On the Robustness of Prunnig Algorithms to Adversarial Attacks

Ca' Foscari University of Venice
Department of Environmental Sciences, Informatics and Statistics

Computer Science Master's Thesis
Year 2022-2023

Graduand Muhammad Tajwar
Supervisor Sebastiano Vascon

Assistant supervisor Antonio Emanuele Cinà

Acknowledgments

First and foremost, I express my sincere gratitude to my mentor, Professor Sebastiano Vascon. His unwavering support and guidance have been invaluable throughout the development of my thesis. His recommendations for relevant research papers have not only been useful but also a source of inspiration for expanding the scope of this work. Under his tutelage, I have gained profound insights into the art of research, making him an exceptional advisor.
I must also acknowledge the collaborative efforts of Antonio Emanuele Cina, one of Professor Vascon's Ph.D. colleagues. Together, we formed an outstanding team, generating numerous innovative ideas that have not only contributed to the success of this project but also laid the groundwork for future endeavors that I plan to pursue during my after journey.
I extend my appreciation to my friends whose unwavering support and contributions were a constant source of motivation during the demanding phases of this project.
Last but certainly not least, my heartfelt thanks go to my parents and especially my brother Muhammad Zakwan for his unconditional support. Their unwavering support has enabled me to receive the best education possible and to pursue my academic career here in Venice. Their patience, encouragement, and warm smiles have been a source of comfort that I will forever cherish.

Abstract

Pruning is a technique in machine learning used to simplify models, reduce overfitting, and improve efficiency. It works by reducing the complexity of a model through the removal of certain components. In the context of neural networks, traditional weight pruning involves setting the smallest weights to zero, effectively eliminating their contribution to the network's output. Structural pruning, on the other hand, takes this concept further by removing not just individual weights, but entire neurons, connections, or layers. This leads to a change in the network architecture itself, potentially resulting in a model that's more efficient, easier to understand, and simpler to implement on hardware. The challenge lies in achieving the right balance, removing enough components to gain efficiency without sacrificing too much model performance. In the context of pruning, a dependency graph can help determine which parts of a neural network can be removed without disrupting the remaining architecture. The graph visualizes how different operations and layers of the network rely on each other. By examining these dependencies, we can identify nodes or connections that, if removed, would not affect the overall data flow, or would only minimally impact the model's performance. This makes dependency graphs a valuable tool for optimizing the process of structural pruning in neural networks. On one hand, a pruned network could potentially be more robust against adversarial attacks. Its reduced complexity might limit the avenues an attacker can exploit. Additionally, the increased interpretability could help in identifying and understanding potential vulnerabilities, pruning could also potentially make a model more vulnerable if important defensive features are pruned away. Also, the change in data flow and dependencies from the pruning process could open up new vulnerabilities.

Keywords Structural Pruning, Spars Learning, Dependency Graph, Neural Networks, ResNet,Wideresnet , Adversarial Attacks, DDN, FMN, Robust Models, Non-Robust Models

Contents

Chapter 1

Introduction

Machine learning has gained unprecedented momentum in various fields, revolutionizing how we approach complex tasks and make data-driven decisions. However, as models continue to grow in size and complexity, they become resource-intensive and often suffer from issues such as high memory consumption, slow inference times, and overfitting. This is where pruning algorithms come into play. Pruning algorithms offer an effective solution to optimize and enhance the efficiency of machine learning models. The concept of pruning draws inspiration from horticultural practices, where unnecessary branches or leaves are trimmed to promote healthier growth. Similarly, in the context of machine learning, pruning involves selectively removing parts of a model that contribute less to its overall performance. This process not only reduces the computational burden but also improves generalization and prevents overfitting.The primary goal of pruning is to create a more compact, efficient, and manageable model without significantly sacrificing its predictive capabilities. Pruning algorithms achieve this by identifying and removing redundant or irrelevant features, nodes, or branches within a model. This leads to a streamlined architecture that retains the essential information necessary for accurate predictions while discarding the noise or unnecessary complexity.

There are several motivations for employing pruning algorithms in machine learning:

Model Efficiency Pruned models are more lightweight, making them suitable for deployment on resource-constrained devices such as mobile phones or Internet of Things (IoT) devices. Reduced model size also translates to faster inference times, enabling real-time applications.

