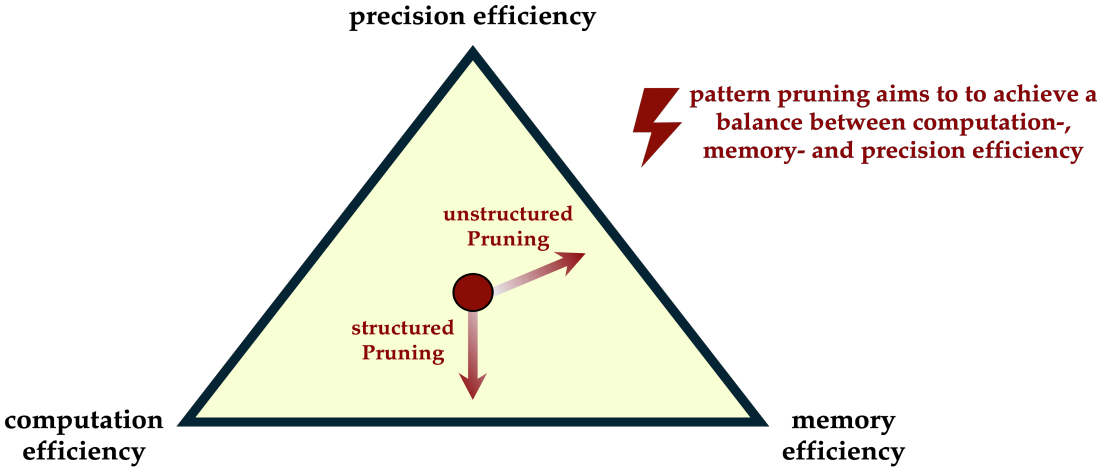


Figure 14: Tradeoff Illustration Regarding Computation Efficiency, Memory Efficiency and Precision Efficiency

Pattern pruning promises an effective compromise between precision- and memory efficiency,

which can only be fully utilized if the model is hardware-optimized afterwards [20, p. 632]. Researchers often combine pattern pruning with filter kernel pruning to achieve higher compression efficiency and computation efficiency while simultaneously increasing or maintaining model precision [19, p. 912] [5, p. 2]. Since hardware optimization is not the focus of this thesis, it is only mentioned as a side note that the full potential of the pattern pruning method can only be realized through further optimization. Within this chapter all important aspects of a foundational understanding of pattern pruning are explained. A more detailed description follows in Chapter 3.

2.4. ADMM - Alternating Direction Method of Multipliers

The *Alternating Direction Method of Multipliers (ADMM)* is an advanced optimization technique that integrates elements of *dual ascent*, *dual decomposition*, *the method of multipliers* and *the augmented Lagrangian* to provide efficient solutions for complex problems [21]. It is particularly effective in solving large-scale optimization problems that can be decomposed into smaller, manageable subproblems, which are solved iteratively. By utilizing dual decomposition, ADMM optimizes each subproblem independently, facilitating parallel and distributed computations [21, pp. 3–5]. The method is capable of handling various problems, including convex and some non-convex problems, although approximate optimal convergence in the case of a non-convex optimization problem cannot be guaranteed [21, pp. 73, 74]. ADMM is considered a versatile tool applied in areas of machine learning, statistics and artificial intelligence [21, p. 2].
To demonstrate the procedure, the method is illustrated using an example of magnitude pruning. A non-convex set is defined as a constraint [22, pp. 5–6].

$$S_i = \{\{\mathbf{W}_i\} \mid \text{card}(\{\mathbf{W}_i\}) \leq l_i\}, \quad i = 1, \dots N \tag{8}$$

In Eq. (8), l_i represents the desired number of non-zero parameters for layer i within the model. $\mathbf{W}_i$ stands for the weight parameters per Layer. The weight pruning optimization problem can be formulated as follows [22, p. 5]:

$$\begin{aligned} &\text{minimize} \quad f(\{\mathbf{W}_i\}) \\ &\text{subject to} \quad \{\mathbf{W}_i\} \in S_i, \quad i = 1, \dots, N \end{aligned} \tag{9}$$

Since it is difficult to solve a function with non-convex constraints, the problem is rewritten and the constraint is isolated into the function g according to Equation (10) defined by Zhang et al [22].

$$\min_{\{\mathbf{W}_i\}} f(\{\mathbf{W}_i\}) + \sum_{i=1}^{N} g_i(\mathbf{W}_i) \tag{10}$$

f in this case, stands for the differentiable and convex loss function of the neural network. g describes a non-differentiable, non-convex discrete combinatorial function that defines the pruning constraints. $g()$ can be defined as an indicator function of Set S_i [22, pp. 1–6]:

$$g_i(\{\mathbf{W}_i\}) = \begin{cases} 0 & \text{if card}(\{\mathbf{W}_i\}) \leq l_i \\ +\infty & \text{otherwise} \end{cases} \tag{11}$$

In equation (10), $\mathbf{W}_i$ can be separated, allowing the general problem to be decomposed into sub-

problems that can be solved isolated to get the solution for the larger optimization problem. We apply dual decomposition from the dual ascent method along with the method of multipliers to divide the general problem into subproblems [21, pp. 9–15] and introduce the auxiliary variable $\mathbf{Z}$. Equation (10) can thus be formulated as:

$$\begin{aligned} &\text{minimize} \quad f(\{\mathbf{W}_i\}) + \sum_{i=1}^{N} g_i(\mathbf{Z}_i) \\ &\text{subject to} \quad \mathbf{W}_i = \mathbf{Z}_i, \quad i = 1, \ldots, N \end{aligned} \tag{12}$$

Now the augmented Lagrangian can be formed from the decomposed optimization Equation (12) [21, p. 13] [22, pp. 6–7].

$$\begin{aligned} L_\rho(\{\mathbf{W}_i\}, \{\mathbf{Z}_i\}, \{\mathbf{\Lambda}_i\}) =& f(\{\mathbf{W}_i\}) + \sum_{i=1}^{N} g_i(\mathbf{Z}_i) \\ &+ \sum_{i=1}^{N} \mathrm{tr}[\mathbf{\Lambda}_i^T(\mathbf{W}_i - \mathbf{Z}_i)] + \sum_{i=1}^{N} \frac{\rho_i}{2} \|\mathbf{W}_i - \mathbf{Z}_i\|_F^2, \end{aligned} \tag{13}$$

Equation (13) introduces two new variable $\mathbf{\Lambda}_i$ also known as Lagrange multiplier and the penalty term ρ_i. Dual variable $\mathbf{\Lambda}$ can be substituted as the scaled dual variable $\mathbf{U_i} = \frac{1}{p_i}\mathbf{\Lambda}$ [22, pp. 6–7].

$$\begin{aligned} L_\rho(\{\mathbf{W}_i\}, \{\mathbf{Z}_i\}, \{\mathbf{U}_i\}) =& f(\{\mathbf{W}_i\}) + \sum_{i=1}^{N} g_i(\mathbf{Z}_i) \\ &+ \sum_{i=1}^{N} \frac{\rho_i}{2} \|\mathbf{W}_i - \mathbf{Z}_i + \mathbf{U}_i\|_F^2 - \sum_{i=1}^{N} \frac{\rho_i}{2} \|\mathbf{U}_i\|_F^2. \end{aligned} \tag{14}$$

The augmented Lagrangian can be divided into primal-proximal problems for iterative solving using ADMM, where the specific variables $\mathbf{W}$, $\mathbf{Z}$, $\mathbf{U}$ used for one of these subproblems to get the updated variable are fixed. The subproblems are solved sequentially, with variables being non-changeable during each step [21, pp. 76–77]. $\mathbf{W}_i$ and $\mathbf{Z}_i$ are almost symmetric, as the update of the dual variable $\mathbf{U}_i$ occurs after $\mathbf{Z}$-update but prior to $\mathbf{W}$-update step [21, p. 14]. To perform ADMM, various steps need to be derived from Eq. (14), which are applied sequentially in each $k = 0, 1, \ldots, n$ iteration [22]:

1. Step Solve Primal Subproblem

$$L_{\text{primal}}(W_i^k, Z_i^k, U_i^k) = f(W_i^k) + \frac{\rho_i}{2} \|W_i^k - Z_i^k + U_i^k\|_F^2 \tag{15}$$

$$W_i^{k+1} = \arg\min_{W_i^k} L_{\text{primal}}(W_i^k, Z_i^k, U_i^k) \tag{primal}$$

Where $f(W_i^k)$ can be solved using stochastic gradient descent.

2. Step Solve Proximal Subproblem

$$L_{\text{proximal}}(\{Z_i^k\}) = \sum_{i=1}^{N} g_i(Z_i^k) + \sum_{i=1}^{N} \frac{\rho_i}{2} \|W_i^{k+1} - Z_i^k + U_i^k\|_F^2 \tag{16}$$

This can be simplified under the condition $W_i = Z_i$ with definition (11) and Z_i meeting the cardinality condition in Eq. (8) to:

$$L_{\text{proximal}}(\{Z_i^k\}) = \sum_{i=1}^{N} \frac{\rho_i}{2} \|W_i^{k+1} - Z_i^k + U_i^k\|_F^2 \tag{17}$$

To get the proximal solution, we can define the proximal solution, which can be solved through the euclidean projection $\Pi_{S_i}(\cdot)$ onto the set S_i:

$$Z_i^{k+1} = \arg\min_{Z_i^k} L_{\text{proximal}}(\{W_i^{k+1}\}, \{Z_i^k\}, \{U_i^k\}) \tag{18}$$

$$Z_i^{k+1} = \Pi_{S_i}(W_i^{k+1} + U_i^k) \qquad \text{(proximal)}$$

3. Step Update Dual Variable

$$U_i^{k+1} := U_i^k + W_i^{k+1} - Z_i^{k+1} \qquad \text{(dual update)}$$

Iterating over primal-, proximal-, and dual update- steps in each iteration leads the model to get an optimal solution while incorperating complex pruning techniques. The solution will be a convergence in the direction of an optimal solution. The algorithm is finished by the given iteration amount or by the given threshold condition ϵ.

$$\|\mathbf{W}_i^{k+1} - \mathbf{Z}_i^{k+1}\|_F^2 \leq \epsilon_i, \quad \|\mathbf{Z}_i^{k+1} - \mathbf{Z}_i^k\|_F^2 \leq \epsilon_i \qquad \text{(termination criterion)}$$

The first termination criterion can be expressed as a low change between the original weights of the model and the projected pruned weights which function as targeted states while ADMM optimization. The second termination criterion is an expression of the difference between the prior and updated projected pruned weight states. Both conditions are checking if an optimal solution is found through measuring if still big changes happen within the iterations. Previous explanations are based on unstructured magnitude pruning. To adapt this mechanism to pattern pruning, we just need to define a different constraint set S_i, that is used for updating auxiliary variable $\mathbf{Z}$ similar to Eq. (proximal). Niu et al. defined the constraint for pattern pruning as follows [19, p. 913]:

$$S_i^{\text{pattern}} := \{\mathbf{W}_i \mid \text{each kernel in } \mathbf{W} \text{ needs to satisfy one specific pattern shape in the pattern set (and non-zero weight values can be arbitrary)}\} \tag{19}$$

3. Pattern-based Pruning Implementation

In this chapter, the practical implementation of the methods and concepts applied in the research for this thesis will be explained. As part of sharing knowledge, the problems I faced and gained knowledge from will also be discussed, with a special focus on the key points that were crucial for achieving good results. The chapter starts with an overview of the application of the ADMM algorithm for pattern pruning, followed by an examination of its convergence and detailed algorithms used. Subsequently, the pattern library will be explained, which forms the foundation of pattern pruning. Various approaches, as well as their backgrounds, are discussed. A special focus is placed on the explanation of *Sparse Convolutional Patterns (SCPs)*, which make use of knowledge from the field of computer vision. In the closing chapter, I introduce the method for a numerical evaluation of convolutional filter performance. This evaluation method is called *Channel-wise Increase of Confidence*, abbreviated *CIC*, and originates from a visualization technique known as *Score-CAM*.

3.1. ADMM Implementation

3.1.1. Convergence

In this chapter, we will dive into the behavior and implementation of the ADMM algorithm. In the previous chapter 2.4, the theoretical foundations and steps of the optimization algorithm were derived. Therefore, we can now directly address the behavior and adaptation of the algorithm. ADMM utilizes a regularization term in the training process to steer the weights toward the seemingly optimal pruning structures. During test runs, it became clear how important it is to balance the ADMM parameters with common hyperparameters used in training, such as gradient clipping, normalization, and so on. One of these ADMM parameters is the *ADMM-Iteration*. ADMM-Iterations do not correspond to the usual batch iterations while training. This type of iteration specifies how many batch iterations must be completed before the ADMM variables are updated and a new approximation to the solution is calculated. In my experiments, I observed that smaller values for ADMM-Iterations cause the model to converge faster toward the epsilon-threshold stopping criterion, defined in equation (termination criterion). Smaller ADMM-Iterations pose a problem as the model can only process a small fraction of the dataset before updating the ADMM variables. These quick update cycles lead to rapid convergence but also cause the algorithm to approach toward a minimum early. There is a risk that a local minimum will be instantly identified as the optimal solution and as the approach progresses, the model's gradients regularized by ADMM become too small to enforce a fundamental change in their pruning structure or in simple words a direction change. Fast convergence can cause the algorithm to get stuck in suboptimal solutions.

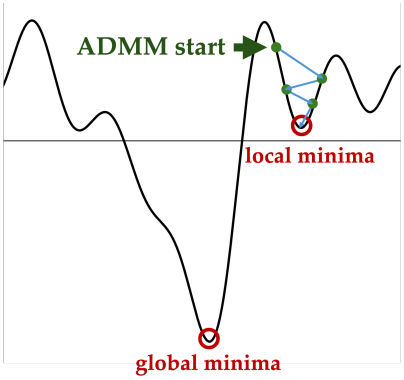


Figure 15: Simplified Illustration of the Risk of Convergence to Local Minima

Figure 15 illustrates this case by using a simple non-convex graph. This aims to demonstrate that shorter update cycles may converge faster but may not lead to an optimal result. ADMM merely aims for the next best improvement available. If the gradients generated during the batch iterations defined by the ADMM-Iterations are not strong enough to escape the minimum the algorithm will stay stuck. As mentioned in Chapter 2.4, ADMM does not necessarily find an optimal solution for non-convex optimization problems. A typical LeNet model comprises approximately 60,000 trainable parameters. Considering the variety of pruning combinations, which can be calculated using the binomial coefficient for a global pruning rate of 50%, the number of possible combinations is in the order of $\approx 10^{18059}$. This high variability suggests a large number of local minima. Therefore, I used longer ADMM cycles, allowing enough input images to be processed, reducing the probability of getting stuck at a suboptimal solution. The number of processed training samples is expanded and the weights can better stabilize and if necessary, resist the ADMM regularization. In addition to the ADMM cycles, tuning parameters such as gradient clipping, learning rate, penalty term, normalization parameters and batch size are also crucial for successful results. If gradient clipping is set too high, ADMM loses its influence on the model, if it is too low, ADMM oscillates at a given point. If the learning rate is set too high, the graph oscillates even more and forms a plateau. A high learning rate causes strong changes in the weight parameters, causing the optimization to make very slow or no progress beyond a certain point. Extremely important is the penalty term ρ seen in formulas (15) and (17), which determines the regularization impact. This term needs to be adjusted for the gradients. Since the ADMM algorithm is applied as a regularization term to the loss function, it results in a superposition of gradient vectors. Equation (14) demonstrates that ADMM regularization is added to the gradients propagated by the loss function. If the ADMM term is too small due to a low ρ, ADMM won't have enough influence to reverse the gradient direction when necessary. To put it simply, the loss function calculates gradients for all weights to guide the training optimization to achieve better classification. For example, a weight parameter is supposed to increase according to its gradient. However, according to ADMM, it should be pruned. ADMM aims to influence the gradient so that it decreases toward zero instead of increasing. If the penalty term is low, the gradient computed from the loss function is weakened but not reversed. On the other hand, a higher value would be counterproductive. The high influence of ADMM would continuously disrupt the gradients during training, resulting in a decrease in accuracy. For these reasons, it is essential to maintain a balance between the hyperparameters. ADMM tries to provide an optimization direction for the model. The model must still be able to counteract if the target state of ADMM cannot align with the gradients. In this case, a new regularization direction can be calculated for the next iteration by generating new combinations of masks which means selecting new weights for pruning. During the subsequent training, it will be determined whether the new mask is more suitable.

Figure 16 shows both convergence graphs for determining an optimal pruning mask. The graph exhibits a sideways movement with significant jitter. Both convergence graphs do not converge toward each other, which is another indicator of a failed optimization. The iterations on the x-axis represent the ADMM-Iterations excluding the warm-up iterations. It is clearly visible that the regularization does not converge during training. Reasons for this include an excessively large ADMM cycle, insufficient gradient preprocessing and a low penalty value. The ADMM regularization is applied over many input samples due to the large ADMM cycle. The insufficient gradient limitation in combination with a large batch size, overshadows the regularization influence. It has been observed that no approximation to an optimal solution can be achieved.

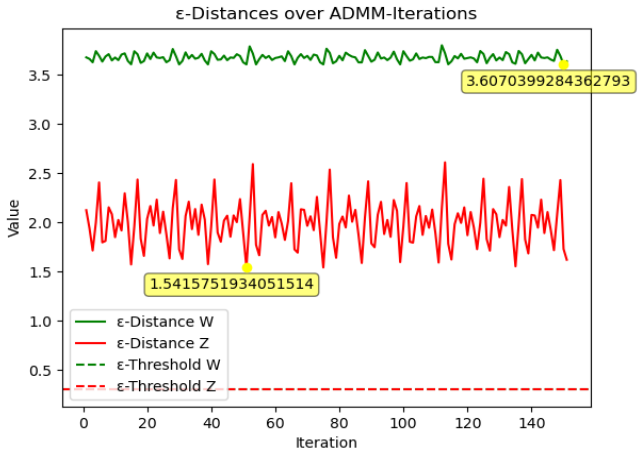


Figure 16: Example of Poor ADMM convergence on threshold conditions in Eq. (termination criterion)

Figure 17 on the other side shows more optimal convergence graphs. Smaller outliers can be identified, which nevertheless steadily move toward a minimum. The presented graphs exhibit a healthy progression. The optimization does not converge too early, enabling a more balanced regularization. We can clearly see that the distance between both graphs decreases which indicates the meeting of the constraints incorporated through Eq. (12). Each model requires its own customized parameterization. For example, smaller models with fewer parameters require smaller penalty values than larger models. As the number of parameters increases, the influence of the ADMM regularization decreases if ρ is kept constant. This can also be compensated through adjustments to other hyperparameters, like a smaller batch size, but in return, downsizes the model's generalization ability learned while training. During development, it became evident that dynamically adjusting the hyperparameters is necessary to achieve good accuracy results. In this project, only the learning rate for big CNNs was dynamically adjusted, which improved the accuracy after optimization by approximately 10%. Ye et al. stated in their work a further improvement by adjusting the penalty value ρ dynamically [23].

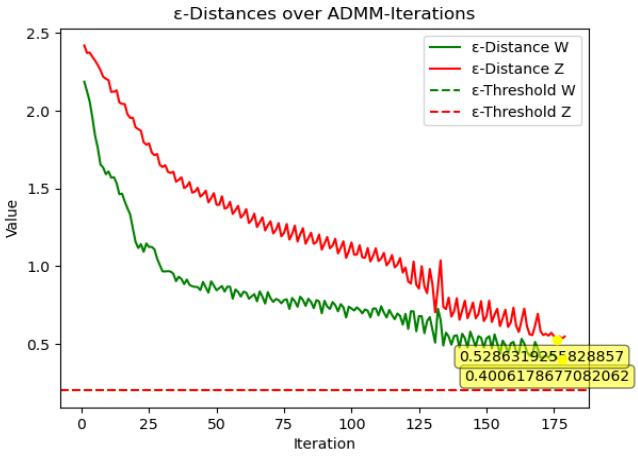


Figure 17: Example of optimal convergence on threshold conditions in Eq. (termination criterion)

In conclusion, it should be noted that the accuracy of the model decreases due to ADMM regularization. This accuracy is restored through retraining. It is important to ensure that the pruned weights are excluded from the retraining process. After backpropagation, the gradients are removed from the pruned weights. Consequently, only the weights that have not been removed by the pruning masks are updated.

3.1.2. Algorithm

As inspiration for the derived implementation of ADMM I referred to previous work by Ren et al. released in their github repository [24]. Algorithms 1 & 2 demonstrate how ADMM is implemented in model training. Algorithm 1 shows the pseudocode for ADMM regularization. To generate the masks, the pruning mechanism will be used. The only constraint to this is, that the masks exhibit the same dimensions as the weight matrices and contain only 0 and 1 values. These binary values indicate which weights should be pruned and which should not. Pruning occurs in Algorithm 1 where the Hadamard product between the auxiliary variable $\mathbf{Z}^k$ (target state) and *Masks* is stored in $\mathbf{Z}^{k+1}$. This step is the final part of the Euclidean projection onto the set S_i in Eq. (proximal) under the constraint in Eq. (19).

Algorithm 1: ADMM Pruning Regularization

Data:
- $\mathbf{W}$: model weights;
- $\nabla\mathbf{W}$: model gradients;
- ρ: ADMM penalty term;

while *model training* || *convergence criterion* $\neq$ *True* **do**
- **if** *ADMM Iteration* **then**
  - project $\mathbf{Z}^{k+1}$: /* solution to Eq. (proximal) */
    - $\mathbf{Z}^k \leftarrow \mathbf{W}^{k+1} + \mathbf{U}^k$;
    - **Masks** $\leftarrow$ pruning technique ; /* desired pruning technique */
    - $\mathbf{Z}^{k+1} \leftarrow \mathbf{Z}^k \odot$ **Masks**;
  - update $\mathbf{U}^{k+1}$: /* solution to Eq. (dual update) */
    - $\mathbf{U}^{k+1} \leftarrow \mathbf{U}^k + \mathbf{W}^{k+1} - \mathbf{Z}^{k+1}$;
- **end**
- update $\mathbf{W}^{k+1}$: /* solution to Eq. (primal) */
  - forward pass: compute model prediction;
  - compute loss using loss function;
  - backward pass: compute gradients $\nabla\mathbf{W}$;
  - regularization: clipping, normalization etc. on gradients $\nabla\mathbf{W}$;
  - ADMM regularization on gradients $\nabla\mathbf{W}$;
    - $\nabla\mathbf{W}^{k+1} \leftarrow \nabla\mathbf{W}^k + \rho \cdot (\mathbf{W}^{k+1} + \mathbf{U}^{k+1} - \mathbf{Z}^{k+1})$;
- update model with $\nabla\mathbf{W}^{k+1}$ using optimizer;

end

Algorithm 2 briefly outlines the process for retraining the model after ADMM regularization. Retraining is crucial for restoring or improving model accuracy. The regularization attempts to identify the best pruning combination during training. This process alters the model parameters and finds an optimal pruning-combination/ -mask, but does not yet achieve optimal accuracy. Steps to achieve optimal accuracy are similar to those in the update step of Algorithm 1 and typical model training. The only difference is that the masks are applied to both, weights and gradients through element-wise matrix multiplication.