Generalization Improvement Pruning helps prevent overfitting, a common issue in machine learning, by removing parts of the model that memorize noise in the training data rather than learning meaningful patterns. This results in models that generalize better to unseen data.

Interpretability A pruned model often has a simpler structure, which enhances its interpretability. Understanding the relationships between features and predictions becomes easier, promoting trust and accountability in critical applications.

Energy Efficiency In scenarios where energy consumption is a concern, such as in edge computing or embedded systems, pruned models require less computational resources, leading to reduced power consumption.

There are several various pruning algorithms used in machine learning. Brief overview of different types of pruning techniques, including:

Weight Pruning This technique involves removing connections with low-weight magnitudes from neural networks. Various criteria can be used to determine which weights to prune, such as small weight values, gradients, or sensitivity analysis.

Node Pruning Node pruning focuses on removing entire nodes (neurons) from a neural network. Nodes that contribute less to the overall network performance, based on metrics like activation values or gradients, are pruned.

Filter Pruning Commonly applied to convolutional neural networks (CNNs), filter pruning involves removing entire convolutional filters. Filters that correspond to less important features or channels are pruned, resulting in a more efficient model.

Structural Pruning This technique involves removing entire layers or groups of layers from a neural network architecture. Structural pruning requires careful consideration of the network topology and its impact on performance.

Throughout this paper, we will explore the mechanics of these pruning algorithms, discuss their advantages and limitations, and showcase real-world applications to highlight their efficacy. By the end, readers will have a comprehensive understanding of how pruning algorithms contribute to the optimization of machine learning models, enabling the development of faster, more efficient, and highly performant applications across diverse domains.Moreover Pruning algorithms have found extensive applications across a spectrum of real-world domains, showcasing their versatility and impact on enhancing machine learning systems. These applications demonstrate how pruning algorithms address challenges and drive efficiency in various industries including Autonomous Vehicles,Healthcare Diagnostics,Natural Language Processing (NLP),Image and Video Compression,Climate Research and so on.

1.1 Problem Description

In recent years, pruning algorithms have emerged as a powerful tool for optimizing and streamlining machine learning models. These algorithms selectively remove redundant or less influential components of models, resulting in more efficient, lightweight, and faster-to-infer architectures. While pruning brings significant benefits to model efficiency and performance, there is a growing concern about the susceptibility of pruned models to adversarial attacks. Adversarial attacks, aimed at exploiting vulnerabilities in machine learning models, pose a unique challenge to the robustness of pruned models. The question arises: How well do pruned models withstand adversarial manipulations of their input data, and how do pruning algorithms' choices impact this vulnerability? The problem lies at the intersection of two critical domains: pruning algorithms and adversarial machine learning. On one hand, pruning algorithms carefully decide which components to remove, considering factors like weight magnitudes, importance scores, and architectural constraints. On the other hand, adversarial attacks craft subtle perturbations in input data, intending to deceive models into making incorrect predictions.The complexity arises from the potential interaction between pruning choices and adversarial perturbations. Pruned models may have altered decision boundaries, making them differentially sensitive to adversarial attacks compared to their unpruned counterparts. Moreover, the removal of certain components during pruning might lead to the loss of valuable information, rendering the model more susceptible to adversarial manipulations.

1.2 Outline

For the remainder of this thesis, we structure our presentation as follows, Chapter 1 Introduction.In this section, we provide an overview of the context and motivation of our work. We introduce the challenges and significance of addressing Pruning algorithms and adversarial attacks in machine learning models.Chapter 2 is all about the Adversarial Machine learning , we explained in details about how the adversarial machine leaning help us to verify how any machine learning models is effected by the attack, and how it makes any model robust or vulnerable, furthermore we explained in details how DDN(Decoupling Direction and Norm) and FMN(Fast Minimum Norm) attack works and what algorithm they use to perturbed the model thus degrading it.In chapter 3 we present the pruning methods,which is dedicated to the exposition of various pruning techniques employed to enhance the efficiency and performance of machine learning models. We delve into the specifics of these methods, discussing their formulation, operation, and notable properties moreover, we specifically choose two of the pruning techniques namely L1 and Random, to perform pruning on the models, these techniques are known as magnitude pruning,that constitute the core of our research. We begin by elucidating the fundamentals of the machine learning models paradigm. Subsequently, we delve into the intricacies of the various models namely, Addepalli(Resnet18), SehwagProxy(Resnet 18) and Engstrom(Resent18) of which all of these are robust models, and rest of the models are Non-Robust models i.e VGG19, ResNet50 and WideResNet.We expound on their inner workings, formulation, and key attributes.In chapter 4 Experiments is dedicated to the comprehensive presentation of our experimental setup. We detail the experimental scenarios and parameters employed to evaluate the robustness of the proposed algorithms. Through a variety of visualization techniques, we demonstrate how these algorithms respond to changes in the attacker's capacity.In this section, we analyze the outcomes and implications of our experiments. We provide a thorough examination of the robustness demonstrated by the six models in the face of adversarial attacks.Moverover, we use DDN(Decoupling Direction and Norm) and FMN(Fast Minimum Norm) attach to check the robustness of the Robust models and Non-robust models. The results shed light on the effectiveness of our proposed methodologies.Lastly, in chapter 5 Conclusions and Future Work encapsulates our conclusions drawn from this study. We summarize the findings and contributions of this thesis, addressing the achievements made and the broader implications of our work. Additionally, we identify open issues, suggest potential improvements, and outline avenues for future research in the domain of adversarial machine learning.

Chapter 2

Adversarial Machine Learning

Adversarial machine learning has emerged as a critical field aimed at understanding and mitigating vulnerabilities in machine learning models. It explores the susceptibility of models to adversarial attacks, deliberate manipulations of input data with the intent to deceive or mislead the model's predictions. These attacks exploit the underlying vulnerabilities of machine learning algorithms, raising concerns about model security and reliability.
In adversarial machine learning, attackers craft perturbations that are imperceptible to humans but can lead machine learning models to make incorrect predictions. These attacks can have far-reaching consequences, from misclassifying objects in image recognition systems to fooling natural language processing models into generating inappropriate content. As models are increasingly integrated into safety-critical systems, such as autonomous vehicles and medical diagnostics, the need for robust defenses against adversarial attacks becomes paramount.

2.1 The Attacker Goals

The goals of attackers in the context of adversarial machine learning are to exploit vulnerabilities in machine learning models for their own advantage. Attackers aim to manipulate the behavior of the models by introducing carefully crafted perturbations into the input data. These perturbations are designed to be imperceptible to humans but can lead the model to make incorrect predictions. The primary objectives of attackers in adversarial machine learning include:

Misclassification Attackers aim to cause misclassification by altering input data so that the model assigns it to the wrong class. For example, an attacker might manipulate an image of a stop sign to make it look like a yield sign to a self-driving car's image recognition system.
Evasion Attackers try to evade detection or security mechanisms by crafting input data that is incorrectly classified as benign or safe. This can be problematic in scenarios where machine learning models are used for security screening or anomaly detection.
Data Poisoning In situations where models are trained on data from multiple sources, attackers might attempt to manipulate the training data to inject malicious samples. These samples can compromise the model's integrity and lead to incorrect predictions.
Privacy Violation Attackers may attempt to extract sensitive information about

the training data or the model itself. This can be particularly concerning when models handle confidential or private data, such as medical records or financial information.
Generating Adversarial Examples Attackers create adversarial examples to understand model vulnerabilities, evaluate the effectiveness of attacks, or test the robustness of defense mechanisms. These adversarial examples can also serve as a means to expose flaws in machine learning systems.

Understanding attackers' goals is essential for developing effective defense mechanisms and strategies to safeguard machine learning models against adversarial attacks. Adversarial machine learning research focuses on creating models and techniques that can withstand these manipulation attempts, ensuring the security, reliability, and trustworthiness of machine learning systems in real-world applications.
Targeted and Untargeted Attacks Targeted attacks focus on undermining the Adversarial Machine Learning model's performance for specific instances. In contrast, untargeted attacks are broader in scope, aiming to diminish the model's accuracy across a variety of instances.