Algorithm 2: ADMM Pruning Retraining

Data:
  W: model weights;
  ∇**W**: model gradients;
  Mask: pruning mask;
while *model training* **do**
  forward pass: compute model prediction;
  compute loss using loss function;
  backward pass: compute gradients ∇**W**;
  regularization: clipping, normalization etc. on gradients ∇**W**;
  apply mask:
    W $\leftarrow$ **W** $\odot$ **Masks** ; /* mask is applied on weights */
    ∇**W** $\leftarrow$ ∇**W** $\odot$ **Masks** ; /* mask is applied on gradients */
  update model with ∇**W** using optimizer;
end

3.2. Pattern Library

To implement pattern pruning incooperated by ADMM, a set of patterns needs to be defined. Patterns are stored in a so called pattern library, which is the heart of pattern-based pruning and plays a central role in optimizing the neural network. It enables the selection and storage of efficient pruning patterns that can be tailored to the model's specific needs or externally injected to enforce certain structures in the convolutional layers. A good set of patterns can also contribute to minimizing memory accesses, thereby significantly enhancing hardware-level performance. The pattern library ensures the formation of effective filters that can improve both the image processing properties and the network performance [5, p. 6]. Patterns are 2-dimensional binary 3x3 matrices with the values 0,1. The constraint specifies that each pattern must contain exactly 5 zero values and 4 non-zero values.

3.2.1. Trivial Pattern Library

Based on the research by Ma et al. [20], I decided to use all possible pattern variations as the pattern set of the pattern library. I refer to this as the *trivial pattern library*. Using the binomial coefficient, a total of 126 different patterns can be generated based on the given constraints. Thus, covering all

variations each filter kernel can be assigned to the most suitable pattern. However, the downside is that the large number of patterns doesn't align with the core idea that pattern-based pruning is based on. While a large pattern library achieves optimal pruning results, high variations of positioned weight patterns are difficult to accelerate. Pattern pruning originated from the idea of using a strongly limited number of patterns to lower the variations and enforce similarity within the layers. Filter kernels pruned with just a few different patterns can be rearranged and grouped together. Those new convolution structures can be efficiently adapted to specific hardware to make use of acceleration techniques like parallel vector multiplication *(SIMD-/Single Instruction Multiple Data- Operations)*, which reduce memory accesses and *floating-point operations* (abbreviated *FLOPs*). The grouped filters display identical index accesses, enabling the cache to retain weights and reducing the need for control structures by reusing indices with non-zero values. [25, pp. 1–2] [5, p. 2]. Considering the core idea of hardware optimization, I implemented an own algorithm to reduce the pattern library within the optimization process similar to the authors behind "An Image Enhancing Pattern-Based Sparsity for Real-Time Inference on Mobile Devices" [20].
One of the differences between the approach of the authors Ma et al. and this thesis is that I used their reduction logic as a foundation to develop an extended form of this reduction logic. In their work, they removed patterns from the pattern library based on their occurrences inside the CNN [20, p. 637]. The problem with this approach is that it prioritizes quantity over quality. This means that five weak filter kernels could be assigned greater importance than one strong filter kernel. It is taken into account that weak filter kernels exhibit higher fluctuation in their assignment of patterns during regularization, as small weight values can easier form new pattern structures. The regularization process would prioritize these volatile patterns over strong patterns with larger weight values, which could consistently contribute more to the model prediction. Due to this fact, I extended the reduction logic based on pattern occurrence to include the 'strength' or influence of each pattern. Equations (20) and (21) describe the procedure in which the pattern with the lowest score is removed from the pattern library.

$$r = \frac{|\text{available patterns}|}{|\text{total patterns}|} \tag{20}$$

$$\begin{aligned} P_i &= r \cdot O_i + (1 - r) \cdot I_i \\ &= r \cdot (O_i - I_i) + I_i \end{aligned} \tag{21}$$

- P_i: Pattern score for pattern i
- r: Ratio of available patterns to total patterns
- O_i: Occurrence of pattern i
- I_i: Influence of pattern i

Additionally, Ma et al. do not retrain the model parameters after the ADMM regularization [20, p. 637].
With the presented reduction method, the pattern library can be reduced to a predetermined amount, allowing for an increased number of suboptimal pattern-kernel coverages. The results of this approach will be presented and evaluated in the evaluation chapter of this work. In addition, I utilized another pattern library that produced better results in many applications compared to the *trivial pattern library* with reduction logic. These patterns will be presented in the next chapters.

For illustration, the algorithm for creating the mask based on the *trivial pattern library* can be found under Algorithm 3.

Algorithm 3: Pruning Mask generation on trivial pattern library

Data:
  W: model weights;
  $\mathbf{\Omega}_{pattern}$: pattern library;
Result:
  Mask: pruning mask;
begin
  if *iteration* == *0* **then**
    initialization pattern library $\mathbf{\Omega}_{pattern}$;
  else if *iteration != 0 && len*($\mathbf{\Omega}_{pattern}$) > *min_patterns* **then**
    remove pattern from pattern library $\mathbf{\Omega}_{pattern}$;
  for *filter kernel* $\in$ **W do**
    store best fitting pattern $\in \mathbf{\Omega}_{pattern}$ in **Mask**;
  end
end

3.2.2. ELoG-Filter Approximation & Derivation

The motivation behind the development of *Sparse Convolution Patterns* as a pattern library is based on the need to combine high model accuracy with regular pruning structures. *SCPs* are derived from mathematical theories in the field of computer vision. They attempt to exploit well known aspects of image processing filters, such as the *Gaussian filter's* picture smoothing, which eliminates disturbing high-resolution details [10, chap. 17.3.1] or the effects of the *Laplacian of Gaussian filter* in advanced edge detection and smoothing [26, pp. 181–183] [5, p. 4]. While randomly selected patterns could also contribute to reduced model complexity, they lack of systematic structure and mathematical foundation is crucial for optimizing image processing capabilities [5, p. 3]. As mentioned in the previous section, the number of pattern variations is kept as small as possible to enable the following hardware acceleration. SCPs are interpolations of the *ELoG-Filter* and consist of exactly 4 patterns. The patterns derived from vision theory not only help to improve the efficiency of CNNs but can also maintain and enhance feature extraction, which is not guaranteed with random patterns. The following text aims to clearly list the steps for deriving the SCPs based on the work of Ma et al [5].

Derivation of the Gaussian Filter

For creating the discrete 2D Gaussian filter, starting from the Gaussian function definition:

$$G(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{x^2}{2\sigma^2}} \tag{22}$$

For the derivation I got the best results and the targeted ratio for the Gaussian filter with σ as [27]:

$$\sigma = \sqrt{\frac{1}{2\ln 2}} \tag{23}$$

For a 1D filter of length 3 centered around zero, the spatial points would be $x \in \{-1, 0, 1\}$. Applying (23) we get:

$$G(x) = \frac{1}{\sqrt{2\pi\left(\sqrt{\frac{1}{2\ln 2}}\right)^2}} e^{-\frac{x^2}{2\left(\sqrt{\frac{1}{2\ln 2}}\right)^2}} \tag{24}$$

which is simplified to:

$$G(x) = \sqrt{\pi \ln 2} \cdot e^{-x^2 \ln 2} \tag{25}$$

Inserting the spatial positions we get the unnormalized discrete 1D Gaussian filter:

$$G_x = \left[\frac{\sqrt{\pi \ln 2}}{2}, \sqrt{\pi \ln 2}, \frac{\sqrt{\pi \ln 2}}{2}\right]$$

With a scaling factor k:

$$k = \frac{2}{\sqrt{\pi \ln 2}}$$

We get the scaled 1D Gaussian filter:

$$G_x = k \cdot G_x = [1, 2, 1] \tag{26}$$

The 1D Gaussian filter in y direction can be approximated the same way. As G_y is G_x^T we can convolve both to get the final 2D Gaussian filter:

$$G_{xy} = G_x * G_y = \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix} \tag{27}$$

"$*$" refers to the convolution operation. The approximated 2D Gaussian filter can be used for convolution in image processing, preserving the properties of the Gaussian distribution. When applied to an image, the Gaussian filter smooths out noise and minor details by averaging pixel values resulting in blurred images with filter values following the Gaussian distribution [28].

First Derivation of the Laplacian of Gaussian Filter

To approximate the Laplacian operator ∇^2 for the LoG filter $\nabla^2 G(x, y)$ we can start with the 1D case and then extend it to 2D. To do this I refer to some parts of the information presented by Ma et al [5]. At first, we start with the *taylor expansion* of the 1D Gaussian filter in x direction.
Using (26) we can apply Taylor series expansion on forward and backward passes [20].

$$G(x + \delta) = G(x) + \delta G'(x) + \frac{1}{2}\delta^2 G''(x) + \frac{1}{3!}\delta^3 G'''(x) + O(\delta^4) \tag{28}$$

$$G(x - \delta) = G(x) - \delta G'(x) + \frac{1}{2}\delta^2 G''(x) - \frac{1}{3!}\delta^3 G'''(x) + O(\delta^4) \tag{29}$$

which can be summed and simplified to

$$G(x - \delta) - 2G(x) + G(x + \delta) = \delta^2 G''(x) + O(\delta^4) \tag{30}$$

In the context of applying the *central difference theorem* to discrete filters, a *denominator* δ^2 is needed. It refers to the limit operator of a derivative function, which represents an infinitesimal small change. As we are working with filters, this normalization is implicitly accounted for by the filter coefficients and is often referred to as the interval of discretization [29]. In simple words, it's the step size between spatial directions x, y inside the filter. As shown in the formulas before, we are expanding around the zero position of x and y with the maximum forward or backward step of 1. As δ in $O(\delta^4)$ is the smallest discrete interval approaching the central zero position. The error $O(\delta^4)$ decreases around the point of expansion proportional to δ by a factor of 4 and can therefore be ignored [30, pp. 437–361, 1077, 1079].

$$\frac{G(x-\delta) - 2G(x) + G(x+\delta)}{\delta^2} \approx G''(x) \tag{31}$$

In the discrete domain, for $\delta = 1$, the approximated second derivative of Gaussian filter is:

$$G''(x) = [1, -2, 1] * G(x) \tag{32}$$

The second derivative of $G''(x)$ is also known as the 1D LoG filter in x direction $\nabla^2_{xx} G(x)$ [5]. The second 1D derivative of $G''(y)$ can be approximated the same way. As both operators $\nabla^2_{xx} G(x)$ and $\nabla^2_{yy} G(y)$ are independent in dimensionality, we can combine them by convolution as in the Eq. (27).

$$\nabla^2_{xy} G(x,y) = (\nabla^2_{xx} * G(x)) * (\nabla^2_{yy} * G(y)) \tag{33}$$

According to the Fourier theorem, convolution in the spatial domain corresponds to multiplication in the frequency domain [31]. This property allows us to combine the effects of the Laplacian operators in the x and y directions and can be illustrated as follows [32]:

$$\nabla^2_{xx} * \nabla^2_{yy} = \mathcal{F}^{-1}\left\{\mathcal{F}\left\{\frac{\partial^2}{\partial x^2}\right\} \cdot \mathcal{F}\left\{\frac{\partial^2}{\partial y^2}\right\}\right\}$$

The convolution of two Gaussian functions is another Gaussian function with a variance that is the sum of the variances of the individual Gaussians [5, p. 4]. This property simplifies the equation as I assume $G(x, y)$ as combined convolution. For later adjustments, the matrix is inverted.

$$\nabla^2_{xy} G(x,y) = (-1) \cdot (\nabla^2_{xx} * \nabla^2_{yy}) * G(x,y) \tag{34}$$

$$\nabla^2_{xy} G(x,y) = (-1)\left(\begin{bmatrix} 1 \\ -2 \\ 1 \end{bmatrix} * \begin{bmatrix} 1 & -2 & 1 \end{bmatrix}\right) * G(x,y) = \begin{bmatrix} -1 & 2 & -1 \\ 2 & -4 & 2 \\ -1 & 2 & -1 \end{bmatrix} * G(x,y) \tag{35}$$

Now we have the first approximation of the Laplacian operator in Eq. (36).

$$\nabla^2 = \begin{bmatrix} -1 & 2 & -1 \\ 2 & -4 & 2 \\ -1 & 2 & -1 \end{bmatrix} \tag{36}$$

Second Derivation of the Laplacian of Gaussian Filter

To understand the different approaches taken to approximating the Laplacian operator ∇^2 and

how they are applied, let's clarify the concepts and methods involved. The first approximation is created by the convolution of two spatially independent approximations. The following second approximation is created by *Multivariate finite differences* analogous to partial derivatives in several variables [29].
We create the second derivation of the 2D LoG filter by *finite difference approximation* with similar steps, but with minor changes in formulation and constraint $G_{xy}(x_0, y_0) = G_{xx}(x_0, y_0) + G_{yy}(x_0, y_0)$.

$$G_{xx}(x,y) \approx \frac{G(x+h,y) - 2G(x,y) + G(x-h,y)}{h^2} \tag{37}$$

$$G_{yy}(x,y) \approx \frac{G(x,y+k) - 2G(x,y) + G(x,y-k)}{k^2} \tag{38}$$

Similar to the first approximation of the Laplacian filter (30) h and k denote to the denominator and are equal to the step size. The difference from the first approximation is that they don't represent 1D filters, but instead they represent 2D filters caused by their additional dimensionality.

$$G_{xx}(x,y) = \begin{bmatrix} 0 & 0 & 0 \\ 1 & -2 & 1 \\ 0 & 0 & 0 \end{bmatrix} * G(x,y), \qquad G_{yy}(x,y) = \begin{bmatrix} 0 & 1 & 0 \\ 0 & -2 & 0 \\ 0 & 1 & 0 \end{bmatrix} * G(x,y) \tag{39}$$

To get the approximation we need to sum both approximations under the mathematical law of the *Laplacian Operator* for cartesian coordinates [30, 1003]. This can be seen as a linear combination of both derivations. Seeing this approximation in cartesian coordinates enables the application of differential operators as both are connected through the constraint defined, regarding the shared influence on x_0- and y_0-coordinates. Under given constraints an *system of equations* can be used to derive Equation (40). Substituting the approximations derived in (39) we can calculate the second approximation of the LoG filter.

$$\begin{aligned} \nabla^2_{xy} G(x,y) &= G_{xx}(x,y) + G_{yy}(x,y) \\ &= \left(\begin{bmatrix} 0 & 0 & 0 \\ 1 & -2 & 1 \\ 0 & 0 & 0 \end{bmatrix} + \begin{bmatrix} 0 & 1 & 0 \\ 0 & -2 & 0 \\ 0 & 1 & 0 \end{bmatrix} \right) * G(x,y) \\ &= \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix} * G(x,y) \end{aligned} \tag{40}$$

Therefore the second approximation is defined as:

$$\nabla^2 = \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix} \tag{41}$$

Inspecting both Laplacian filters, you can see that the approximation (36) seems to be a more detailed filter. It exhibits a more radial continuous structure, which is crucial as it enables it to capture a broader range of features. To recap, the first approximation is created from dimensionality-

independent 1D filters, while the second originates from dimensionality-dependent 2D filters. From this point on, we need to take a few more steps to get to the final SCP patterns.

ELoG-Filter Derivation

Before proceeding, we need to define the environment in which we take these derivatives. This environment is called a *Linear System*, which is important in the field of signal theory. A linear system establishes specific rules that must be considered when performing transformations and changes to equations. The most important aspect is that only linear operators can be used, excluding operations like the *Hadamard Product*, as it would cause a violation of the theorems used in linear systems. The Hadamard Product, as an example, requires matrices of equal shape, which causes a mismatch with input samples of different size. Additivity, homogenity and superposition principles are required by linear systems to fulfill. Therefore, the Hadamard Product is not applicable as it does not satisfy all the necessary conditions for systems to be linear [33] [34]. Up to this point, the paper I am referencing and interpreting in this chapter [5], shows numerous inconsistencies regarding the derivation. The claimed convolution of both Laplacian approximations used to create the *ELoG-Filter*, which, upon closer examination, is actually a Hadamard product, leading to a mismatch with the *Convolution Theorem*, as this transformation can only be achieved through *Fourier Transformation* [35]. The operations stated were incorrect and required significant effort to validate and correct. The lack of clarity in their work [5, pp. 4–5] made it challenging to follow their information, but I was able to find a solution within the constraints of the linear system.

Variable Description:

Variable	Description
$I^{'}$	Filtered image (output of the linear system)
I	Image to be processed (input of the linear system)
H	Filter processed on input (transfer function of linear system)
$L_{1/2}$	First/Second Laplacian Operator approximation
$G_{1/2}$	First/Second Gaussian Filter approximation
F	Intermediate Filter incorporating two Differential Operators and a Gaussian Filter
$F^{'}$	ELoG Filter (baseline for SCPs)

Table 3: Variable Description Regarding the ELoG Filter Derivation as Linear System

The following definition outlines the linear system for image filtering:

$$I^{'} = I * H \tag{42}$$

In Eq. (42) we can see the definition of a linear image processing system. $I^{'}$ defines the filtered image, while I denotes to the input image. H is the transfer function where the filters are placed and applied to the inputs. For this use case, the transfer function H can be decoupled to gain insights into its functionality.

$$H = (L_1 * G_1 * L_2 * G_2) \tag{43}$$

The transfer function can be rearranged and substituted according to the theorems of linear systems:

$$H = F * G_2 \tag{44}$$

Focusing on F which is needed as an intermediate state to develop the predicted ELoG filter. F is a linear combination of the first and second approximated Laplacian filters, convolved with on of the Gaussian filters from the previous approximations (35) or (40). F results in:

$$F = (L_1 * L_2) * G_1 \tag{45}$$

$$= \begin{bmatrix} 0 & -2 & 0 \\ -2 & 8 & -2 \\ 0 & -2 & 0 \end{bmatrix}$$

We introduce a new transformed version of the filter $F -> F^{'}$:

$$F^{'} = a \cdot F + b \cdot \mathbf{C}_{\text{identity}} \tag{46}$$

Applying these values:

$$a = -\frac{1}{2}, \qquad b = 12, \qquad \mathbf{C}_{\text{identity}} = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{pmatrix}$$

We are allowed to introduce these variables as long as additivity and homogeneity rules are not violated. These are holding for $b \in \mathbb{R} \setminus \{0\}$, resulting in the so called ELoG-Filter $F^{'}$ [5, p. 4].

$$F^{'} = \begin{pmatrix} 0 & 1 & 0 \\ 1 & 8 & 1 \\ 0 & 1 & 0 \end{pmatrix} \tag{ELoG-Filter}$$

Replacing F with the newly derived filter F' in Eq. (44), the system can be explicitly described as:

$$I^{'} = I * \left(a \cdot (L_1 * L_2) * G_1 + b \cdot \mathbf{C}_{\text{identity}}\right) * G_2 \tag{47}$$

In this chapter, it was demonstrated how the ELoG filter $F^{'}$ is derived from the combination of Laplacian and Gaussian filters. According to the derivation by Ma et al., the Gaussian filter is interpolated in the form of specially generated patterns, similar to those in the following Chapter 3.2.3, but extended by four additional patterns for representing the Gaussian filter [20, p. 633]. Since the Gaussian filter G_1 is already included in the derivation, these additional patterns would result in an overhead in terms of hardware acceleration and are therefore not used.

3.2.3. Sparse Convolution Pattern Interpolation

In the preceding chapter, we extensively derived the ELoG filter, an essential component of the analysis. This chapter focuses on demonstrating how the ELoG filter is interpolated by the SCPs during the forward propagation through a CNN based on the work of Ma et al. [5, p. 4].

$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & 1 & 1 \\ 0 & 0 & 0 \end{bmatrix} \quad \begin{bmatrix} 0 & 1 & 0 \\ 1 & 1 & 0 \\ 0 & 1 & 0 \end{bmatrix} \quad \begin{bmatrix} 0 & 0 & 0 \\ 1 & 1 & 1 \\ 0 & 1 & 0 \end{bmatrix} \quad \begin{bmatrix} 0 & 1 & 0 \\ 0 & 1 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

A convolutional layer can be described as a set of filter kernels. The set of filter kernels is defined as $\mathcal{K}$:

$$\mathcal{K} = \{K_1, K_2, \ldots, K_m\}$$

where K_i stands for the i-th filter kernel and m is the number of filter kernels. For demonstration purposes, I set the interpolation steps over $\mathcal{K}$ to 4 to see symmetric results.

$$\sum_{i=1}^{n} \frac{\sum_{j=1}^{4} K_j}{4} = \begin{bmatrix} 0 & p \cdot n & 0 \\ p \cdot n & n & p \cdot n \\ 0 & p \cdot n & 0 \end{bmatrix} \cdot \frac{1}{n}$$

$$= \begin{bmatrix} 0 & p & 0 \\ p & 1 & p \\ 0 & p & 0 \end{bmatrix}$$

The summation formula above describes the normalized matrix representation of a single convolutional layer substituted by the SCP patterns in the probability domain. Figure 18 illustrates this equation for $n = 1$.

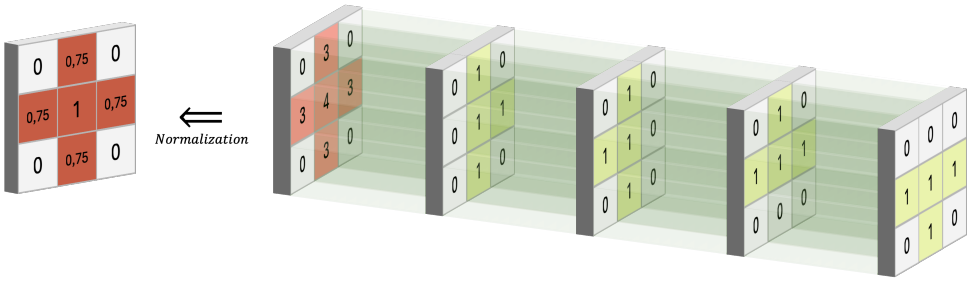


Figure 18: Probability distribution as linear combination of SCPs according to convolutional layer

The interpolated matrix represents the probability distribution indicating the probability of covering the kernel positions with a pattern. Combining all SCPs results in a probability of 75% pattern coverage at edge positions as illustrated in Figure 18.
According to Ma et al., the best approximation of the ELoG-Filter can be achieved through $l = 8$ interpolations and thus of forward propagation through 8 convolutional layers [5, p. 5].

$$\begin{bmatrix} 0 & p & 0 \\ p & 1 & p \\ 0 & p & 0 \end{bmatrix}^{l} = \begin{bmatrix} 0 & 0.1 & 0 \\ 0.1 & 1 & 0.1 \\ 0 & 0.1 & 0 \end{bmatrix} \quad \text{for } l = 8 \text{ and } p = 0.75$$

$$\approx \frac{1}{8} \cdot \begin{bmatrix} 0 & 1 & 0 \\ 1 & 8 & 1 \\ 0 & 1 & 0 \end{bmatrix} \quad \textbf{ELoG-Filter}$$

For default CNN inferences, the outputs of sequential convolutional layers are derived from their convolution operations. By interpreting convolutional layers with probability distributions as characteristic functions within a probability space, the convolution operations on independent random variables can be simplified to Hadamard products of these probability distributions [36] [30, p. 1494].

3.2.4. Comparison of ELoG-, Gaussian- and Laplacian-Filter

In this chapter, I investigate and compare the ELoG-, Gaussian-, and Laplacian filters to examine the properties of the ELoG filter better. The ELoG filter combines the strengths of Gaussian and Laplacian filters. To illustrate the differences, I apply them to noisy input images with a maximum perturbation ratio of r=±20%. This comparison highlights the unique capabilities of each filter. The Gaussian filter is known for its smoothing properties caused by the specific structured positive values, which make it ideal for noise reduction while preserving edges. The Laplacian filter excels in edge detection but is sensitive to noise. I compare the ELoG filter to the Gaussian filter because of their similar positive value descent from the center toward the edges. This helps to understand how ELoG leverages Gaussian's smoothing while incorporating Laplacian's edge extraction. I also slightly modify the ELoG filter for a fair comparison between the Laplacian and ELoG, focusing on their edge detection capabilities. Through this analysis, supported by empirical data and visual examples, I aim to provide clear insights into the operational characteristics and practical applications of these filters in their image processing capabilities.
The ELoG and Gaussian filters are compared in terms of their performance in noise reduction, assessed through the *Signal-to-Noise Ratio (SNR)* and their ability to preserve the image sharpness, evaluated by *Laplacian variance* [37]. Furthermore, the filtered images are visually represented to demonstrate their differences. The comparison between the ELoG and Laplacian filters is based on the visual interpretation of their filtered output images. The derived ELoG filter does not exhibit the characteristics of a differential operator, as it contains no negative values. However, during the pattern pruning process, only masks are defined without controlling whether the weights of the filter kernels will lie in the positive or negative range. Figure 19 shows the structure of all filter kernels after SCP pattern pruning. Figure 20 represents the difference in the number of negative versus positive weight values dependent on the positions aggregated over all convolutional layers. These differences, which I refer to as *polarity*, are mapped on a scale of [0,100] for predominantly positive weight values and [-100,0] for more frequent occurrences of negative values.

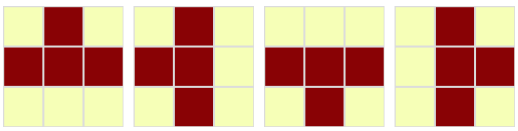
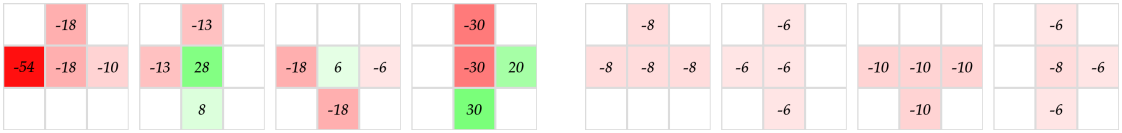


Figure 19: Filter Kernel Structure After SCP Pattern Pruning



(a) LeNet Polarity Distribution (b) ResNet18 Polarity Distribution

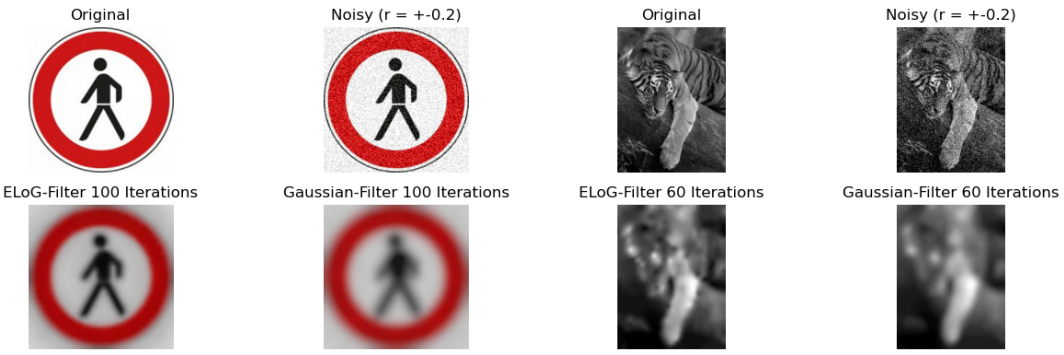
Figure 20: Polarity Distributions After SCP Pattern Pruning

Therefore, for a fair comparison, I compare the Laplacian filter with a slightly modified version of the ELoG filter. This modified ELoG filter is obtained by inverting its central component, resulting in a Laplacian-like filter. The primary difference is that the partially inverted ELoG filter's center has twice the magnitude. As demonstrated by the polarity distributions in Fig. 20, the occurrence of a partially inverted ELoG filter is indeed possible. Additional information about the filter kernel structures and polarity distributions of the models used in evaluation can be found in Appendix A.1.

$$ELoG = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 8 & 1 \\ 0 & 0 & 0 \end{bmatrix} \quad ELoG_{Differential} = \begin{bmatrix} 0 & 1 & 0 \\ 1 & -8 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

$$Gaussian = \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix} \quad Laplacian = \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

Figure 21a visualizes the comparison of image convolutions using the ELoG- and Gaussian filter on a RGB image. The same strategy was applied to the grayscale image in Fig. 21b, while for a better visualization, the filtered images were presented after 60 iterations.



(a) Comparison on RGB-Images (b) Comparison on Grayscale-Images

Figure 21: ELoG- and Gaussian filtered Images

Based on the filtered images, it is evident that the ELoG filter has a reduced smoothing effect, allowing edges to remain more distinct. Both filters effectively remove noise to the same extent. Observations regarding SNR and σ_{log} sharpness in Figures 22 & 23 propose a lower noise filtering ability of ELoG in comparison to Gaussian. In sharpness comparison the ELoG filter outperforms Gaussian with an overall higher edge retention. Decibel conversion was used in the sharpness calculation because it provides a clearer representation for this use case.

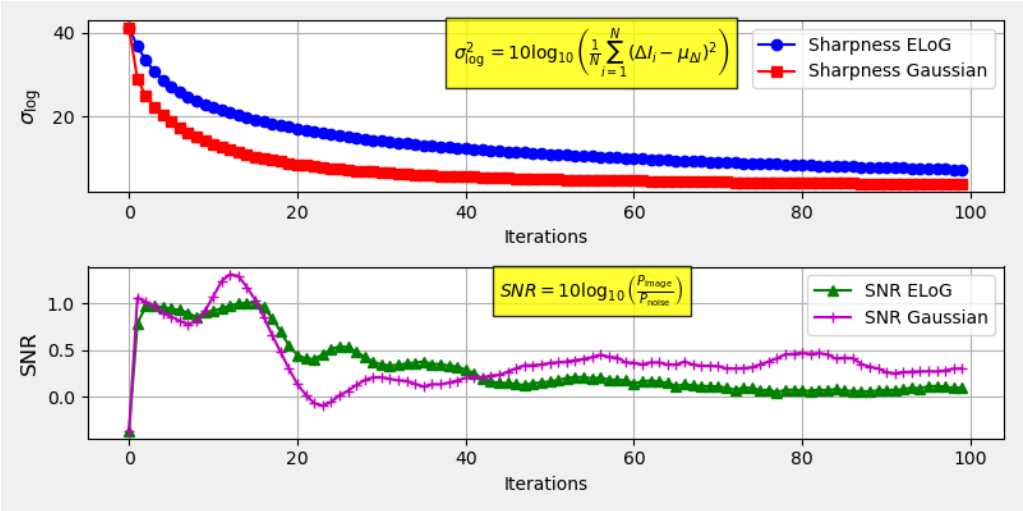


Figure 22: SNR and Sharpness Measurement on RGB-Image 21a

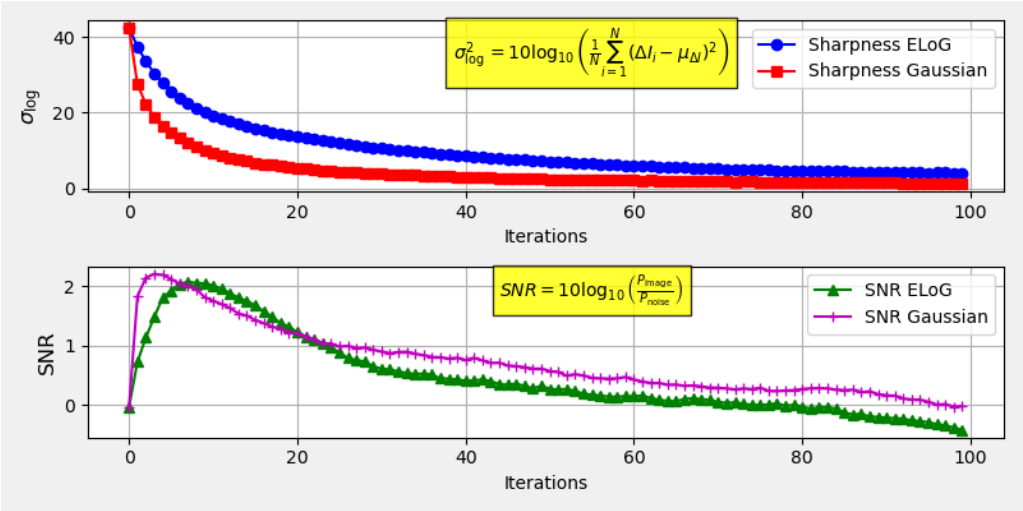
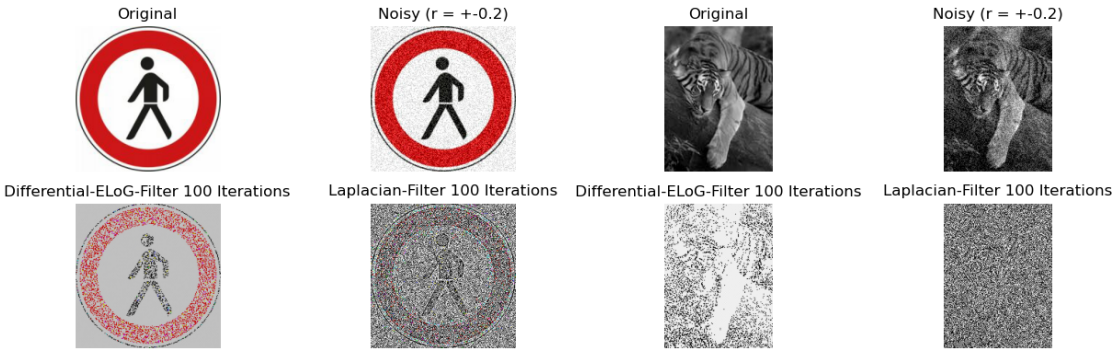


Figure 23: SNR and Sharpness Measurement on Grayscale-Image 21b

ELoG and Laplacian filters are compared in Figures 24a and 24b. In the case of noisy images, the vulnerabilities of the Laplacian filter become obvious. Noisy inputs cause steeper gradients than normal inputs due to the high difference between neighboring pixel values. Typical images have smoother transitions between their pixels, resulting in flatter gradients. This is confirmed by Fig. 24, where the Laplacian amplifies the noise from the base image. In the grayscale image, no structures are recognizable anymore, while the RGB image still retains a few edges but is overshadowed by more noise than the original image.



(a) Comparison on RGB-Images (b) Comparison on Grayscale-Images

Figure 24: Partial inverted ELoG- and Laplacian filtered Images

In contrast, the ELoG filter in the case of RGB images exhibits clean edges and fewer noisy perturbations. Observing the grayscale sample, ELoG appears to maintain a few structures outlining the edges around the tiger. Despite the fact that the tiger is no longer recognizable, ELoG is less susceptible to noise and retains more information than the Laplacian filter.

3.3. CIC: Local Filter Evaluation in Convolutional Layers

As part of my research, I discovered a significant lack of methods for monitoring and comparing local filters in the form of numerical results. To address this gap, I created a mechanism for evaluating influential filters, which is used in a well-known visualization method for convolutional neural networks. This approach, known as *Channel-wise Increase of Confidence (CIC)*, serves as the foundation for Score-CAM. I adapted CIC from existing code for Class Activation Mappings (CAM) from the repository "TorchCAM:classactivationexplorer" [38] and extended it to develop a method for local evaluation. Although I cannot fully confirm the validity of this method, comprehensive tests can be utilized to compare filters and layers between models, which I believe confirms the method's significance. This chapter provides a brief overview of CAM visualization, its extensions, and its differences, as these are also used to evaluate the pattern-pruned models. Building on this, I explain CIC as an evaluation method. The fundamental equations of the different CAM variants are presented in modified versions to emphasize their similarities and highlight their mathematical differences. The equations are consistently shown in yellow boxes, while blue boxes are used to highlight the equations based on Score-CAM, which have been implemented in the CIC evaluation method.

3.3.1. CAM: Class Activation Mapping

Class Activation Mapping is a technique for visualizing and interpreting neural networks, particularly CNNs. The motivation behind CAM is to determine which regions of an image contribute to a specific classification. This technique enables the identification of relevant areas within the input image that the model considers critical for its prediction [8, p. 1–2]. The fundamental equation for CAM is:

$$L^c_{CAM} = \text{ReLU}\left(\sum_k \alpha^c_k A^k_{l-1}\right) \tag{48}$$

The last convolutional layer is indexed $l-1$. With weighting α on activation map A^k_{l-1} as shown by Wang et al. [9, p. 2]:

$$\alpha^c_k = w^c_{l,l+1}[k] \tag{49}$$

Where $w^c_{l,l+1}[k]$ is defined as the k-th neuron weight toward class c regarding the last global pooling layer l and the following fully connected layer $l+1$. CAM utilizes the weights of the last fully connected layer $l+1$ to weight the activation maps of the last convolutional layer $l-1$, to generate a weighted saliency map L^c_{CAM}. ReLu activation function is used to only account positive influences on class c. This map is represented as a heatmap, highlighting important regions important for developers to provide insights into the functioning of CNNs. This is particularly useful for debugging neural networks. However, a limitation of CAM is that it is restricted to specific architectures and requires a global pooling layer after the last convolutional layer. Due to this limitation, specialized CAM methods such as GradCAM, GradCAM++ and ScoreCAM have been developed to enable architecture independent interpretations. Equations in (50) present the saliency map function on the left and the Grad-CAM specific activation map weighting function on the right side [9, p. 2].

$$L^c_{\text{Grad-CAM}} = \text{ReLU}\left(\sum_k \alpha^c_k A^k_l\right) \quad \Bigg| \quad \alpha^c_k = GP\left(\frac{\partial Y^c}{\partial A^k_l}\right) \tag{50}$$

Grad-CAM uses gradients instead of weights and is not limited by architectural constraints such as the requirement of a pooling layer or being restricted to a single fully connected layer [9, p. 2]. Here, Y^c represents the probability associated with the predicted class, while A^k_l represents the k-th activation map in convolutional layer l. Below are the equations for an advanced version of Grad-CAM called Grad-CAM++ [8]. I modified the Equations (51) for alignment purposes between all CAM equations.

$$L^c_{\text{Grad-CAM++}} = \text{ReLU}\left(\sum_k \alpha^c_k A^k_l\right) \quad \Bigg| \quad \alpha^c_k = \left[\frac{\frac{\partial^2 Y^c}{\partial (A^k_l)^2}}{2\frac{\partial^2 Y^c}{\partial (A^k_l)^2} + \sum_a \sum_b A^k_{ab} \frac{\partial^3 Y^c}{\partial (A^k_l)^3}}\right] \cdot \left(\frac{\partial Y^c}{\partial A^k_l}\right) \tag{51}$$

Chattopadhyay et al. wanted to create a more precise and reliable saliency map that not only identifies the regions influencing the target prediction but also differentially weights their relative contributions toward it [8]. This method takes into account the various derivatives of the activation functions to gain deeper insights into the model's behavior. Specifically, the first derivative allows to identify the activation map with the greatest impact on classification. The second derivative helps to understand the curvature direction of the classification function, revealing how the influence curve behaves. The third derivative provides information about the gradient's curvature behavior i.e., the third derivative indicates the curvature properties of the first derivative. This ap-

proach leads to better visualizations, particularly in cases where multiple instances or parts of an object are present in the image. It improves the model's transparency and strengthens confidence in its predictions [8, pp. 3, 11]. Grad-CAM and Grad-CAM++ face the issue of gradients becoming noisy and vanishing in saturated or zero-valued regions of the activation function, leading to inaccurate gradients. Score-CAM addresses this issue by eliminating the dependence on gradients and instead calculates a score C^k for weighting based on the activation maps themselves. Equations (52) illustrate the base formulas for saliency maps in Score-CAM [9, pp. 8, 2–3].

$$L^c_{\text{Score-CAM}} = \text{ReLU}\left(\sum_k \alpha^c_k A^k_l\right) \quad \Big| \quad \alpha^c_k = C^k(A^k_l) \tag{52}$$

The identification of classification-relevant features based on these scores lead to more accurate saliency maps [9, p. 8]. Now the question arises, what is this score and how is it different from the previous approaches?
In their publication, Wang et al. introduce the concepts of *Increase of Confidence* and their advancement *Channel-wise Increase of Confidence (CIC)*, as the core mechanism behind Score-CAM [9, p. 3]. I utilize this mechanism to evaluate pattern pruning optimization, which results in visual interpretations and scalar metrics. The following chapter will explain the principles behind CIC and the measurement methods that derive from it.

3.3.2. Channel-Wise Increase of Confidence

Channel-wise Increase of Confidence is the method by which regions of the base image are assigned a score corresponding to their importance. Score-CAM utilizes this benchmarking technique to rank image features. Each activation map is converted into a mask, which is multiplied with the input image to compare the change in prediction after each masked inference. Initially, the activation maps and the baseline prediction must be computed through inference from the original input. Equation (53) shows the principle of score assignment C [9, p. 3].

$$C^k(A^k_l) = f(X_b \circ H^k_l) - f(X_b) \tag{53}$$

$f(X_b)$ corresponds to the model inference on the original image which is compared against the masked inference $X_b \circ H^k_l$. Increase of confidence follows the same strategy as CIC but differs in mask generation and values. Masks for utilizing Increase of confidence are created manually and are not based on activation maps. These masks consist of binary values $H : \mathbb{R} \to \{0, 1\}$ [9, p. 3].

CIC instead creates continuous valued masks based on activation maps $H_{CIC} : \mathbb{R} \to [0,1]$. Equation (54) defines the process of mask generation used in Score-CAM and thus also in CIC [9, p. 3].

$$H_l^k = s(\mathrm{Up}(A_l^k)) \tag{54}$$

These mentioned masks, which are applied to the input image, are created by first upsampling the activation maps as illustrated in Figure 25 by using *bilinear interpolation*, followed by a normalization through the function s in Equation (55).

$$s(A_l^k) = \frac{A_l^k - \min A_l^k}{\max A_l^k - \min A_l^k} \tag{55}$$

The key difference between the methods is that normalization distinguishes CIC from the former approach Increase of Confidence. Binary masks completely remove regions of the input image from classification. CIC utilizes softer masks, creating a slight highlighting/ shadowing effects on the samples instead of removing entire areas. Negative scores are filtered out using the ReLU activation function (52) and normalized again through the *softmax* function to produce saliency maps with a limited value range [9, pp. 3–5]. Even though gradient-based class activation maps have minor weaknesses, they offer a different view enabled through gradients. I will use them along with Score-CAM to gain insights from the optimized model. Since GradCAM++ provides more precise results than its predecessor and has overcome some initial issues, I have decided to adopt it for evaluation.

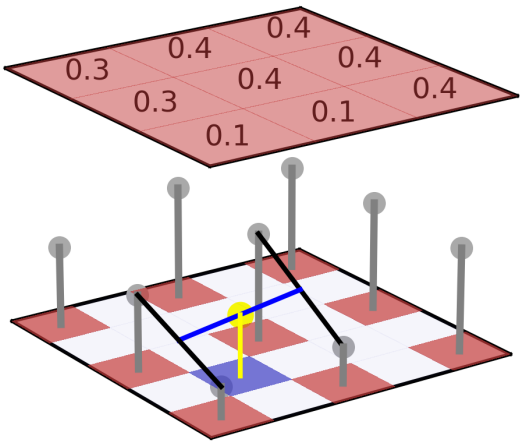


Figure 25: Illustration of Upsampling via Bilinear Interpolation

The CIC-based evaluation method used in this thesis, slightly deviates from the approach used in Score-CAM. Unlike the visual interpretation method, I do not filter out negative scores or perform a final normalization in between the scores of the measured layer. Score-CAM is designed to create saliency maps that only show positive influences in the form of a heatmap over the input image. In contrast, I aim to measure both positive and negative influences of the activation maps. As in the batch processing of CIC, the resulting score is created by Frobenius-norm $\|\cdot\|_F$ over the samples. My goal is to obtain metrics that correspond to the activation maps of the pattern-pruned filter kernels. It is essential to see how these filter kernels affect classification, both positively and negatively. Normalizing the scores as in ScoreCAM would distort the metrics in comparison evaluations between various models. The metrics would be limited to the range of [0,1] and would

not reflect the intensity relative to other models, but only the intensity relative to their own activation maps. Using CIC, I evaluate the individual activation maps of specific layers, as well as the entire architecture of the optimized models. The evaluation of the architecture method sums up the scores within a convolutional layer into scalar values, seperated into positive and negative values. For comparison with other models, I combine them through normalization and inverted normalization, as a higher negative score indicates a lower filter capacity, while on the other hand a higher positive score indicates higher filter capacity.

- $C_l^{[N_i]+}$ represent the positive score at layer l for model N_i
- $C_l^{[N_i]-}$ represent the negative score at layer l for model N_i

$$\text{norm}(C_l^{[N_i]+}) = \frac{C_l^{[N_i]+}}{\max(C_l^{[N_i]+}) - \min(C_l^{[N_i]+})}$$

$$\text{inv\_norm}(C_l^{[N_i]-}) = 1 - \frac{C_l^{[N_i]-}}{\max(C_l^{[N_i]-}) - \min(C_l^{[N_i]-})}$$

$$\tilde{C_l}^{[N_i]} = \frac{\text{norm}(C_l^{[N_i]+}) + \text{inv\_norm}(C_l^{[N_i]-})}{2} \tag{56}$$

Equation (56) shows how positive and negative CIC scores are combined into a single value, for comparison between models. My CIC evaluation method currently lacks of scientific validation and should be interpreted with caution. CIC provides valid results in the context of Score-CAM, as it only shows relative differences in between a single model's layers and not between different models. Despite this, I have obtained explainable results from the CIC method and use them with caution.