2.2 Attacker's Knowledge

Based on the attacker's level of knowledge, various types of attacks are orchestrated to exploit system vulnerabilities. These attacks can be categorized as follows:

- **White-box Attack:** The attacker possesses comprehensive knowledge of the target system, including its architecture, gradients, and parameters. This in-depth understanding empowers the attacker to execute precise and sophisticated attacks.

- **Gray-box Attack:** In this scenario, the attacker's access and knowledge are limited, making it challenging to breach the system's defenses effectively. The attacker grapples with partial insights into the system's workings.

- **Black-box Attack:** A Black-box Attack unfolds when the attacker is entirely unaware of the system's details, with the sole ability to query samples. This restricted access necessitates creative approaches to exploit vulnerabilities.

- **No-box Attack:** No-box attacks are characterized by the attacker's utilization of a surrogate model. This strategy capitalizes on the attacker's comprehensive understanding of the surrogate model to target the desired system, despite minimal access to the original data.

Black-box, Grey-box, and White-box Attacks These attacks differ based on the attacker's level of knowledge about the model. Black-box attacks stem from limited understanding of the model's architecture and functioning. Grey-box attacks, on the other hand, involve intermediate knowledge, while white-box attacks assume comprehensive knowledge about the model's design and parameters.

In addition to these attack classifications, there exist gradients of attacker knowledge:

- **Perfect Knowledge:** Entails complete awareness of the graph's structure and components.
- **Moderate Knowledge:** Reflects a relatively limited understanding of the dataset.
- **Minimal Knowledge:** Represents the most challenging scenario, where the attacker has access to only a small fraction of the dataset's information.

Each level of attacker knowledge and the corresponding attack techniques contribute to a complex landscape of security challenges, underscoring the importance of robust defense strategies to safeguard against adversarial exploits.

2.3 Adversarial Attacks

Adversarial attacks within the realm of machine learning extend their reach to neural networks dealing with graph data, aiming to disrupt the accuracy of predictions. These attacks are orchestrated to introduce intentional distortions in order to deceive Adversarial Machine Learning models into making incorrect forecasts. Several attack categories exist in this context:

2.3.1 Poisoning Attacks

Poisoning attacks represent a malicious tactic in the realm of cybersecurity, wherein adversaries exploit vulnerabilities within machine learning models by deliberately introducing tainted data during the training process. These attacks aim to compromise the integrity and functionality of the model's learning process, leading to erroneous predictions or behaviors during deployment. Such attacks can manifest in various forms, including the injection of misleading or fabricated training samples, often designed to influence the model's decision boundaries. Poisoning attacks can be particularly harmful as they compromise the trustworthiness of machine learning systems, impacting sectors like autonomous driving, e-commerce, and security applications. Detecting and mitigating these attacks requires advanced defense mechanisms such as outlier detection, robust training techniques, and vigilant data validation. The dynamic nature of these attacks calls for ongoing research and development to stay ahead of evolving threats and safeguard the reliability of machine learning models.

2.3.2 Evasion Attack

Poisoning attacks, also referred to as training-time attacks, occur during the model's training phase. They involve the insertion of manipulated data into the training set, causing the Adversarial Machine Learning model to learn from these misleading instances. Conversely, evasion attacks, or test-time attacks, transpire during the testing phase. In these attacks, models trained on unaltered data are tested using data intentionally perturbed or modified to produce misleading outcomes.

An evasion attack is a type of cyberattack that targets machine learning models, specifically aiming to manipulate their decision-making processes by intentionally altering input data in ways that evade detection or mislead the model's predictions. These attacks are designed to exploit vulnerabilities within the model's training and inference process, often by adding imperceptible perturbations to input samples. Evasion attacks can compromise the integrity and reliability of machine learning systems, allowing malicious actors to trick the model into making incorrect classifications or decisions.

Evasion attacks typically involve crafting adversarial examples – subtly modified inputs that appear similar to the original data but can lead the model to produce erroneous outputs. These attacks are particularly concerning in applications where machine learning systems play a critical role, such as security, finance, and healthcare. Countermeasures against evasion attacks include the development of robust machine learning models, adversarial training, input sanitization techniques, and improved model evaluation procedures. As evasion attacks continue to evolve, ongoing research and the adoption of advanced defense strategies are crucial to maintaining the security and effectiveness of machine learning systems.