4. Experimental Setup

As a part of this work, a CUDA-enabled framework was developed, allowing for the modular integration of various pruning strategies. This framework supports the adaptive inclusion of testing procedures and the easy expansion, control and parameterization of adversarial tests. Additionally, model-specific configuration files can be generated automatically, enabling modifications to be exchanged with minimal effort. The framework is published under the GitHub account https://github.com/Dontoronto. As a test setup, a MacBook Pro M1 2020 with 16 GB of RAM, a tower PC with a Ryzen 7 3700x CPU and a Nvidia GeForce RTX 2060 with 6 GB of GDDR6 memory and 32 GB of RAM were used. LeNet consists of approximately 60,000 parameters and is trained using the relatively small MNIST dataset. This dataset comprises 60,000 training samples and 10,000 test samples in the form of single-channel 28x28-pixel grayscale images, divided into 10 classes [39]. The informational content that can be extracted from MNIST is relatively limited. Additionally, the LeNet model is a small model which is very suitable for testing purposes due to its short inference times, but offers only a few real-world applications. To implement pattern pruning on LeNet, some modifications were made to the convolutional layers. Specifically, the filter kernel size was adjusted from 5x5 to 3x3. As a result of these changes, the number of parameters increased to 105,918. For small models like the LeNet model, the performance of the MacBook Pro M1 processor was completely sufficient and could typically perform the ADMM optimization of the LeNet model within an hour. The same computations could be significantly accelerated using CUDA. The CUDA optimization duration under computationally light configurations was approximately $0:12h$, while $0:30h$ was needed for optimization with adversarial samples and pattern pruning. Both times were measured for 62 epochs, 200 ADMM-Iterations and a batch size of 32. Adversarial samples accounted for 20 percent of the batch size.

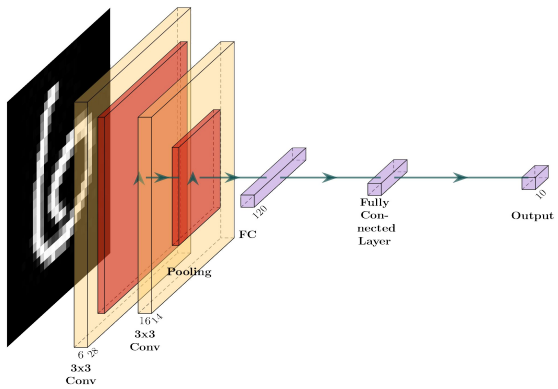


Figure 26: Illustration of the LeNet Architecture

ResNet, also referred to as a *residual neural network* [40], is a more advanced CNN architecture compared to LeNet. ResNet's success is based on the approach of multiple skip connections in parallel to convolutional layers. By using skip connections, the weights are less affected by vanishing gradients. There are various types of ResNet models that differ significantly in the number of layers and parameters. Due to limited hardware resources, I selected the smallest variant, ResNet18, as the test subject. ResNet18 consists of approximately 11.8 million trainable parameters and is

trained and optimized using the ILSVRC2012 ImageNet-1k dataset [41]. This dataset comprises 1,281,167 training samples and 50,000 validation samples (test samples from ImageNet-1k are not used due to the lack of label assignments). The dataset is divided into 1,000 different classes with samples of varying quality and size. Before this dataset can be used, it must be standardized through a *preprocessing pipeline*. This involves rescaling, cropping and normalizing the samples to make them suitable for the ResNet model. Since the ResNet18 model with the ImageNet dataset is significantly larger than the LeNet model with the MNIST dataset, it more closely resembles to real-world applications. Because of slow ADMM optimization, optimizing ResNet18 on the MacBook was nearly impossible. GPU computations using Apple's *Performance Shader (MPS)* required longer optimization times than using the CPU. The lack of maturity in the PyTorch framework regarding adaptation to Apple's new ARM processor line led to this slow performance. Both variants were not compatible with the time constraints of this thesis. The optimization using the Ryzen 7 CPU showed the worst performance among all computing platforms and could not be used in this project. As a result, ResNet18's optimization was based on CUDA. The optimization time for approximately 17 epochs, 2500 ADMM-Iterations and batch size of 64 with the most resource-efficient optimization configuration took 15 hours. The most ressource-intensive optimization to measure took 51 hours .

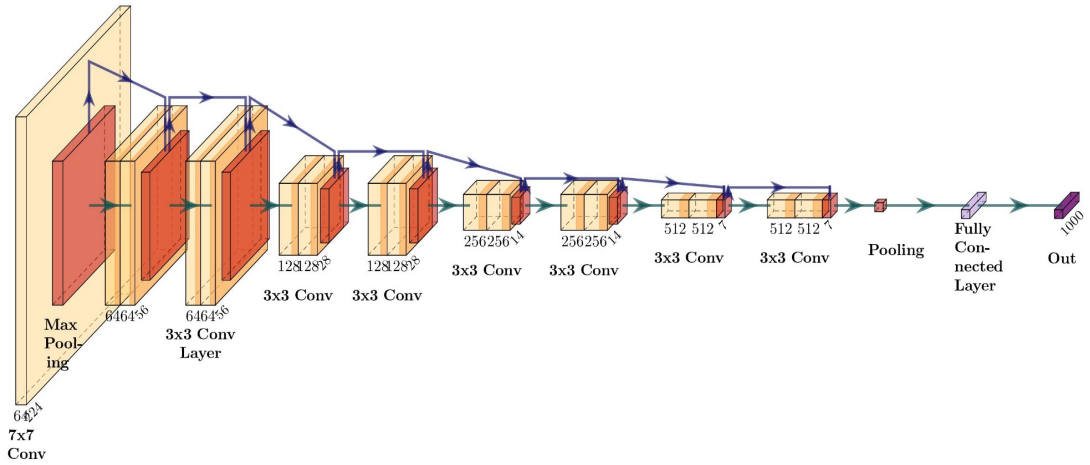


Figure 27: Illustration of the ResNet18 Architecture

5. Evaluation

In this chapter, the results of the examined techniques are evaluated to investigate the advantages and disadvantages of the regularization technique named pattern pruning. This chapter aims to contribute to answering the research question, "Can pattern pruning enhance the robustness and filter capacity of convolutional neural networks against adversarial attacks?" The focus is on the application of pattern pruning using the numerical optimization method known as alternating direction method of multipliers. Pattern pruning is applied in various forms and with additional mechanisms such as adversarial training to show the performance of the regularization technique from a broader angle. The selected techniques focus specifically on structured pruning, as this opens up the possibility of subsequently adapting and accelerating the model on specific hardware. In view of the possibility of even greater acceleration, this chapter evaluates the extension of pattern pruning additionally through *connectivity pruning* [5]. This technique removes entire filter kernels and allows for even higher acceleration of convolution operations by eliminating entire convolution matrices from the model inference, as seen in Chapter 2.3.2. Appendix A.2 details how the ADMM-defined equations can be combined with connectivity pruning. The evaluation of connectivity pruning, as well as the models with adversarial training, are considered a bonus and are not focus of this work. They merely serve to assess whether the filter capacity and robustness can be enhanced through pattern pruning.

The slightly modified LeNet model with the MNIST dataset and the ResNet18 model with the ImageNet-1K dataset are examined. First, general benchmark measurements such as the accuracy of the respective test sets, accompanied by the corresponding pruning characteristics, are presented. Subsequently, the accuracies on manipulated test sets without adversarial techniques are examined. These perturbations are based on uniformly applied noise at various intensity levels over the input image. Following the random perturbations, the success rates of various adversarial attacks on the optimized models are evaluated. For the visual interpretation of the models, feature maps, saliency maps and class activation maps are presented to illustrate the focus of the models on the original images of successful adversarial attacks. As the final test, I will present and interpret the results of my benchmark method derived from Channel-wise Increase of Confidence. Finally, the collected results will be presented to determine whether the filter capacity and adversarial robustness can be increased through pattern pruning.

5.1. Accuracy Evaluation

This chapter focuses on the accuracym achieved by the models. Tables highlighted in yellow indicate their association with the LeNet architecture, while tables highlighted in blue indicate their association with the ResNet18 architecture. Each model of the respective architecture was optimized under the same hyperparameters.

Pattern pruning and the comparison models are referenced by the following abbrevations:

Abbrevations	Reference Descriptions
$S:D, S-D$	SCP pattern pruning on the default training dataset
$S:A, S-A$	SCP pattern pruning on the adversarial-mixed training dataset
$T:D, T-D$	trivial pattern pruning on the default training dataset
$T:A, T-A$	trivial pattern pruning on the adversarial mixed training dataset
$U:D, U-D$	unstructured pruning on the default training dataset
$U:A, U-A$	unstructured pruning on the adversarial-mixed training dataset
B	base model, from which all optimized models are derived

Table 4 shows the LeNet accuracy of the base model, the model accuracies after ADMM regularization and the accuracies after retraining.

Pruning Type	Base Model	ADMM Model	Retrained Model
Unstructured Pruning	96.97%	99.07%	99.02%
Trivial Pattern Pruning	96.97%	99.1%	98.78%
SCP Pattern Pruning	96.97%	97.9%	98.82%

Table 4: Accuracy Measurements of Different LeNet Models Seperated by Optimization Phase on Default Dataset

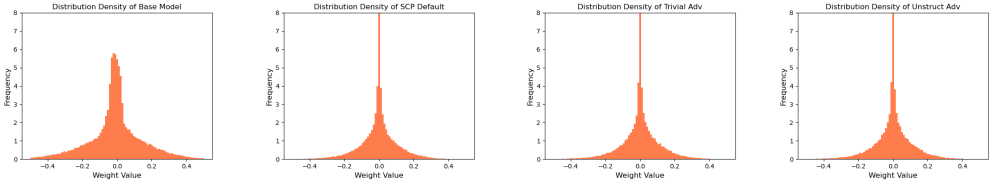
These models were trained on the standard MNIST dataset. The ADMM regularization comprised approximately 42 epochs with 200 ADMM-Iterations, a batch size of 32, a penalty value ρ of 0.0005, gradient clipping of 1 and a constant learning rate of 1e-5. The retraining spanned 10 epochs. The pruning rates and weight distribution densities for the LeNet models are shown in Table 28. Due to the small size of the convolutional layers, low pruning rates of only 0.48% are achieved. The trivial pattern pruning method reduces the number of applied patterns to 4 to enable a fair comparison with SCP pattern pruning.
In terms of the base model, the accuracy was improved for all pruning methods. A slight decrease in accuracy from the pure ADMM regularization to the retrained model is observed. This is because no direct pruning occurs in ADMM regularization. The model parameters are driven toward zero but do not necessarily become zero values. During the retraining step, these values are directly masked and set to zero. This initially causes a drop in accuracy, which is restored during retraining. As expected, unstructured pruning achieved the highest accuracy. As mentioned in the theoretical foundations Chapter 2.3, unstructured pruning is characterized by its flexible selection of pruning parameters, capturing the best combinations to achieve the highest possible accuracy. Although structured pruning is known for lower accuracy due to fixed pattern specifications, both pattern pruning variants can demonstrate improved accuracy. The application of mathematically derived sparse convolutional patterns shows a slightly better accuracy compared to the trivial pattern pruning method, where all patterns are available and gradually reduced.

Pruning Type	Base Model	ADMM Model	Retrained Model
Adv. Unstructured Pruning	96.97%	99.14%	99.04%
Adv. Trivial Pattern Pruning	96.97%	99.13%	98.16%
Adv. SCP Pattern Pruning	96.97%	97.8%	98.65%

Table 5: Accuracy Measurements of Different LeNet Models Seperated by Optimization Phase With 20% Adversarial Manipulation on Batches

In Table 5, the same optimization methods are applied, with the difference that 20% of the training dataset was manipulated using the adversarial attack PGD. This corresponds to adversarial training as an adversarial defense strategy. PGD is an l_∞ attack and does not limit the number of altered pixels but sets an upper bound on the maximum change of single pixels. According to [17, p. 8], this is particularly suitable as it simultaneously targets all the model's vulnerabilities, enabling more generalized adversarial training. This can be seen as a kind of *data augmentation*. Adversarial training based on l_0 attacks would only alter a small number of pixels, leading to one-sided training. The changes made by PGD in adversarial training are limited to a maximum of $\epsilon = 1.0$.

As shown in Table 28 based on the default training dataset, the adversarial-pattern-pruned models exhibit similar structures. The difference between the trivial pattern pruning method and the SCP pattern pruning method becomes slightly more pronounced and clearer. Akronyms *Default* stands for the models trained with the default dataset while *Adv* refers to the adversarial mixed dataset. *U:* stands for unstructured-pruned model, *P:* for pattern-pruned model, while *:R* refers to pruning ratio and *:P* to the pruned parameters. *Total:P* represents the total number of parameters within the model.



(a) Base Model (b) SCP Default (c) Trivial Adv (d) Unstruct Adv

Layer	U:R	P:R	U:P	P:P	Total:P
conv1	55,56%	55,56%	30	30	54
conv2	55,56%	55,56%	480	480	864
fc1	0	0	0	0	94080
fc2	0	0	0	0	10080
fc3	0	0	0	0	840
Total	0,48%	0,48%	510	510	105918

Figure 28: Pruning Rates and Weight Distribution Densities of the LeNet Models

The differences become more pronounced when considering the ResNet model. Table 6 shows the accuracy rate of the ResNet18 models in the same structural representation as for the LeNet models.

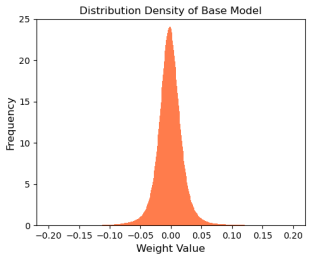
Pruning Type	Base Model	ADMM Model	Retrained Model
Unstructured Pruning	68.66%	69.35%	69.19%
Trivial Pattern Pruning	68.66%	67.82%	58.84%
SCP Pattern Pruning	68.66%	67.1%	67.27%
Adv. Unstructured Pruning	68.66%	69%	68.9%
Adv. Trivial Pattern Pruning	68.66%	67.39%	60.30%
Adv. SCP Pattern Pruning	68.66%	65.71%	66.96%

Table 6: Accuracy Measurements of Different ResNet18 Models Seperated by Optimization Phase on Default Dataset and With 20% Adversarial Manipulation on Batches

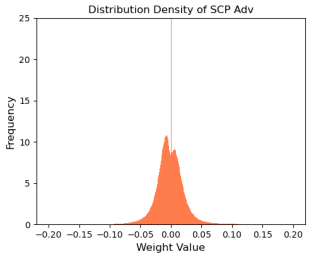
As with the LeNet models, the best accuracy was achieved through unstructured pruning. Unlike the LeNet models, the pattern pruning variants fall below the base model's accuracy. A slight decrease in accuracy due to adversarial training was also expected. This is because the decision boundaries are flattened as the model attempts to achieve correct classification despite the manipulated data. For a clearer illustration of the decision boundary, see Figure 5b. Adversarial attacks aim to exploit steep decision boundaries by making small changes at pixel coordinates with steep decision boundaries, which lead to significant changes in output [12, p. 4]. By accepting a small accuracy loss, higher robustness can be achieved. Although a loss was anticipated, pattern pruning based on SCP patterns shows very good results. On regular training data, a small drop of 1% is recorded, and on the manipulated dataset, a drop of 2% is observed. Compared to trivial pattern pruning, a significantly better classification rate was achieved. Such a slight drop is remarkable for a structured pruning technique. The advantages resulting from the acceleration and computational savings could compensate for the lower accuracy.

Different from the LeNet model, the ResNet model consists almost entirely of convolutional layers. Pattern pruning can result in much higher pruning rates. Pruning rates and weight distribution densities are shown in Table 29.

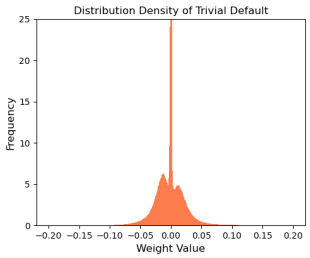
Layer	U:R	P:R	U:P	P:P	Total:P
conv1	0	0	0	0	9408
bn1	0	0	0	0	64
layer1.0.conv1	55,5%	55,56%	20460	20480	36864
layer1.0.bn1	0	0	0	0	64
layer1.0.conv2	55,5%	55,56%	20460	20480	36864
layer1.0.bn2	0	0	0	0	64
layer1.1.conv1	55,5%	55,56%	20460	20480	36864
layer1.1.bn1	0	0	0	0	64
layer1.1.conv2	55,5%	55,56%	20460	20480	36864
layer1.1.bn2	0	0	0	0	64
layer2.0.conv1	55,5%	55,56%	40919	40960	73728
layer2.0.bn1	0	0	0	0	128
layer2.0.conv2	55,5%	55,56%	81838	81920	147456
layer2.0.bn2	0	0	0	0	128
layer2.0.down.0	0	0	0	0	8192
layer2.0.down.1	0	0	0	0	128
layer2.1.conv1	55,5%	55,56%	81838	81920	147456
layer2.1.bn1	0	0	0	0	128
layer2.1.conv2	55,5%	55,56%	81838	81920	147456
layer2.1.bn2	0	0	0	0	128
layer3.0.conv1	55,5%	55,56%	163676	163840	294912
layer3.0.bn1	0	0	0	0	256
layer3.0.conv2	55,5%	55,56%	327352	327680	589824
layer3.0.bn2	0	0	0	0	256
layer3.0.down.0	0	0	0	0	32768
layer3.0.down.1	0	0	0	0	256
layer3.1.conv1	55,5%	55,56%	327352	327680	589824
layer3.1.bn1	0	0	0	0	256
layer3.1.conv2	55,5%	55,56%	327352	327680	589824
layer3.1.bn2	0	0	0	0	256
layer4.0.conv1	55,5%	55,56%	654705	655360	1179648
layer4.0.bn1	0	0	0	0	512
layer4.0.conv2	55,5%	55,56%	1309409	1310720	2359296
layer4.0.bn2	0	0	0	0	512
layer4.0.down.0	0	0	0	0	131072
layer4.0.down.1	0	0	0	0	512
layer4.1.conv1	55,5%	55,56%	1309409	1310720	2359296
layer4.1.bn1	0	0	0	0	512
layer4.1.conv2	55,5%	55,56%	1309409	1310720	2359296
layer4.1.bn2	0	0	0	0	512
fc	0	0	0	0	512000
Total	51,19%	52,24%	6096937	6103040	11683712



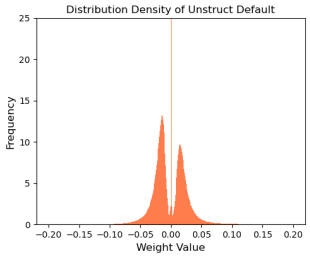
(a) Base Model



(b) SCP Adv Model



(c) Trivial Default Model



(d) Unstruct Default Model

Figure 29: Pruning Rates and Weight Distribution Densities of the ResNet18 Models

The ADMM- and the retraining-phase each comprise 8 epochs. The ADMM-Iterations occur every 2500 batch iterations, with a batch size of 64. The learning rate was reduced as training progressed. The initial learning rate was set to $1e-5$. Due to the large batch size, larger input images and the greater number of parameters, gradient clipping was increased to 24.

Before the chapter ends, figures will be presented to illustrate and compare the different accuracy rates more clearly. As previously mentioned, model data for the connectivity-pruned models, is also included. Using additional connectivity pruning I could reach up to 73, 13% pruning rate on ResNet18 and with unstructured pruning on top of connectivity pruning 51, 75% on LeNet models. These models will be labeled with the acronym *Conn*.

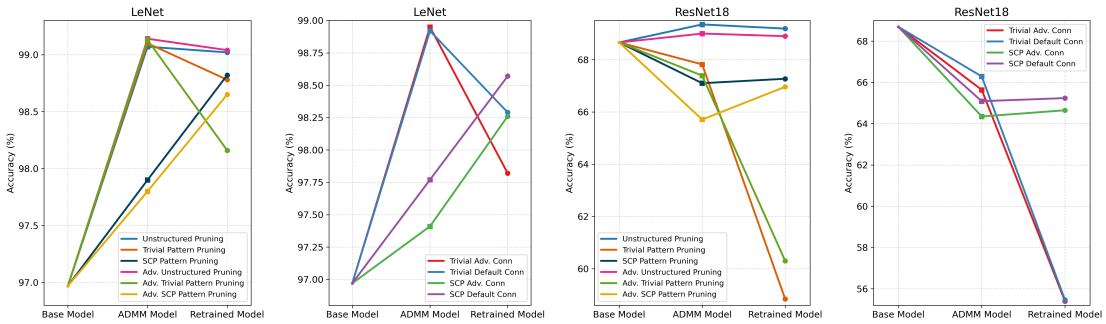


Figure 30: LeNet & ResNet18 Accuracy Graphs With Optimized Models Including Connectivity Pruning

5.2. Measurements on Perturbated Data

In this chapter, measurements based on manipulated input data are evaluated. Each model of the respective architecture was optimized under the same hyperparameters. LeNet data is highlighted in yellow and ResNet data is highlighted in blue. Naming conventions in Figures and Tables are also adopted from Chapter 5.1. The perturbations generated in the following test for the LeNet models were measured to have an l_2-norm in the range of 24 to 29.

Typ	Pruning Typ	r=0,2	r=0,36	r=0,52	r=0,68	r=0,84
	Base-Model	95,97%	96,19%	89,22%	78,43%	66,18%
Default	Unstruct	97,5%	95,01%	85,4%	74,06%	63,75%
	Trivial	96,26%	95,36%	91,35%	81,17%	69,31%
	SCP	96,2%	94,3%	90,52%	85,54%	79,4%
Adversarial	Unstruct	97,9%	96,09%	89,48%	80,32%	70,14%
	Trivial	97,15%	95,97%	91,19%	81,39%	68,34%
	SCP	98,27%	98,03%	97,31%	95,32%	90,96%
Connectivity Default	Trivial	88,31%	91,43%	91,88%	90,37%	85,31%
	SCP	92,16%	91,84%	90,52%	84,37%	77,1%
Connectivity Adversarial	Trivial	97,13%	96,68%	94,3%	89,2%	77,81%
	SCP	97,6%	97,35%	96,11%	92,49%	84,84%

Table 7: Random Noise Overlay Accuracy Measurement of LeNet Models

Table 7 shows the accuracy of the different models on noisy images. applied random noise with intensity r to the test data from the MNIST dataset. The table is divided according to the training type used and the respective pruning technique that the models were optimized for. Connectivity pruning is not within the scope of this work and is therefore treated as supplementary information. As noise intensity increases, the table shows a loss of accuracy. Each of the pattern-pruned models, in the case of both default and adversarial training, achieved almost exclusively better accuracies than the base model. Pattern pruning with SCP patterns achieved the best results in this test and is highlighted in green. The sparse convolutional patterns originate from the Laplacian filter, which

was demonstrated in previous chapters and is used for edge detection and extraction. Since the MNIST dataset consists of grayscale images with very rapid and strong changes in color intensity, the potential of the SCP models can be fully realized. The trivial-generated patterns also achieve very good results, showing superiority over unstructured pruning on more intensive noise ratios. Pattern pruning dominates in more perturbated images. In the case of adversarial-trained models the SCP model entirely dominates all other models.
I measured poorer results in the test runs of the ResNet models. The ImageNet images contain 192x times more pixel information than the MNIST dataset. Consequently, there are many more possibilities in the representations. Unlike the MNIST images, there are fewer sharp edges and more color gradients. Unstructured pruning achieves the highest accuracy among the optimized models and can maintain the accuracy of the base model with some tolerance. The SCP model on default training performed best as representative for pattern pruning. Even at cases of increasing noise ratio the pattern pruning method could not reach the accuracy of the base model. Perturbations generated in the following test for the ResNet18 models were measured to have an l_2-norm in the range of 480 to 561.