2.3.3 DDN (Decoupling Direction and Norm)

The DDN (Decoupling Direction and Norm) attack is an advanced and sophisticated form of adversarial attack in the field of machine learning, particularly targeting deep neural networks. This attack strategy is designed to manipulate the network's behavior by simultaneously perturbing both the direction and the norm of the input data. Unlike traditional adversarial attacks that focus solely on perturbing the data points within a constrained region, the DDN attack exploits a more comprehensive approach by allowing for variations in both direction and magnitude. This technique is particularly insidious as it can potentially overcome defensive mechanisms like input normalization and regularization that many neural networks employ.

The DDN attack, due to its dual focus on direction and norm, seeks to deceive the network into misclassifying input samples. By allowing for more varied perturbations, it can bypass existing defense mechanisms that predominantly account for norm-based perturbations. The attack can effectively exploit the network's sensitivity to variations in input data and lead to misclassification or other undesirable outputs. To counter this sophisticated form of attack, researchers and practitioners need to develop novel defense mechanisms that account for a wider range of adversarial perturbations and vulnerabilities. This underscores the ongoing need for innovative strategies to enhance the robustness and security of deep neural networks in the face of evolving adversarial threats like the DDN attack.

Problem Formulation

Let's consider a scenario where we have a sample, denoted as x, from the input space χ, and it's associated with a true label, y_{true}, which belongs to a set of possible labels, . Now, there exists a distance measure, $D(x_1, x_2)$, which quantifies the comparison between two input samples, ideally capturing their perceptual similarity. $P(y|x, \theta)$ represents a model or classifier parameterized by θ. We define an example, $\tilde{x} \in X$, as adversarial, particularly for non-targeted attacks, if the

label with the highest probability, $\arg\max_j P\left(y_j \mid \tilde{x}, \theta\right) \neq y_{\text{true}}$, is not equal to y_{true}, and the distance $D(x, \tilde{x}) \leq \epsilon$ to a specified maximum perturbation ϵ.
For targeted attacks, which additionally require achieving a desired class y_{target}, it must hold that $\arg\max_j P\left(y_j \mid \tilde{x}, \theta\right) \neq y_{\text{target}}$. Here, we denote $J(x, y, \theta)$ as the cross-entropy between the model's prediction for input x and label y. You can visualize a targeted attack in Figure 2.3.3 on the ImageNet dataset against an Inception v3 model.
Generally, attacks generated through gradient-based optimization procedures, limiting our analysis to differentiable classifiers. These attacks can be framed in two ways: first, to minimize the distortion $D(x, \tilde{x})$, and second, to maximize the loss within a defined region where $D(x, \tilde{x}) \leq \epsilon$.
As an illustration, consider a scenario where the distance function is a norm (e.g., L_0, L_2, or L_∞), and the inputs are images where each pixel value is constrained between 0 and M. In a white-box setting, the optimization process to obtain a non-targeted attack with the least distortion δ can be expressed as:

$$\begin{gathered}\min_\delta \|\delta\| \text{ subject to } \arg\max_j P\left(y_j \mid x+\delta, \theta\right) \neq y_{\text{true}} \\ \text{and } 0 \leq x+\delta \leq M\end{gathered} \tag{2.1}$$

A similar formulation applies for targeted attacks, where the constraint is modified to enforce equality with the target class.
Now, if the objective is to achieve the worst possible loss within a specified maximum noise of norm ϵ, we can frame the problem as follows:

$$\begin{aligned}\min_\delta P\left(y_{\text{true}} \mid x+\delta, \theta\right) \text{ subject to } \|\delta\| \leq \epsilon \\ \text{and } 0 \leq x+\delta \leq M\end{aligned} \tag{2.2}$$

Our focus primarily revolves around gradient-based attacks optimizing the L_2 norm of the distortion. Although this measure may not perfectly capture perceptual similarity, it finds widespread usage in computer vision to gauge image similarity. For instance, when comparing image compression algorithms, the Peak Signal-to-Noise Ratio (PSNR), directly related to the L_2 measure, is frequently employed. Nonetheless, developing a differentiable distance measure that effectively captures perceptual similarity remains an ongoing research challenge.