Typ	Pruning Typ	r=0,2	r=0,36	r=0,52	r=0,68	r=0,84
	Base-Model	47,34%	36,19%	25,16%	16,85%	11,2%
Default	Unstruct	44,77%	32,26%	21,83%	14,4%	9,61%
	Trivial	30,92%	15,59%	7,58%	3,82%	2,2%
	SCP	39,25%	24,72%	14,33%	8,67%	5,29%
Adversarial	Unstruct	40,08%	26,2%	15,68%	9,03%	5,58%
	Trivial	30,01%	15,3%	7,62%	4,03%	2,31%
	SCP	35,99%	20,76%	10,63%	5,55%	3,0%
Connectivity Default	Trivial	28,04%	14,26%	6,78%	3,47%	2,03%
	SCP	36,4%	21,63%	12,2%	7,24%	4,34%
Connectivity Adversarial	Trivial	27,37%	13,48%	6,09%	3,08%	1,76%
	SCP	32,08%	17,33%	8,96%	4,95%	2,82%

Table 8: Random Noise Overlay Accuracy Measurement of ResNet18 Models

Worth mentioning are the good results of the additional connectivity-pruned models. Even though they have fewer parameters because of the use of another structured pruning technique on top. The connectivity-pruned SCP model based on the default dataset exhibits overall better accuracy than the other pattern-pruned models. For LeNet, the connectivity-pruned models achieved, in almost every case, better results than the base model. Example images for different noise ratios on MNIST and ImageNet datasets can be seen in Appendix A.3.

Measurements of adversarial attacks are presented in the following. For understanding the adversarial test runs, it should be noted that adversarial attacks in the case of whitebox attacks are highly adapted to the specific model. The attacks have extensive information about them. In real-world attack scenarios, this information is not available. But for the purpose of investigating filter capacity and adversarial robustness, these attacks are perfectly suited for evaluation, as very targeted attacks can be launched to expose the weaknesses of neural networks.

Almost no scientific work in the field of adversarial attacks and defenses mentions the *preprocessing* of input images. Possibly, this would disprove some results or one prefers the theoretical rather than the practical domain. Preprocessing plays an important role in the generation of adversarial attacks in the context of this thesis. Models in the industry are trained with incredibly large datasets. These data can come from different sources, have different formats, various color settings, lighting conditions or even disturbances. To counteract the large deviations in raw data, the data is preprocessed through a *preprocessing pipeline* to create a certain data standard. Models are trained on preprocessed data. In the case of ImageNet-1K, for example, the dimensions are adjusted and the image edges are cropped, as shown in Figure 6. Subsequently the pixel values are normalized using standard deviations and average values. Test data are also preprocessed, as the model was trained on higher-quality data. The model requires preprocessing to achieve high accuracy since it was specifically trained on such preprocessed samples. Models based on raw data achieve poorer results than models based on preprocessed data.
These points became important as adversarial attacks were developed and tested. Successful attacks were confirmed, but they were not reproducible. The comprehensive project "Adversarial -Attacks-PyTorch" [42] was used to apply adversarial attacks. Numerous scientific works use attacks from this repository. Upon closer investigation of the code, it became clear that normalization is removed before the adversarial sample is generated and then added back afterward. This means that adversarial attacks operate on raw data and report them as success. Once these raw data are preprocessed, they do not necessarily lead to a false classification. This aspect should be more frequently explained in scientific work. Taking these points into account, attacks that do not withstand preprocessing are not considered successful. Only samples leading to false predictions despite preprocessing are considered successful. Attacks like *DeepFool* are more affected by this, as they attempt to produce false outputs with minimal changes to the sample, which can often be compensated by preprocessing. The l_2-norm limit as well as the intensity of the attacks were intentionally set higher, as small perturbations are easily compensated by preprocessing and thus can hardly cause false predictions.

As shown by Ye et al. [43], verified adversarial robustness is only possible by increasing the network size and thereby the network capacity. Since this work aims at hardware optimization as a long-term goal, the network size is not increased, and the measurements are conducted with a constant number of parameters. Full robustness is not expected, but rather smaller successes through a slight increase in filter capacity and thereby adversarial robustness. As a result, only 20% of the samples in a training batch were modified by incorporating weaker adversarial attacks. This approach is intended to avoid over-stressing the network capacity and causing fewer accuracy drops [43, p. 118].

A few notes on data collection should be made first. Adversarial attacks that exceeded the limit of the respective l_2-norm are excluded. In the diagrams, these are marked as *O* for overflow. Additionally, adversarial-generated samples that were already falsely classified without the addition of perturbations are excluded. I denote these as false-positive values, abbreviated *F-P*. Attack parameters are located in Appendix A.4. Tests are split into weak, medium and strong settings. Measurements are conducted by applying the test runs to 100 random samples from datasets while maintaining the same samples for a single test run. Bar charts accompany the tables for a simple visualization of relative adversarial success rates between the models. Three colors distinguish the table cells, highlighting the best results. Green defines the best, yellow the second-best and or-

ange the third-best adversarial robustness against the specific attack. Because of high similarities, weaker attacks only highlight the top-2 models.

$l_2 = 1.4$	DeepFool			JSMA			OnePixel			PGD		
Models:	S	O	F-P	S	O	F-P	S	O	F-P	S	O	F-P
Base Model	1%	18%	0%	0%	23%	0%	0%	0%	0%	1%	0%	0%
Unstruct Default	2%	12%	0%	0%	16%	0%	0%	0%	0%	2%	0%	0%
Unstruct Adv	4%	43%	0%	0%	1%	0%	0%	0%	0%	1%	0%	0%
Trivial Default	1%	13%	0%	0%	13%	0%	1%	0%	0%	3%	0%	0%
Trivial Adv	13%	42%	2%	1%	1%	2%	0%	0%	2%	2%	0%	2%
SCP Default	0%	15%	1%	0%	8%	1%	0%	0%	1%	1%	0%	1%
SCP Adv	18%	43%	1%	0%	2%	1%	0%	0%	1%	2%	0%	1%
Trivial Default Conn	0%	16%	1%	0%	2%	1%	0%	0%	1%	1%	0%	1%
Trivial Adv Conn	5%	46%	2%	1%	2%	2%	2%	0%	2%	2%	0%	2%
SCP Default Conn	0%	15%	2%	1%	9%	2%	0%	0%	2%	0%	0%	2%
SCP Adv Conn	15%	37%	2%	1%	2%	2%	0%	0%	2%	1%	0%	2%

Table 9: Soft Adversarial Attacks on LeNet Models

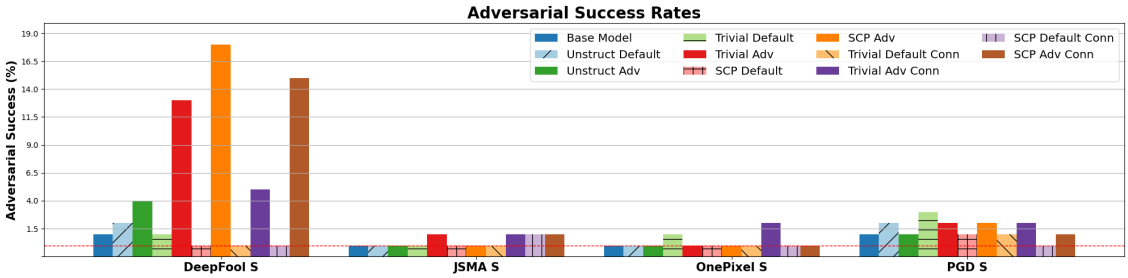


Figure 31: Soft Adversarial Attacks on LeNet Models

$l_2 = 3.0$	DeepFool			JSMA			OnePixel			PGD		
Models:	S	O	F-P	S	O	F-P	S	O	F-P	S	O	F-P
Base Model	11%	15%	0%	3%	0%	0%	4%	6%	0%	25%	0%	0%
Unstruct Default	10%	14%	0%	1%	0%	0%	8%	5%	0%	21%	0%	0%
Unstruct Adv	24%	68%	0%	0%	0%	0%	16%	7%	0%	42%	0%	0%
Trivial Default	3%	12%	0%	1%	0%	0%	4%	5%	0%	18%	0%	0%
Trivial Adv	41%	55%	1%	2%	0%	1%	8%	3%	1%	52%	0%	1%
SCP Default	4%	12%	1%	1%	0%	1%	5%	6%	1%	18%	0%	1%
SCP Adv	28%	60%	1%	0%	0%	1%	11%	6%	1%	43%	0%	1%
Trivial Default Conn	2%	13%	2%	0%	0%	2%	5%	4%	2%	20%	0%	2%
Trivial Adv Conn	35%	39%	2%	1%	0%	2%	6%	8%	2%	45%	0%	2%
SCP Default Conn	0%	13%	0%	2%	0%	0%	11%	5%	0%	16%	0%	0%
SCP Adv Conn	39%	50%	2%	0%	0%	2%	14%	11%	2%	51%	0%	2%

Table 10: Medium Adversarial Attacks on LeNet Models

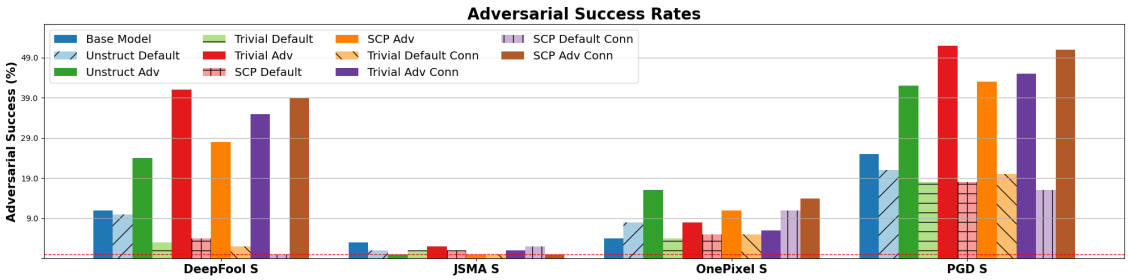


Figure 32: Medium Adversarial Attacks on LeNet Models

$l_2 = 4.5$	DeepFool			JSMA			OnePixel			PGD		
Models:	S	O	F-P	S	O	F-P	S	O	F-P	S	O	F-P
Base Model	24%	13%	1%	4%	0%	1%	15%	3%	1%	1%	98%	1%
Unstruct Default	11%	9%	1%	4%	0%	1%	18%	0%	1%	0%	99%	1%
Unstruct Adv	47%	47%	3%	0%	0%	3%	45%	2%	3%	89%	8%	3%
Trivial Default	8%	6%	3%	5%	0%	3%	16%	2%	3%	0%	97%	3%
Trivial Adv	61%	33%	2%	4%	0%	2%	28%	0%	2%	78%	20%	2%
SCP Default	4%	8%	1%	2%	0%	1%	11%	3%	1%	1%	98%	1%
SCP Adv	51%	38%	2%	6%	0%	2%	36%	2%	2%	97%	1%	2%
Trivial Default Conn	1%	8%	4%	6%	0%	4%	15%	4%	4%	0%	96%	4%
Trivial Adv Conn	51%	25%	4%	3%	0%	4%	33%	2%	4%	64%	32%	4%
SCP Default Conn	2%	5%	3%	3%	0%	3%	20%	1%	3%	68%	0%	3%
SCP Adv Conn	54%	28%	2%	6%	0%	2%	45%	0%	2%	97%	1%	2%

Table 11: Strong Adversarial Attacks on LeNet Models

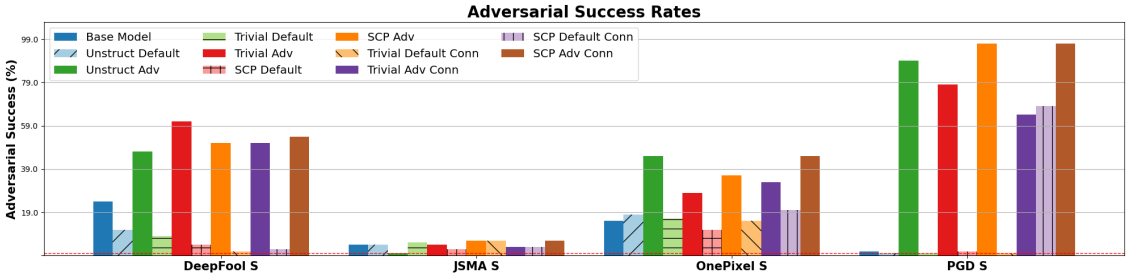


Figure 33: Strong Adversarial Attacks on LeNet Models

In Table 20, the models perform almost equally. There are no significant differences between the pattern-pruned models, the base model and the unstructured-pruned models. The only notable observation is that the adversarial-trained models record higher adversarial successes and overflows in the case of the DeepFool attack. This aspect will be discussed in greater detail after the LeNet model is evaluated. The attacks in this test run were limited to $l_2 = 1, 4$.
A similar behavior is observed in Table 21. These attacks were limited to $l_2 = 3, 0$. Better robustness of pattern pruning compared to the base model can be seen. In comparison to the unstructured-pruned models, an improvement through lower success rates is also noticeable. Table 20 once again shows a high adversarial success rate and overflow rate in the field of adversarial-trained models.
On the strongest attacks in Table 22, most of the best-of-3 models can be seen in the pattern-pruned models category. The attacks were limited to $l_2 = 4, 5$. Generally, almost all pattern-pruned mod-

els based on the default dataset show greater robustness than the base model. The adversarial variants exhibit the least robustness.
Notably, the adversarial-trained models exhibit a very low overflow rate on PGD attacks. As previously mentioned, these models were trained based on these attacks. It is possible that the attacks were chosen too strongly and the model was trained in such a way that it causes a kind of oscillation on PGD-generated samples. Structures created by strong PGD samples are unable to find a suitable optimum during training and may not produce averages, but rather extrema. These extrema are then exploited by PGD to trick the model. Since this work does not focus on adversarial-trained models, we will only discuss this explanation as a hypothesis.
When looking more closely at DeepFool's overflow rates, it is evident that these are higher than those of the default models. One possible explanation is that the models' decision boundaries have been flattened. Smaller changes to the test samples produce little to no impact on the classification compared to the default models. An increased rate here indicates that DeepFool had to generate larger perturbations with more iterations than on the default models to fool the adversarial models. In the case of normal models, steeper decision boundaries cause larger changes in a single step to the output. These lead to an early termination of the attack, as a success is recorded. Since these attacks involve a smaller range of perturbations, they can be more easily compensated for by preprocessing. This is different for adversarial-trained models, which have flatter decision boundaries and are less susceptible to perturbations. Perturbations have a smaller impact on their output. Since DeepFool is an iterative algorithm, the sample is guided towards false classification in little steps. When there are more iterations and more targeted changes, adversarial attacks become more accurate and strong, and preprocessing can't really make up for that. Because these are whitebox attacks, they have extensive information about the model, enabling precise attacks. In the case of a blackbox attack, as would occur in the real-world, attackers would find it more challenging to identify weaknesses with a significant impact on the output. This indicates that increased robustness has been achieved on these. This, like the special behavior in the case of PGD attacks, will not be discussed further.

In the following, the results of the ResNet18 models are presented. These are shown using the same methodology. Once again, 100 random samples are used per test run to evaluate adversarial robustness.

$l_2 = 7.0$	DeepFool			OnePixel			PGD		
Models:	S	O	F-P	S	O	F-P	S	O	F-P
Base Model	4%	0%	23%	1%	0%	23%	0%	77%	23%
Unstruct Default	3%	0%	25%	0%	0%	25%	0%	75%	25%
Unstruct Adv	0%	0%	23%	2%	0%	23%	3%	27%	23%
Trivial Default	1%	0%	33%	0%	0%	33%	0%	67%	33%
Trivial Adv	0%	0%	35%	0%	0%	35%	0%	37%	35%
SCP Default	1%	0%	23%	0%	0%	23%	0%	77%	23%
SCP Adv	0%	0%	26%	0%	0%	26%	3%	28%	26%
Trivial Default Conn	1%	0%	37%	1%	0%	37%	0%	63%	37%
Trivial Adv Conn	0%	0%	41%	0%	0%	41%	0%	59%	41%
SCP Default Conn	2%	0%	26%	3%	0%	26%	0%	74%	26%
SCP Adv Conn	0%	0%	29%	0%	0%	29%	1%	21%	29%

Table 12: Soft Adversarial Attacks on ResNet18 Models

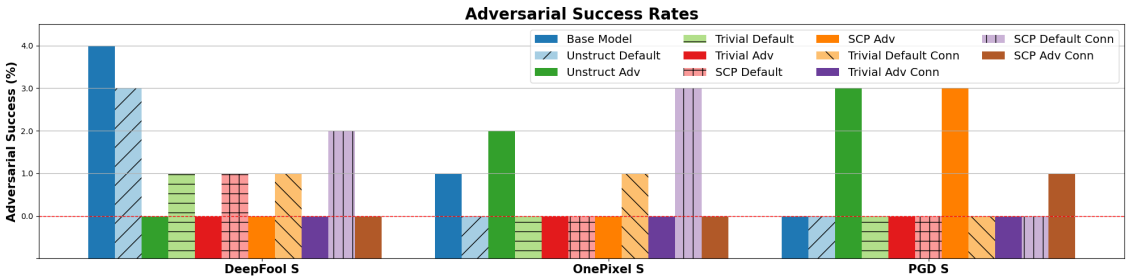


Figure 34: Soft Adversarial Attacks on ResNet18 Models

$l_2 = 5.0$	DeepFool			OnePixel			PGD		
Models:	S	O	F-P	S	O	F-P	S	O	F-P
Base Model	4%	0%	32%	5%	2%	32%	0%	68%	32%
Unstruct Default	6%	0%	32%	4%	1%	32%	0%	68%	32%
Unstruct Adv	0%	0%	31%	3%	1%	31%	5%	22%	31%
Trivial Default	1%	0%	47%	1%	1%	47%	0%	53%	47%
Trivial Adv	0%	0%	47%	3%	1%	47%	2%	30%	47%
SCP Default	7%	0%	31%	5%	1%	31%	0%	69%	31%
SCP Adv	0%	0%	34%	1%	1%	34%	1%	28%	34%
Trivial Default Conn	0%	0%	54%	4%	0%	54%	0%	46%	54%
Trivial Adv Conn	0%	0%	51%	3%	0%	51%	0%	49%	51%
SCP Default Conn	2%	0%	39%	3%	1%	39%	0%	61%	39%
SCP Adv Conn	0%	0%	42%	1%	0%	42%	2%	22%	42%

Table 13: Medium Adversarial Attacks on ResNet18 Models

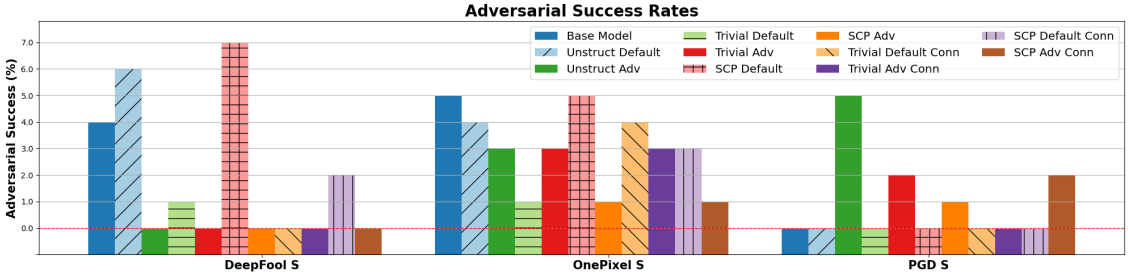


Figure 35: Medium Adversarial Attacks on ResNet18 Models

$l_2 = 8.0$	DeepFool			OnePixel			PGD		
Models:	S	O	F-P	S	O	F-P	S	O	F-P
Base Model	18%	0%	44%	5%	0%	44%	55%	0%	44%
Unstruct Default	19%	0%	48%	9%	0%	48%	46%	6%	48%
Unstruct Adv	0%	0%	51%	4%	0%	51%	6%	2%	51%
Trivial Default	4%	2%	51%	6%	0%	51%	13%	36%	51%
Trivial Adv	0%	0%	56%	4%	0%	56%	9%	3%	56%
SCP Default	15%	0%	44%	12%	0%	44%	54%	2%	44%
SCP Adv	0%	0%	46%	7%	0%	46%	8%	0%	46%
Trivial Default Conn	4%	1%	55%	6%	0%	55%	8%	37%	55%
Trivial Adv Conn	0%	0%	52%	8%	0%	52%	17%	9%	52%
SCP Default Conn	12%	1%	48%	9%	0%	48%	48%	4%	48%
SCP Adv Conn	0%	0%	50%	6%	0%	50%	9%	0%	50%

Table 14: Strong Adversarial Attacks on ResNet18 Models

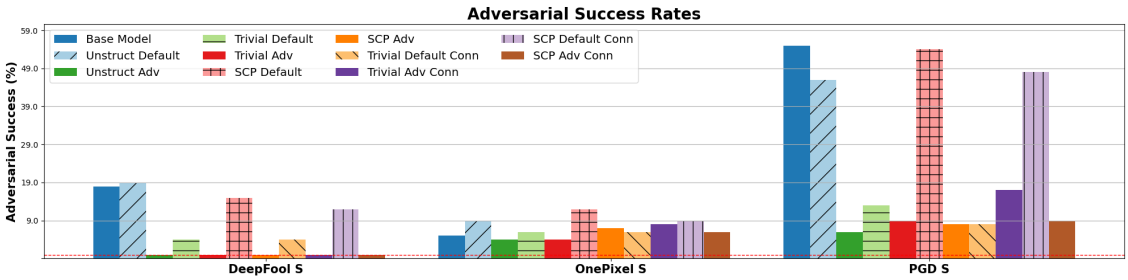


Figure 36: Strong Adversarial Attacks on ResNet18 Models

Table 24 was limited to $l_2 = 7,0$. The pattern-pruned models exhibit a slight dominance. Unlike the results of the LeNet models, there is a lower overflow rate but still a slight vulnerability toward PGD attacks. The parameterization of the adversarial training on the ResNet models is identical to the LeNet models. Information about the parameterization can be found in Appendix A.4.
Table 25 was limited to $l_2 = 5,0$. A slight reduction of the l_2 threshold was applied to identify anomalies, but no particular issues were found. A larger proportion of the top-2 ratings among the pattern-pruned models indicates an overall higher robustness.
Table 26 was limited to $l_2 = 8,0$. Once again, the dominance of the pattern-pruned models is visible. In these experimental series, the pattern-pruned models generally outperform the base model.
Compared to the test runs of the LeNet model, the adversarial-trained models exhibit the lowest overflow rate and prove to be more robust across almost all test runs. In the case of severe attacks, the adversarial models show a clear dominance compared to the default-trained models. As Ma et al. mentioned in their work, the filtering effects of pattern pruning reach their maximum with a larger number of convolutional layers [5, p. 5]. As described in Chapter 3.2.3, the ELoG filter is formed through multiple SCP interpolations. These interpolations correspond to the convolutional layers. The increased adversarial robustness of the pattern-pruned models, especially the adversarial-trained models, can be attributed to having a larger percentage of parameters that can be influenced by ADMM pattern pruning. The proportion of LeNet parameters influenced by optimization comprises only $0,8\%$ of the total parameters, while the proportion for the ResNet18 architecture is $74,94\%$. Thus, the optimization has a much bigger impact on the ResNet models, making the different results more understandable.
To conclude this evaluation, a final simplified measurement of adversarial attacks is presented in Table 15. The uniqueness of these attacks lies in the fact that they are adaptive attacks. Adaptive attacks are considered to be very strong attacks that have been able to break many stated adversarial robust models. Adaptive attacks approximate the gradients of a model and enable the breaking of adversarial defenses that use gradient masking/obfuscation (see chapter 2.2) as a defense strategy [16]. The attacks and parameterizations were adopted from the publication "SymDNN: Simple Effective Adversarial Robustness for Embedded Systems" by Dey et al. [44, p. 3604], which can be found in their repository. Their work focuses similarly on adversarial attacks for testing their model.

Attacks ($l_2 = 20$)	B	U-D	U-A	T-D	T-A	S-D	S-A
APGD_CE	0%	1%	3%	0%	3%	0%	2%
APGD_DLR	0%	1%	2%	0%	2%	0%	2%
BPDA	3%	5%	4%	1%	3%	2%	3%
CW	1%	0%	6%	0%	5%	0%	7%
DIFGSM	2%	3%	4%	1%	3%	2%	3%
EOTPGD	2%	4%	4%	1%	4%	4%	3%
Jitter	1%	3%	4%	0%	4%	0%	3%
MIFGSM	2%	4%	5%	1%	4%	4%	3%
RFGSM	2%	4%	4%	1%	4%	4%	3%
Square	0%	0%	0%	0%	2%	0%	0%
APGDT	25%	24%	0%	8%	0%	25%	0%
APGD_CE	28%	25%	1%	10%	0%	25%	0%
APGD_DLR	21%	23%	0%	9%	0%	23%	0%
AutoAttack	28%	25%	0%	10%	0%	25%	0%
BPDA	40%	37%	9%	38%	19%	39%	20%
CW	2%	2%	0%	1%	0%	5%	0%
DIFGSM	55%	52%	17%	49%	24%	56%	23%
EOTPGD	56%	52%	20%	49%	25%	56%	28%
Jitter	51%	48%	8%	43%	13%	53%	16%
MIFGSM	56%	52%	17%	49%	31%	56%	28%
RFGSM	56%	52%	17%	49%	28%	56%	30%
Square	3%	5%	0%	3%	0%	4%	0%
TIFGSM	50%	49%	8%	35%	5%	53%	8%

Table 15: Adaptive Adversarial Attacks on LeNet- & ResNet18 Models

The LeNet base model shows increased resistance to adaptive attacks. However, in the LeNet case the trivial-pruned model outperforms all other models in Table 15. Following this, the SCP model achieves the third-highest robustness.
Among the ResNet models, the adversarial-trained variants dominate. Especially unstructured pruning stands out from the other models. The trivial- and SCP-pruned models are the next to be ranked. Both unstructured and pattern-pruned models exhibit higher robustness than the base model. In these tests, pattern pruning exhibits increased robustness. Adversarial-trained models stand out for withstanding adaptive attacks the best on ImageNet.

To summarize the adversarial evaluation, I distribute points according to the colors used to highlight the best-performing models. The best results in green are assigned 3 points, the second-best results in yellow are given 2 points, and the third-best results receive 1 point. In Tables 16 & 17, the models are ranked based on the explained benchmarking definitions.

	B	U:D	U:A	T:D	T:A	S:D	S:A
Soft Attacks	11	8	9	7	7	12	8
Medium Attacks	3	4	3	11	0	7	3
Strong Attacks	5	5	3	6	2	10	0
Result	19	17	15	24	9	29	11
Adaptive Attacks	22	12	4	30	6	19	11
Result (+ Adaptive Atk)	41	29	19	54	15	48	22

Table 16: Ranking of LeNet Models on Adversarial Tests

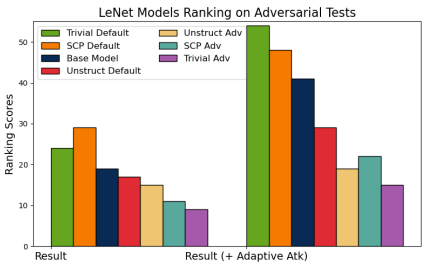


Figure 37: Results of LeNet Models on Adversarial Tests

	B	U:D	U:A	T:D	T:A	S:D	S:A
Soft Attacks	5	6	5	8	9	8	8
Medium Attacks	3	3	5	8	5	3	8
Strong Attacks	2	0	9	3	7	1	4
Result	10	9	19	19	21	12	20
Adaptive Attacks	4	3	37	12	31	2	28
Result (+ Adaptive Atk)	14	12	56	31	52	14	48

Table 17: Ranking of ResNet Models on Adversarial Tests

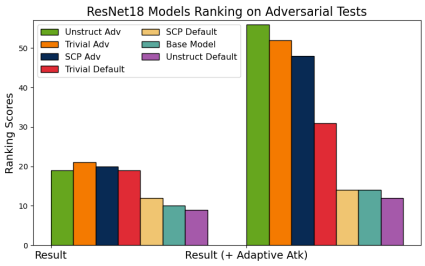


Figure 38: Results of ResNet Models on Adversarial Tests

In the respective benchmark evaluations 16 & 17, pattern pruning achieves a higher score than the base model. In the benchmark ranking of the ResNet architecture, the adversarial-trained models exhibit a significantly higher score than the base model. When the adversarial-trained models are excluded, pattern pruning still achieves a higher and, in one case, an equal score compared to the non-optimized model. For this reason, an increase in filter capacity can be concluded, as higher or equal adversarial robustness is measured despite the lower number of parameters.

5.3. Visual Interpretation

Following are illustrations generated with Grad-CAM++ in Fig. 39 and Score-CAM in Fig. 40 on samples that were altered through adversarial attacks and successfully tricked the base model but not the optimized models. A more focused highlighting on areas that are essential for humans to correctly classify the object can be observed. In the MNIST samples shown in Fig. 40, there is a notable increase in the highlighting of important regions, with no selection of areas outside the object edges. Both class activation maps refer to the ground truth labels and show the most influential regions toward the classification within the model's last convolutional layer.

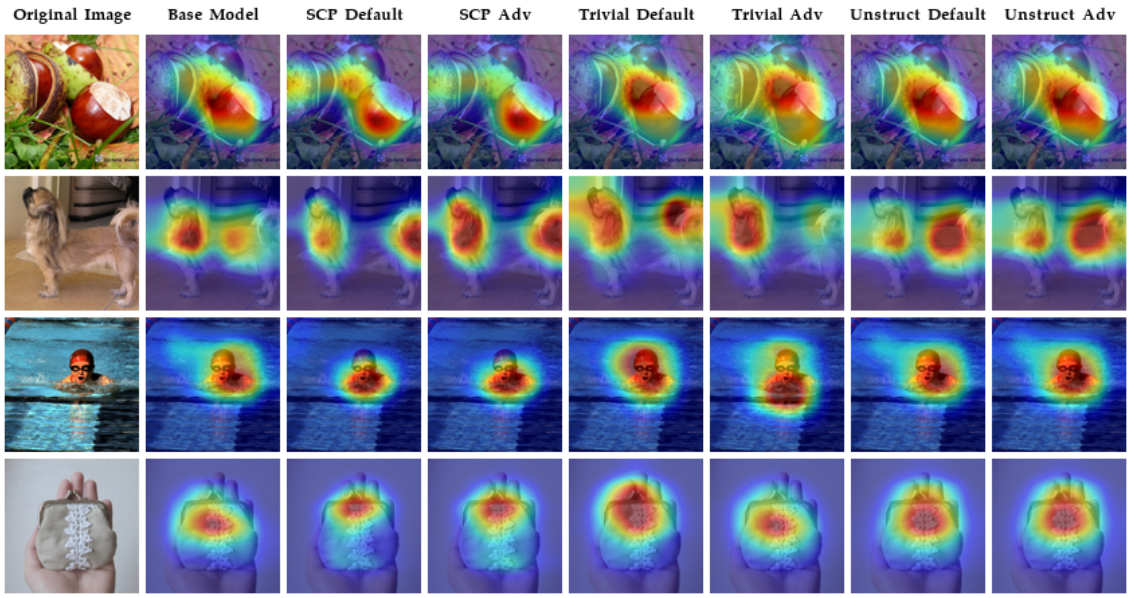


Figure 39: Grad-CAM++ of ImageNet Samples on ResNet18 Models

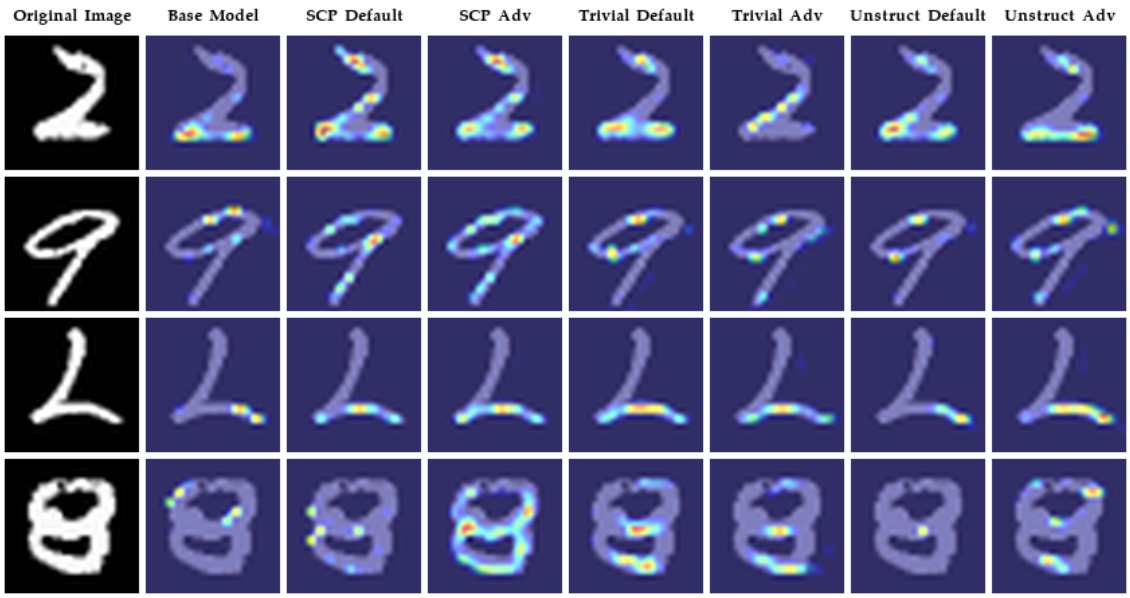


Figure 40: Score-CAM of MNIST Samples on LeNet Models

The following Figure 41 shows a saliency map for LeNet models. The saliency map is linked to the output class and represents the raw gradients toward the targeted label. At each pixel position, the maximum value of the respective channel gradients is assigned. It is important to note that the saliency map exhibits the most vulnerable and impactful regions of the image.

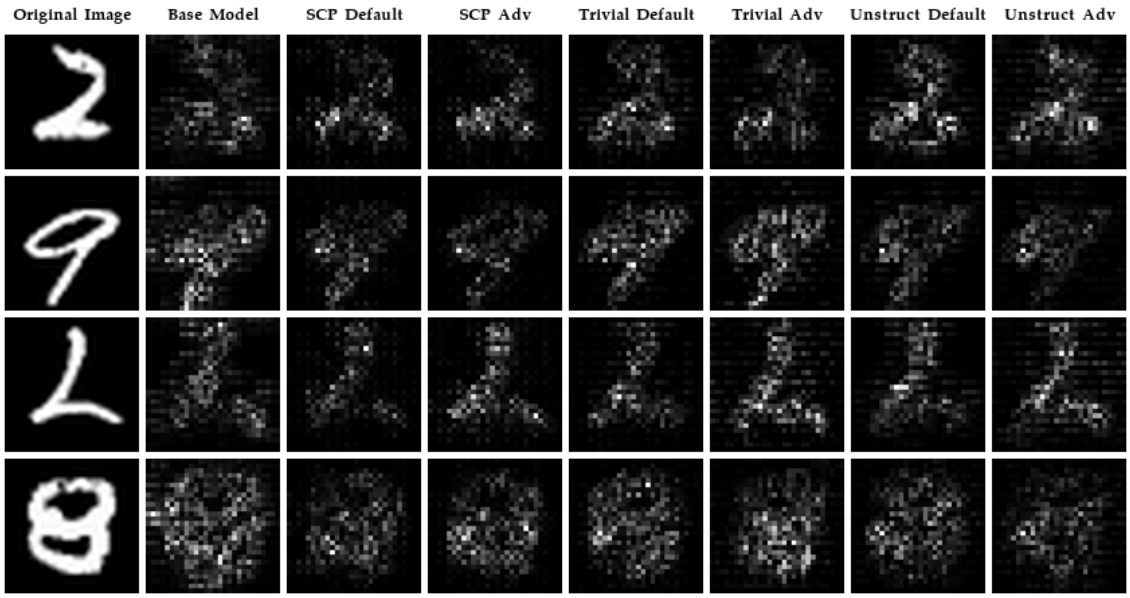


Figure 41: Saliency Map of MNIST Samples on LeNet Models

It can be observed that the areas of influence in the pattern-optimized models are sharper. The base model shows high gradients in regions that do not belong to the object in focus. The base model also relies more on redundant image information for its output. This can be compared to obsessive-compulsive disorder, in which the affected person cannot perform a task because they see chairs out of the corner of their eye that are not equally spaced. Similarly, the CNN takes unnecessary information into account and incorporates it into the classification. Pattern pruning produces sharper object outlines and lower overall gradient intensities in the images. This is a good sign, as the model is less affected by perturbations.

As the final visual interpretation, the feature maps are presented. I have decided to limit this to the LeNet models because the ResNet models include too many feature maps to be presented clearly. A higher-level layer was specifically chosen because it extracts more fundamental, coarse-grained features. These features are less abstracted than those from deeper layers and are easier to interpret. The feature maps represent the activation maps, which are also used for CIC. Dark areas inside the images correspond to low feature values, while bright areas correspond to higher values. Bright areas represent features extracted from the image. Figure 42 shows the feature maps of the first convolutional layer inside the various models.

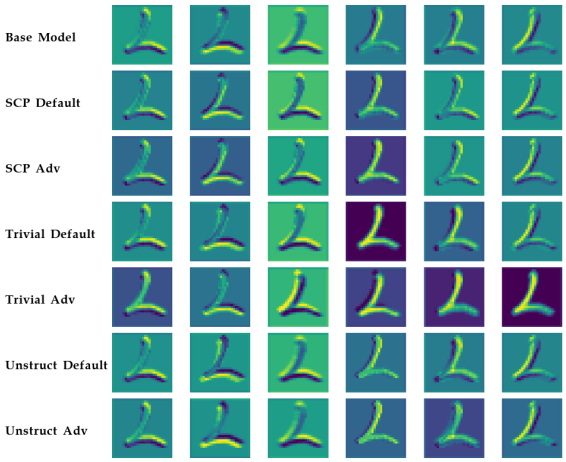


Figure 42: Feature Maps of an MNIST Sample on LeNet Models

It becomes clearly visible that pattern-pruned models better extract the classification object. Background areas are predominantly represented with low values and have a reduced disturbing influence on the model.
Additional images for these visualization methods can be found in Appendix A.5.

5.4. CIC Evaluation

In the following, the custom-derived measurement method based on CIC is presented and discussed. CIC is a fine-grained, layer-wise measurement of convolutional layer performance.
The ResNet data were collected using 1000 samples from the ImageNet dataset (approximately 2% of the ImageNet dataset), with each class represented by one sample. This approach was necessary because my test setup had limited computational power, making it impossible to conduct more extensive tests within the time constraints of this work. The LeNet data were collected using 8000 test samples from the MNIST dataset (80% of the MNIST dataset), with each class represented by 800 samples.
For each model, two test runs were performed. The first run is based on the default dataset, while the second run is applied to the adversarial mixed dataset. Appendix A.6 contains measurement data related to the figures and the adversarial parameters.

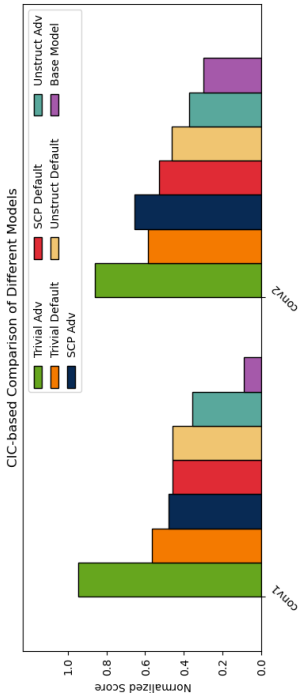


Figure 43: CIC Measurements on LeNet Models

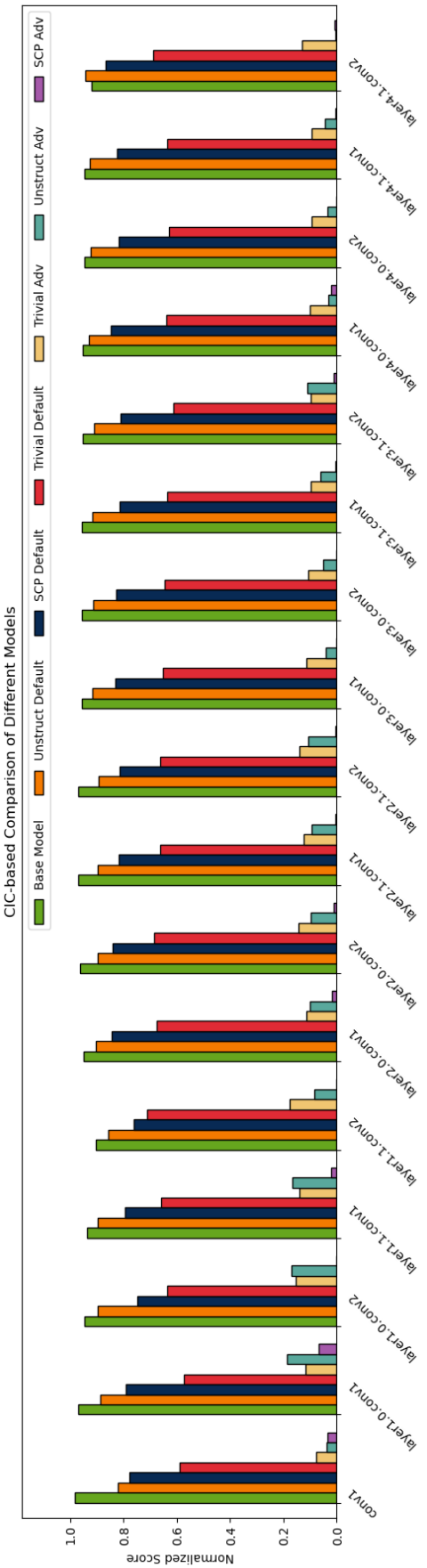


Figure 44: CIC Measurements on ResNet18 Models

In the CIC test of the LeNet models in Fig. 43, the convolutional layers of the base model show the lowest scores compared to the other models. The pattern-pruned models achieve the best results in both the adversarial and default variants.
Figure 44 shows superior results for the base model compared to the other models. In each convolutional layer, the pruned models perform worse here than the base model. This can be explained by the fact that the generated masks of the base model are continuous and contain more information. Unstructured-pruned models have varied value distribution structures in the convolutional layers, allowing them to better compete with the base model. Pattern-pruned models exhibit exactly four different filter kernel structures, with limited activation map variety. The CIC graphic is closely tied to the accuracy measurements. In Table 6, unstructured pruning achieves higher accuracy than the base model. This is evident in the last layer in Figure 44. For the final classification, the last convolutional layer is the most important. It can be concluded that this layer is crucial for the unstructured-pruned model's higher accuracy compared to the base model.

Identical representations are presented for PGD-generated adversarial samples in Figs. 45 and Fig. 46.

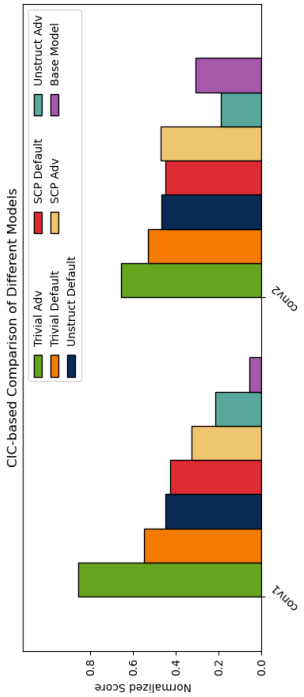


Figure 45: CIC Measurements With Adversarial Samples on LeNet Models

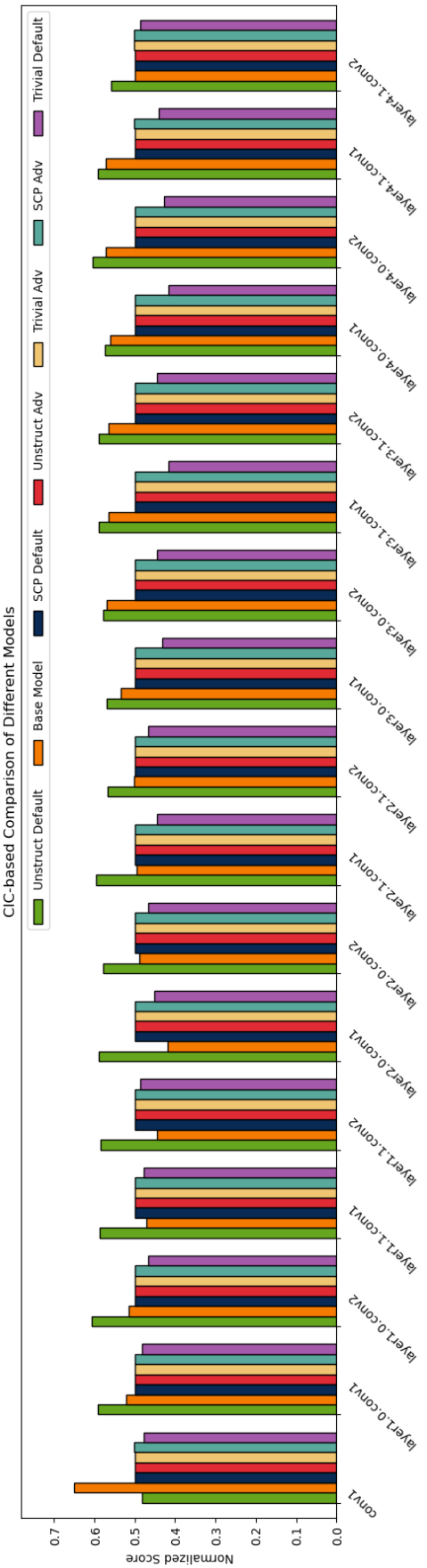


Figure 46: CIC Measurements With Adversarial Samples on ResNet18 Models

Among the LeNet models in Fig. 45, the trivial-pattern-pruned models perform the best. The SCP-optimized variant also shows better results than the base model.
For the ResNet models in Fig. 46, the unstructured-pruned model achieves the best results. The majority of the pattern-pruned models do not match the performance of the base model in 11 out of 17 layers, but they show a slightly higher score in the last and most important convolutional layer.
The results of the ResNet models can be questioned since only 2% of the entire dataset was used, whereas the LeNet tests were conducted on 80% of the respective dataset. Other potential causes for the differences could be the varying network architectures or the significant differences in the amount of information per sample between the datasets.

In the LeNet tests, the pattern-pruned models can clearly be identified as superior to the base model. The evaluation for the ResNet tests is more complex. On the normal dataset, pattern pruning performs worse than the base model, but it shows a generally smaller gap to the base model in the adversarial tests. Additionally, the distribution of scores across the layers allows for considerable interpretation. So far, an increasing filter capacity is evident under stronger adversarial attacks.

5.5. Summary of Findings

In this chapter, various methods and techniques were applied to investigate how pattern pruning affects filter capacity and adversarial robustness.
The results of the conducted measurements clearly demonstrate an increase in filter capacity and robustness against adversarial attacks for LeNet, while the results for ResNet18 are less definitive. Pattern pruning on the LeNet model achieves superior results in all measurements compared to the base model. In half of the tests, the pattern-pruned ResNet models fall short of the base model. In the case of adversarial attacks, pattern pruning's strength over the base model is evident. The results of the CIC test indicate a tendency toward increased filter capacity in the presence of increasing perturbations.

It is difficult to make a general statement for both models, as they differ significantly in their architecture. The ResNet models have a pruning rate that is $109x$ times larger than that of the LeNet models. Additionally, the LeNet model includes several fully connected layers, which can further interpret information. In contrast, ResNet has only a single fully connected layer as its final layer, lacking the final processing stage present in the LeNet model. There is also the possibility that the small number of four patterns was set too low to achieve a clear improvement over the base model.
In the experiments, pattern pruning was applied to all convolutional layers without considering the various possible variations. Specific layers could be selected instead. This would reduce the pruning rate but potentially enhance all previous measured attributes. With the current ResNet18 pruning rate of $52,24\%$, only a slight decrease in benchmark values is observed. If a linear relationship exists between the pruning rate and the benchmark values, a smaller pruning rate would inevitably lead to a clear dominance of pattern-pruned models over the base model.

Considering the points mentioned above, an increased filter capacity as well as an enhanced adversarial robustness can be observed in both models.

6. Conclusion

This thesis investigated whether pattern pruning can enhance the filter capacity and robustness of convolutional neural networks against perturbations generated by various methods, with a greater focus on adversarial attacks. This is very important for applications in the field of automotive, where there is a big interest in ensuring functional safety. Supported by the state-of-the-art optimization algorithm called the alternating direction method of multipliers, the pruning process is controlled and regularized to achieve the best possible solution to the combinatorial non-convex optimization problem of pruning. The experiments demonstrated the advantages given by pattern pruning. Not only is model complexity reduced, but also an increase in adversarial robustness could also be proven. This led to the insight that reducing the model's complexity does not it the model more vulnerable to perturbations. Pattern pruning's second and most important advantage is the ability to accelerate the CNN on specific hardware. As it is not in the scope of this thesis, the test environment was set up to evaluate pattern pruning as it would be adopted for hardware in further research. By removing weights in a structured manner, the model gains the ability to be hardware accelerated, which is particularly beneficial for embedded systems. In the automotive sector, the model could process data faster with less influence from perturbations. This not only enables safe real-time applications but also enhances energy efficiency.
In conclusion, pattern pruning, utilized by ADMM, is a promising optimization method for improving efficiency, filter capacity and robustness against adversarial perturbations and random perturbations. Pattern pruning is also a promising compromise between the precision efficiency of unstructured pruning and the computation efficiency of structured pruning. This technique could play a crucial role in the development of safe autonomous driving applications.
Further research could focus on testing the algorithm on different models with varying configurations to further consolidate the statements made in this thesis.

List of figures

List of tables

Literature references

[1] ITM, "Protect iot applications from adversarial evasion attacks," https://www.itm-p.com/protect-iot-applications-from-adversarial-evasion-attacks, 2024, accessed: 2024-06-02.

[2] S. H. Silva and P. Najafirad, "Opportunities and challenges in deep learning adversarial robustness: A survey," *CoRR*, vol. abs/2007.00753, 2020. [Online]. Available: https://arxiv.org/abs/2007.00753

[3] B. Li, "Yolo detection output for a real stop sign with adversarial stickers (phone camera)," https://www.researchgate.net/figure/YOLO-detection-output-for-a-real-stop-sign-with-adversarial-stickers-phone-camera_fig1_321994706, 2024, accessed: 2024-06-02.

[4] T. X. Bob Yirka, "Adversarial patch ai," https://techxplore.com/news/2019-04-adversarial-patch-ai.html, 2019, accessed: 2024-06-02.

[5] X. Ma, F. Guo, W. Niu, X. Lin, J. Tang, K. Ma, B. Ren, and Y. Wang, "PCONV: the missing but desirable sparsity in DNN weight pruning for real-time execution on mobile devices," *CoRR*, vol. abs/1909.05073, 2019. [Online]. Available: http://arxiv.org/abs/1909.05073

[6] S. Boyd, "The alternating direction method of multipliers," https://stanford.edu/~boyd/admm.html#:~:text=The%20alternating%20direction%20method%20of,are%20then%20easier%20to%20handle, 2021, accessed: 2024-06-02.

[7] R. R. Selvaraju, A. Das, R. Vedantam, M. Cogswell, D. Parikh, and D. Batra, "Grad-cam: Why did you say that? visual explanations from deep networks via gradient-based localization," *CoRR*, vol. abs/1610.02391, 2016. [Online]. Available: http://arxiv.org/abs/1610.02391

[8] A. Chattopadhyay, A. Sarkar, P. Howlader, and V. N. Balasubramanian, "Grad-cam++: Generalized gradient-based visual explanations for deep convolutional networks," *CoRR*, vol. abs/1710.11063, 2017. [Online]. Available: http://arxiv.org/abs/1710.11063

[9] H. Wang, M. Du, F. Yang, and Z. Zhang, "Score-cam: Improved visual explanations via score-weighted class activation mapping," *CoRR*, vol. abs/1910.01279, 2019. [Online]. Available: http://arxiv.org/abs/1910.01279

[10] A. Torralba, P. Isola, and W. Freeman, *Foundations of Computer Vision*, ser. Adaptive Computation and Machine Learning series. MIT Press, 2024. [Online]. Available: https://books.google.de/books?id=xtC_EAAAQBAJ

[11] S. Moosavi-Dezfooli, A. Fawzi, and P. Frossard, "Deepfool: a simple and accurate method to fool deep neural networks," *CoRR*, vol. abs/1511.04599, 2015. [Online]. Available: http://arxiv.org/abs/1511.04599

[12] N. Papernot, P. D. McDaniel, S. Jha, M. Fredrikson, Z. B. Celik, and A. Swami, "The limitations of deep learning in adversarial settings," *CoRR*, vol. abs/1511.07528, 2015. [Online]. Available: http://arxiv.org/abs/1511.07528

[13] M. T. Heath, *Scientific Computing: An Introductory Survey*, revised second edition ed., ser. Classics in Applied Mathematics. Philadelphia: Society for Industrial and Applied Mathematics (SIAM), 2018, vol. 80.

[14] W. Guo, Y. Lou, J. Qin, and M. Yan, "A novel regularization based on the error function for sparse recovery," *Journal of Scientific Computing*, vol. 87, no. 1, Mar. 2021. [Online]. Available: http://dx.doi.org/10.1007/s10915-021-01443-w

[15] R. Tibshirani, Y. Lu, A. Harley, and R. Wang, "10-725/36-725: Convex optimization fall 2018 lecture notes," 2018, lecture notes, LaTeX template courtesy of UC Berkeley EECS dept. [Online]. Available: https://www.math.cmu.edu/~ryantibs/convexopt-F18/lectures/02-Convex.pdf

[16] A. Athalye, N. Carlini, and D. A. Wagner, "Obfuscated gradients give a false sense of security: Circumventing defenses to adversarial examples," *CoRR*, vol. abs/1802.00420, 2018. [Online]. Available: http://arxiv.org/abs/1802.00420

[17] A. Madry, A. Makelov, L. Schmidt, D. Tsipras, and A. Vladu, "Towards deep learning models resistant to adversarial attacks," 2019.

[18] B. Zhang, T. Wang, S. Xu, and D. Doermann, *Neural Networks with Model Compression*, 1st ed., ser. Computational Intelligence Methods and Applications. Springer Singapore, 2024.

[19] W. Niu, X. Ma, S. Lin, S. Wang, X. Qian, X. Lin, Y. Wang, and B. Ren, "Patdnn: Achieving real-time DNN execution on mobile devices with pattern-based weight pruning," *CoRR*, vol. abs/2001.00138, 2020. [Online]. Available: http://arxiv.org/abs/2001.00138

[20] X. Ma, W. Niu, T. Zhang, S. Liu, F. Guo, S. Lin, H. Li, X. Chen, J. Tang, K. Ma, B. Ren, and Y. Wang, "An image enhancing pattern-based sparsity for real-time inference on mobile devices," *CoRR*, vol. abs/2001.07710, 2020. [Online]. Available: https://arxiv.org/abs/2001.07710

[21] S. P. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein, "Distributed optimization and statistical learning via the alternating direction method of multipliers." *Found. Trends Mach. Learn.*, vol. 3, no. 1, pp. 1–122, 2011. [Online]. Available: http://dblp.uni-trier.de/db/journals/ftml/ftml3.html#BoydPCPE11

[22] T. Zhang, S. Ye, K. Zhang, J. Tang, W. Wen, M. Fardad, and Y. Wang, "A systematic DNN weight pruning framework using alternating direction method of multipliers," *CoRR*, vol. abs/1804.03294, 2018. [Online]. Available: http://arxiv.org/abs/1804.03294

[23] S. Ye, X. Feng, T. Zhang, X. Ma, S. Lin, Z. Li, K. Xu, W. Wen, S. Liu, J. Tang, M. Fardad, X. Lin, Y. Liu, and Y. Wang, "Progressive DNN compression: A key to achieve ultra-high weight pruning and quantization rates using ADMM," *CoRR*, vol. abs/1903.09769, 2019. [Online]. Available: http://arxiv.org/abs/1903.09769

[24] A. Ren, T. Zhang, S. Ye, J. Li, W. Xu, X. Qian, X. Lin, and Y. Wang, "ADMM-NN: an algorithm-hardware co-design framework of dnns using alternating direction method of multipliers," *CoRR*, vol. abs/1812.11677, 2018, GitHub Repository. [Online]. Available: http://arxiv.org/abs/1812.11677

[25] M. Lin, Y. Li, Y. Zhang, B. Chen, F. Chao, M. Wang, S. Li, J. Yang, and R. Ji, "1×n block pattern for network sparsity," *CoRR*, vol. abs/2105.14713, 2021. [Online]. Available: https://arxiv.org/abs/2105.14713

[26] J. Peters, *Foundations of Computer Vision: Computational Geometry, Visual Image Structures and Object Shape Detection*, ser. Intelligent Systems Reference Library. Springer International Publishing, 2017. [Online]. Available: https://books.google.de/books?id=CtxmDgAAQBAJ

[27] https://stackoverflow.com/users/1336939/synack, "Simplest way to generate a 1d gaussian kernel," Forum, uRL:https://stackoverflow.com/a/78185217 (version: 2023-04-05). [Online]. Available: https://stackoverflow.com/a/78185217

[28] Wikipedia contributors, "Gaussian blur," https://en.wikipedia.org/wiki/Gaussian_blur, 2023, accessed: 2024-07-15.

[29] ——, "Finite difference," https://en.wikipedia.org/wiki/Finite_difference, 2023, accessed: 2024-07-15.

[30] T. Arens, F. Hettlich, C. Karpfinger, U. Kockelkorn, K. Lichtenegger, and H. Stachel, *Mathematik*, 4th ed., ser. Springer eBook Collection. Berlin: Springer Spektrum, 2018. [Online]. Available: https://doi.org/10.1007/978-3-662-56741-8

[31] P. D.-I. R. Krämer, "Vorlesungsskript signale und systeme (2. semester)," Lecture notes, 2018, version 15.03.2018.

[32] B. L. Evans, "Frequency domain analysis," https://users.ece.utexas.edu/~bevans/courses/ee381k/lectures/03_Frequency/lecture3/index.html, 2023, accessed: 2024-07-16.

[33] Wikipedia contributors, "Hadamard product (matrices)," https://en.wikipedia.org/wiki/Hadamard_product_(matrices), 2023, accessed: 2024-07-16.

[34] ——, "Linear system," https://en.wikipedia.org/wiki/Linear_system, 2023, accessed: 2024-07-16.

[35] W. contributors, "Convolution theorem," https://en.wikipedia.org/wiki/Convolution_theorem, 2024, accessed: 2024-08-13. [Online]. Available: https://en.wikipedia.org/wiki/Convolution_theorem

[36] Wikipedia contributors, "Convolution of probability distributions," https://en.wikipedia.org/wiki/Convolution_of_probability_distributions, 2024, accessed: 2024-07-28.

[37] OpenAI, "Chatgpt: Generative Pre-trained Transformer (GPT-4)," openAI. [Online]. Available: https://www.openai.com/chatgpt

[38] F.-G. Fernandez, "Torchcam: class activation explorer," https://github.com/frgfm/torch-cam, March 2020.

[39] Y. LeCun, C. Cortes, and C. J. Burges, "The mnist database of handwritten digits," 1998, accessed: 2024-07-21. [Online]. Available: http://yann.lecun.com/exdb/mnist/

[40] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *CoRR*, vol. abs/1512.03385, 2015. [Online]. Available: http://arxiv.org/abs/1512.03385

[41] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, "ImageNet Large Scale Visual Recognition Challenge," *International Journal of Computer Vision (IJCV)*, vol. 115, no. 3, pp. 211–252, 2015.

[42] H. Kim, "Torchattacks: A pytorch repository for adversarial attacks," *arXiv preprint arXiv:2010.01950*, 2020.

[43] S. Ye, K. Xu, S. Liu, H. Cheng, J. Lambrechts, H. Zhang, A. Zhou, K. Ma, Y. Wang, and X. Lin, "Second rethinking of network pruning in the adversarial setting," *CoRR*, vol. abs/1903.12561, 2019. [Online]. Available: http://arxiv.org/abs/1903.12561

[44] S. Dey, P. Dasgupta, and P. P. Chakrabarti, "Symdnn: Simple effective adversarial robustness for embedded systems," in *IEEE Conference on Computer Vision and Pattern Recognition Workshop (CVPRW)*, 2022.

A. Appendix

A.1. Filter Kernel Structures and Polarity Distributions Additional Information

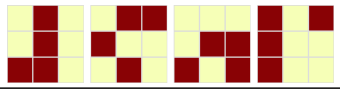
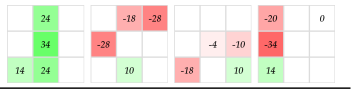
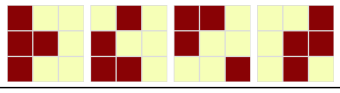
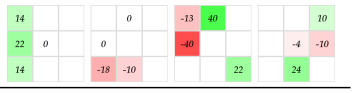
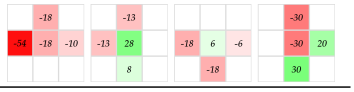
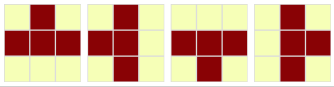
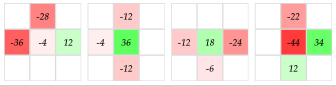
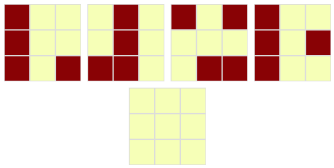
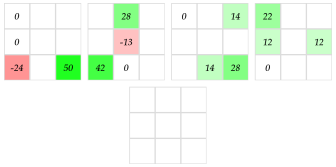
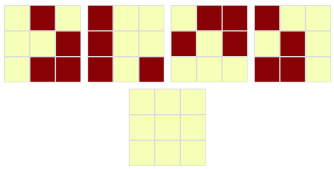
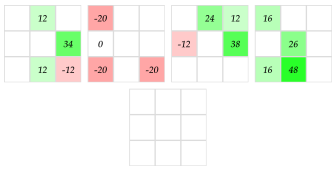
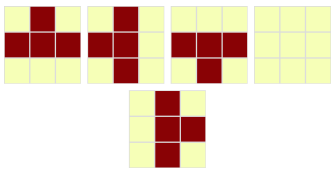
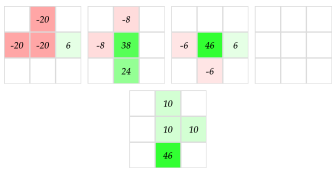
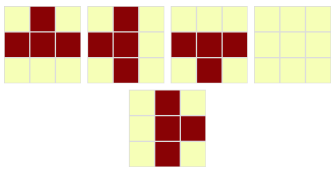
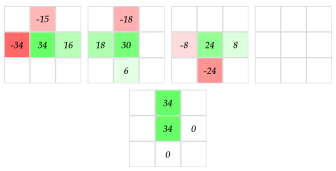
# LeNet Models	Pattern	Polarity
Trivial Default		
Trivial Adv		
SCP Default		
SCP Adv		
Trivial Default Conn		
Trivial Adv Conn		
SCP Default Conn		
SCP Adv Conn		

Table 18: Filter Kernel Structures and Polarity Distributions After Pattern Pruning on LeNet Models

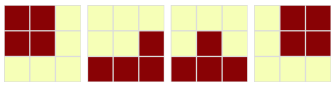
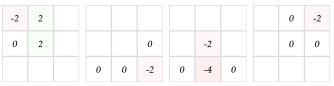
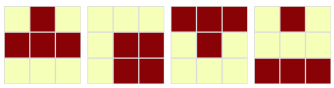
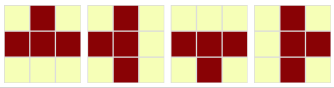
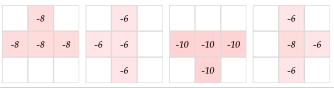
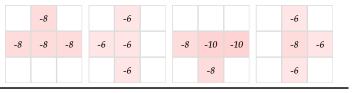
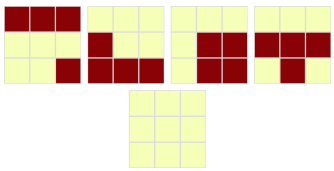
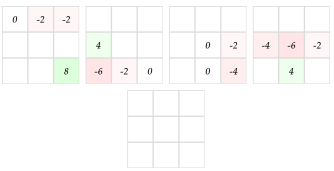
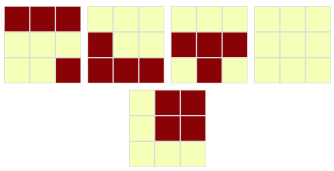
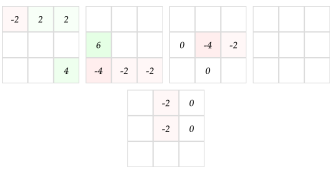
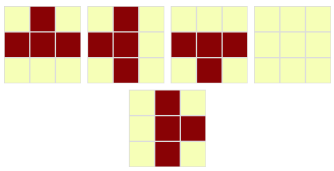
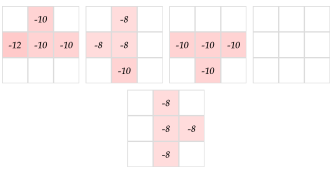
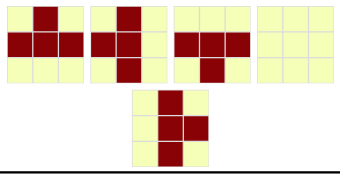
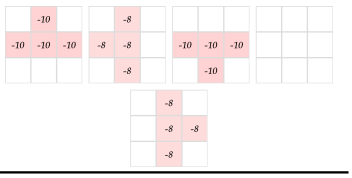
# ResNet18 Models	Pattern	Polarity
Trivial Default		
Trivial Adv		
SCP Default		
SCP Adv		
Trivial Default Conn		
Trivial Adv Conn		
SCP Default Conn		
SCP Adv Conn		

Table 19: Filter Kernel Structures and Polarity Distributions After Pattern Pruning on ResNet18 Models

A.2. ADMM Integration of Connectivity Pruning Additional Information

Set the definition of connectivity pruning [19, p. 913]:

$$S_i^{connectivity} := \{\mathbf{W}_i \mid \text{card}(\{\mathbf{W}_i\}) \leq \alpha_i\}$$

$$\alpha_i : \text{threshold value of filter kernels to be pruned}$$

Adding connectivity pruning extends the optimization problem as shown by Niu et al. [19, p. 913]:

$$\min_{\{\mathbf{W}_i\}} f\left(\{\mathbf{W}_i\}\right) + \sum_{i=1}^{N} \frac{\rho_i}{2} \|\mathbf{W}_i - \mathbf{Z}_i + \mathbf{U}_i\|_F^2 + \sum_{i=1}^{N} \frac{\rho_i}{2} \|\mathbf{W}_i - \mathbf{Y}_i + \mathbf{V}_i\|_F^2$$

As both pruning algorithms operate on the same weights, the equation can be simplified under the assumptions $\mathbf{Z}_i = \mathbf{Y}_i$ and $\mathbf{U}_i = \mathbf{V}_i$ and the equation becomes:

$$\sum_{i=1}^{N} \frac{\rho_i}{2} \|\mathbf{W}_i - \mathbf{Z}_i + \mathbf{U}_i\|_F^2 + \sum_{i=1}^{N} \frac{\rho_i}{2} \|\mathbf{W}_i - \mathbf{Z}_i + \mathbf{U}_i\|_F^2$$

Since both terms are identical, we can combine them:

$$2 \sum_{i=1}^{N} \frac{\rho_i}{2} \|\mathbf{W}_i - \mathbf{Z}_i + \mathbf{U}_i\|_F^2$$

This expression simplifies to:

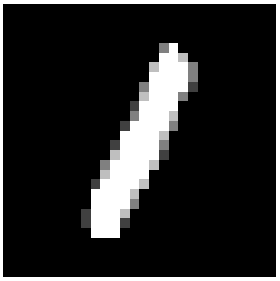
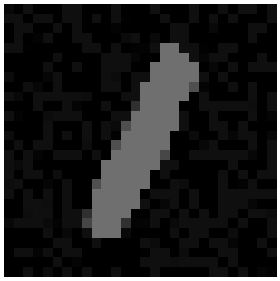
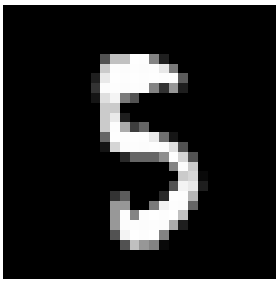
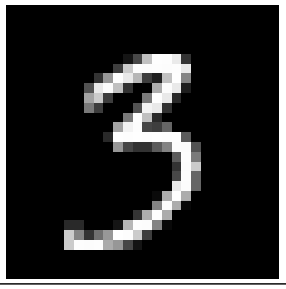
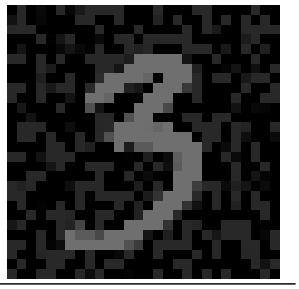
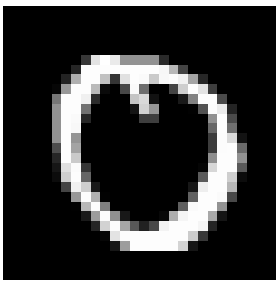
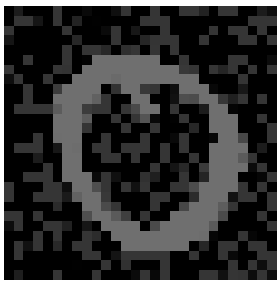
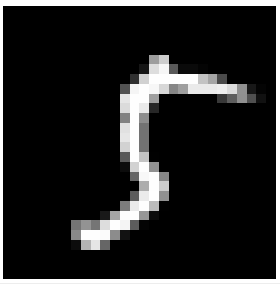
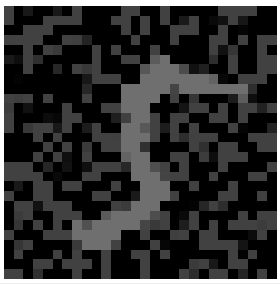
$$\sum_{i=1}^{N} \rho_i \|\mathbf{W}_i - \mathbf{Z}_i + \mathbf{U}_i\|_F^2$$

The final form of the equation is:

$$\min_{\{\mathbf{W}_i\}} f\left(\{\mathbf{W}_i\}\right) + \sum_{i=1}^{N} \rho_i \|\mathbf{W}_i - \mathbf{Z}_i + \mathbf{U}_i\|_F^2$$

This is not a proven change in the equations to combine both subproblems into a single subproblem. These changes were made to address both constraints pattern pruning and connectivity pruning in a single subproblem within the ADMM algorithm used in this thesis.

A.3. Noise Overlay Measurement Additional Information

# Noise Ratio	Original	Noisy Image
MNIST $r = 0.2$		
MNIST $r = 0.36$		
MNIST $r = 0.52$		
MNIST $r = 0.68$		
MNIST $r = 0.84$		

# Noise Ratio	Original	Noisy Image
ImageNet $r = 0.2$		
ImageNet $r = 0.36$		
ImageNet $r = 0.52$		
ImageNet $r = 0.68$		
ImageNet $r = 0.84$		

A.4. Adversarial Attacks Additional Information

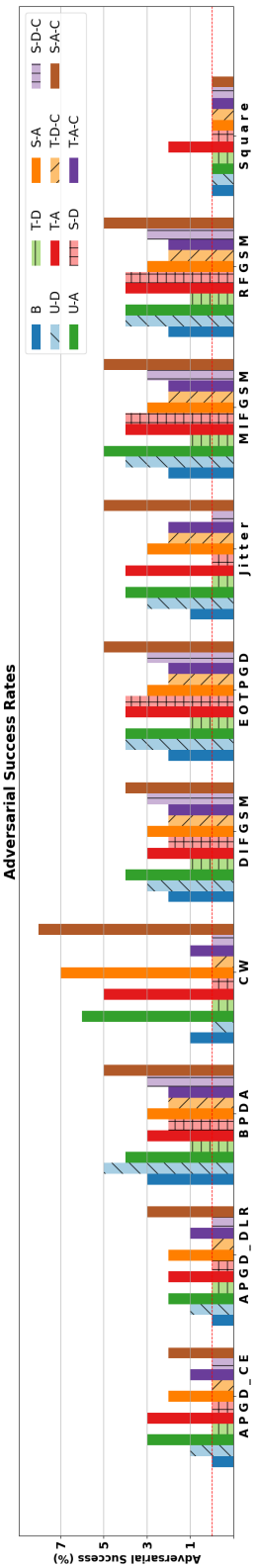


Figure 47: Adaptive Adversarial Attacks on LeNet Models

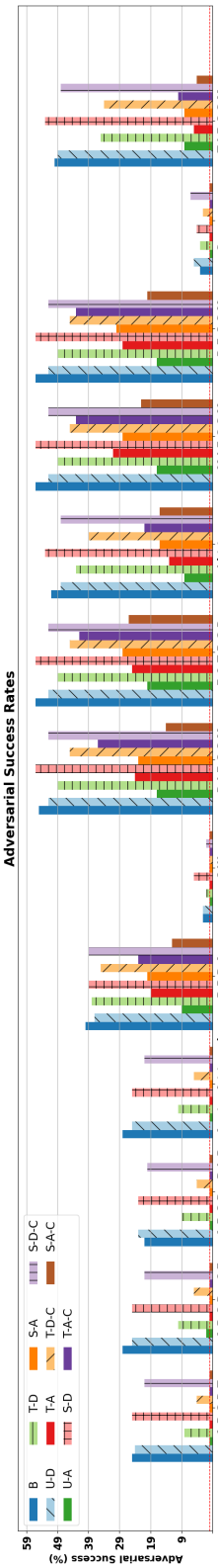


Figure 48: Adaptive Adversarial Attacks on ResNet18 Models

Table 20: Soft Adversarial Attacks Parameters on LeNet Models

Attack Type	**Initialization Parameters**
	l_2-norm: 1.4
DeepFool	steps: 50, overshoot: 0.2
PGD	eps: 0.0314, alpha: 0.0078, steps: 10, random_start: false
JSMA	theta: 1.0, gamma: 0.001
OnePixel	pixels: 1, steps: 10, popsize: 10, inf_batch: 128
SparseFool	steps: 1, lam: 3, overshoot: 0.02

Table 21: Medium Adversarial Attacks Parameters on LeNet Models

Attack Type	**Initialization Parameters**
	l_2-norm: 3.0
DeepFool	steps: 50, overshoot: 3.0
PGD	eps: 0.1414, alpha: 0.0157, steps: 20, random_start: false
JSMA	theta: 2.0, gamma: 0.01
OnePixel	pixels: 20, steps: 15, popsize: 10, inf_batch: 128
SparseFool	steps: 1, lam: 3, overshoot: 0.02

Table 22: Strong Adversarial Attacks Parameters on LeNet Models

Attack Type	**Initialization Parameters**
	l_2-norm: 4.5
DeepFool	steps: 50, overshoot: 4.5
PGD	eps: 0.4014, alpha: 0.0257, steps: 20, random_start: false
JSMA	theta: 2.0, gamma: 0.017
OnePixel	pixels: 50, steps: 15, popsize: 10, inf_batch: 128
SparseFool	steps: 1, lam: 3, overshoot: 0.02

Table 23: Adaptive Adversarial Attacks Parameters on LeNet Models

Attack Type	Initialization Parameters
	l_2-norm: 20
BPDA	epsilon: 0.03137, learning_rate: 0.5, max_iterations: 100
DIFGSM	eps: 0.03137, alpha: 0.00784, steps: 100, diversity_prob: 0.5, resize_rate: 0.9
MIFGSM	eps: 0.03137, alpha: 0.00784, steps: 100, decay: 0.1
RFGSM	eps: 0.03137, alpha: 0.00784, steps: 100
EOTPGD	eps: 0.03137, alpha: 0.00784, steps: 100, eot_iter: 2
APGD_CE	eps: 0.03137, steps: 100, eot_iter: 1, n_restarts: 1, loss: ce
APGD_DLR	eps: 0.03137, steps: 100, eot_iter: 1, n_restarts: 1, loss: dlr
Jitter	eps: 0.03137, alpha: 0.00784, steps: 40, scale: 10, std: 0.1, random_start: true
CW	c: 1, lr: 0.01, steps: 100, kappa: 0
Square	eps: 0.03137, n_queries: 5000, n_restarts: 1, loss: ce

Table 24: Soft Adversarial Attacks Parameters on ResNet18 Models

Attack Type	Initialization Parameters
	l_2-norm: 7
DeepFool	steps: 10, overshoot: 0.2
PGD	eps: 0.03137254901960784, alpha: 0.00784313725490196, steps: 10, random_start: False
OnePixel	pixels: 1, steps: 10, popsize: 10, inf_batch: 128

Table 25: Medium Adversarial Attacks Parameters on ResNet18 Models

Attack Type	Initialization Parameters
	l_2-norm: 5
DeepFool	steps: 10, overshoot: 1.5
PGD	eps: 1.0, alpha: 0.00784314, steps: 10, random_start: false
OnePixel	pixels: 30, steps: 10, popsize: 10, inf_batch: 128

Table 26: Strong Adversarial Attacks Parameters on ResNet18 Models

Attack Type	Initialization Parameters
	l_2-norm: 8
DeepFool	steps: 50, overshoot: 14.0
PGD	eps: 3.0, alpha: 0.00205, steps: 20, random_start: false
OnePixel	pixels: 45, steps: 15, popsize: 10, inf_batch: 128

Table 27: Adaptive Adversarial Attacks Parameters on ResNet18 Models

Attack Type	Initialization Parameters
	l_2-norm: 20
BPDA	epsilon: 0.03137, learning_rate: 0.5, max_iterations: 100
TIFGSM	eps: 0.03137, alpha: 0.00784, steps: 100, diversity_prob: 0.5
AutoAttack	eps: 0.03137, n_classes: 10, version: standard
DIFGSM	eps: 0.03137, alpha: 0.00784, steps: 100, diversity_prob: 0.5, resize_rate: 0.9
MIFGSM	eps: 0.03137, alpha: 0.00784, steps: 100, decay: 0.1
RFGSM	eps: 0.03137, alpha: 0.00784, steps: 100
EOTPGD	eps: 0.03137, alpha: 0.00784, steps: 100, eot_iter: 2
APGD_CE	eps: 0.03137, steps: 100, eot_iter: 1, n_restarts: 1, loss: ce
APGD_DLR	eps: 0.03137, steps: 100, eot_iter: 1, n_restarts: 1, loss: dlr
APGDT	eps: 0.03137, steps: 100, eot_iter: 1, n_restarts: 1
Jitter	eps: 0.03137, alpha: 0.00784, steps: 40, scale: 10, std: 0.1, random_start: true
CW	c: 1, lr: 0.01, steps: 100, kappa: 0
Square	eps: 0.03137, n_queries: 5000, n_restarts: 1, loss: ce

A.5. Visual Interpretation Additional Information

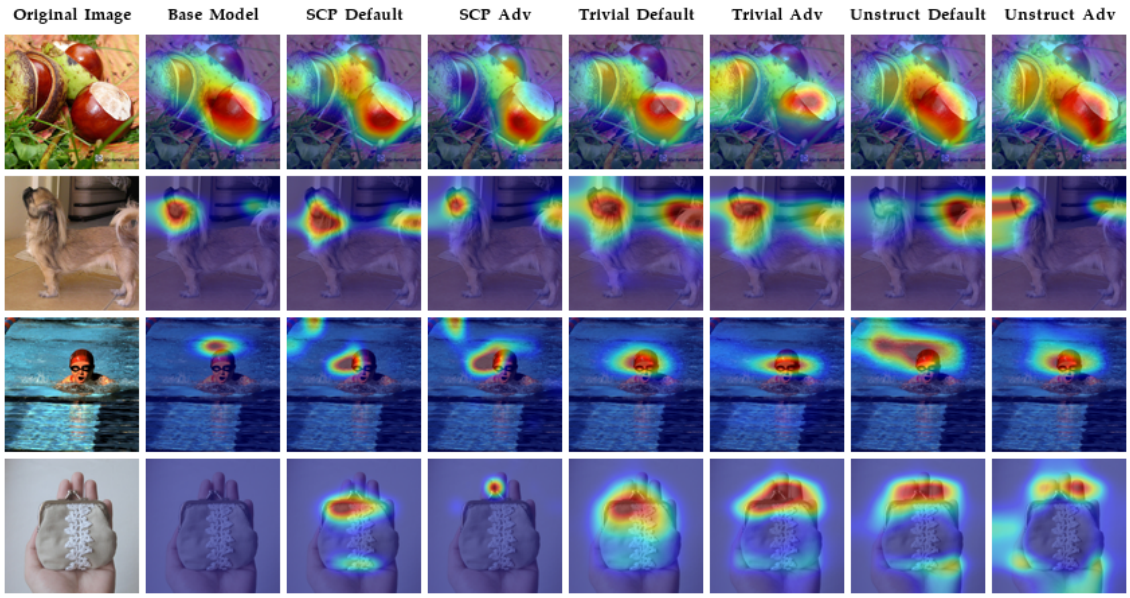


Figure 49: Score-CAM of ImageNet Samples on ResNet18 Models

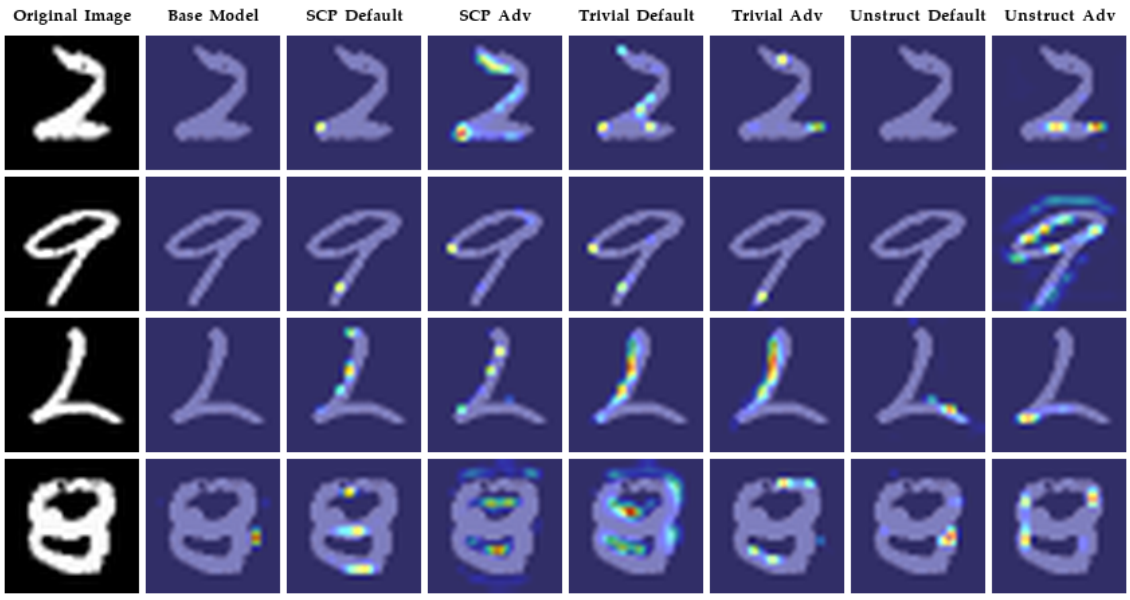


Figure 50: Grad-CAM++ of MNIST Samples on LeNet Models

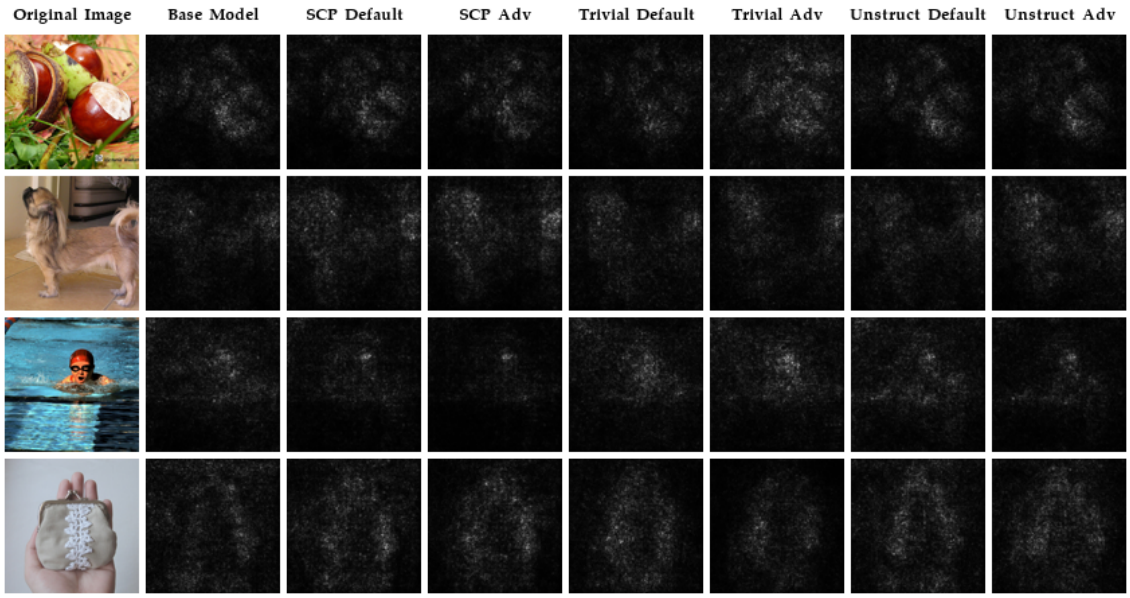


Figure 51: Saliency Map of ImageNet Samples on ResNet18 Models

A.6. CIC Evaluation Additional Information

Layer	Base Model	SCP Adv	SCP Default	Trivial Adv	Trivial Default	Unstruct Adv	Unstruct Default
conv1	0.08781279	0.48008	0.45606107	0.9471709	0.5632543	0.3558637	0.45904762
conv2	0.29960066	0.6552948	0.52955616	0.8598683	0.585877	0.37151915	0.46317258

Table 28: CIC Measurements on LeNet Models

Layer	Base Model	SCP Adv	SCP Default	Trivial Adv	Trivial Default	Unstruct Adv	Unstruct Default
conv1	0.054885756	0.3250901	0.4235409	0.8553723	0.5482346	0.2133922	0.4455451
conv2	0.30437735	0.4681942	0.44806784	0.65282077	0.5279077	0.18600199	0.4661378
PGD: eps: 0.1414, alpha: 0.0157, steps: 20, random_start: false							

Table 29: CIC Measurements With Adversarial Samples on LeNet Models

Layer	Base Model	SCP Adv	SCP Default	Trivial Adv	Trivial Default	Unstruct Adv	Unstruct Default
conv1,	0.98387897	0.034136057	0.7789469	0.07534583	0.5910053	0.035577383	0.821641
layer1.0.conv1,	0.97011673	0.0659272	0.7916894	0.11706206	0.57153636	0.1862359	0.8864051
layer1.0.conv2,	0.9481903	0.0	0.7496066	0.15189044	0.6371079	0.1688061	0.89787394
layer1.1.conv1,	0.9370348	0.019326525	0.79357666	0.14083984	0.6579022	0.16614605	0.8959502
layer1.1.conv2,	0.90468454	0.0	0.7630379	0.17557353	0.7114311	0.08349542	0.8566968
layer2.0.conv1,	0.95154446	0.017862126	0.8431585	0.11266518	0.6765961	0.0990241	0.9034096
layer2.0.conv2,	0.9636862	0.009776085	0.84019995	0.14196101	0.6844055	0.09577027	0.8960773
layer2.1.conv1,	0.96862465	0.0030395824	0.8170408	0.12213799	0.66280085	0.092339605	0.8987754
layer2.1.conv2,	0.96887106	0.004225659	0.81376594	0.14033037	0.66175735	0.10789168	0.89390457
layer3.0.conv1,	0.9567521	0.0	0.8312588	0.11419154	0.6518566	0.04027217	0.91756266
layer3.0.conv2,	0.9564036	0.0	0.8284993	0.10626859	0.6443588	0.05141607	0.9144814
layer3.1.conv1,	0.957949	0.005120044	0.8150743	0.09568095	0.636847	0.060884297	0.91754013
layer3.1.conv2,	0.9533339	0.0112519	0.81270987	0.09563419	0.61120206	0.10937552	0.9097908
layer4.0.conv1,	0.95209646	0.019394841	0.8476887	0.09950563	0.63971436	0.030649872	0.931757
layer4.0.conv2,	0.94554603	0.00032171607	0.81759894	0.093151845	0.6286979	0.035008676	0.92491305
layer4.1.conv1,	0.94574296	0.0054774494	0.8254533	0.091625035	0.6361314	0.04508545	0.92539746
layer4.1.conv2	0.92083603	0.0063372254	0.86763644	0.13040689	0.6897603	0.0045522633	0.9432212

Table 30: CIC Measurements on ResNet18 Models

Layer	Base Model	SCP Adv	SCP Default	Trivial Adv	Trivial Default	Unstruct Adv	Unstruct Default
conv1,	0.6510985	0.5007725	0.5	0.50027144	0.4783777	0.5	0.48113137
layer1.0.conv1,	0.52095807	0.49968746	0.5	0.49984273	0.48271307	0.5	0.5925163
layer1.0.conv2,	0.5141148	0.49906737	0.5	0.4999213	0.46678963	0.5	0.6060387
layer1.1.conv1,	0.4704703	0.4997977	0.5	0.50006926	0.47812366	0.5	0.5869223
layer1.1.conv2,	0.44471398	0.4988686	0.5	0.50003433	0.48752695	0.5	0.58504224
layer2.0.conv1,	0.4182782	0.499503	0.5	0.50011307	0.45224944	0.5	0.5903928
layer2.0.conv2,	0.4887533	0.49921668	0.5	0.50002646	0.4665593	0.5	0.577994
layer2.1.conv1,	0.4954395	0.49937758	0.5	0.49985915	0.4452794	0.5	0.5955707
layer2.1.conv2,	0.5016092	0.4987063	0.5	0.4999775	0.4658717	0.5	0.56662315
layer3.0.conv1,	0.5344697	0.49884883	0.5	0.49949303	0.43193695	0.5	0.56898075
layer3.0.conv2,	0.57017064	0.49937928	0.5	0.4997481	0.4447687	0.5	0.5784451
layer3.1.conv1,	0.5657199	0.49950355	0.5	0.49971902	0.41727486	0.5	0.58912814
layer3.1.conv2,	0.565784	0.49868342	0.5	0.50016934	0.44495124	0.5	0.5884185
layer4.0.conv1,	0.5607878	0.4993746	0.5	0.49884778	0.41611564	0.5	0.5737957
layer4.0.conv2,	0.57286924	0.4999475	0.5	0.4995215	0.42826718	0.5	0.6048833
layer4.1.conv1,	0.5724707	0.50105524	0.5	0.5004205	0.44042006	0.5	0.5918053
layer4.1.conv2	0.49862385	0.5012089	0.5	0.50084126	0.48600984	0.5	0.55810845
PGD: eps: 3.0, alpha: 0.00205, steps: 20, random_start: false							

Table 31: CIC Measurements With Adversarial Samples on ResNet18 Models