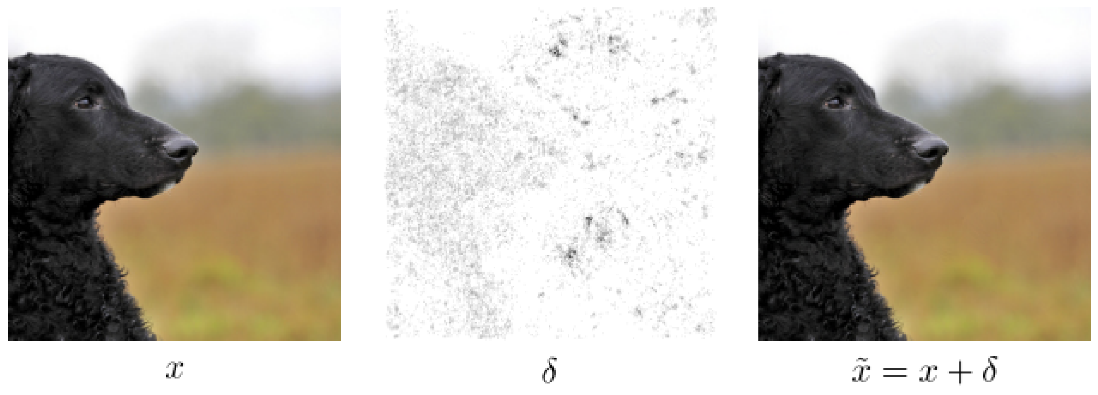


Figure 2.1: Illustrates an instance of an adversarial image within the ImageNet dataset. The original sample x is correctly recognized as a Curly-coated retriever. However, when we introduce a perturbation δ to this image, we create an adversarial image that is now classified as a microwave with $||\delta||_2 = 0.7$

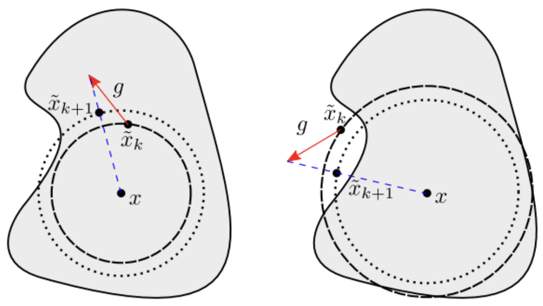
DDN Working

The problem definition indicates that identifying the worst adversary within a fixed region is a relatively more straightforward task. In Equation 2.2, both constraints can be expressed in terms of δ, allowing for optimization using projected gradient descent.

However, finding the closest adversarial example is more challenging. Equation 2.1 introduces a constraint related to the model's prediction, which cannot be effectively managed through a straightforward projection. A common approach, as employed by Szegedy et al. [53] and in the $C\&W$ [3] attack, is to approximate the constrained problem in Equation 2.1 with an unconstrained one by replacing the constraint with a penalty term. This approach involves optimizing both the norm of δ and a classification term concurrently, where a sufficiently high parameter C is introduced. In the broader context of constrained optimization, this penalty-based approach is a well-established principle [19]. However, it's worth noting that penalty methods pose practical challenges. The primary challenge lies in selecting an appropriate parameter C, which often requires ad hoc decisions. If C is too small , the resulting example may not be adversarial, while an excessively large C can dominate, resulting in an adversarial example with more noise. This can become particularly problematic when optimizing with a limited number of steps, such as when applied in adversarial training scenarios.

Algorithm 1 Decoupled Direction and Norm Attack

```
Input: $x$: original image to be attacked
Input: $y$: true label (untargeted) or target label (targeted)
Input: $K$: number of iterations
Input: $\alpha$: step size
Input: $\gamma$: factor to modify the norm in each iteration
Output: $\tilde{x}$: adversarial image
 1: Initialize $\delta_0 \leftarrow \mathbf{0}$, $\tilde{x}_0 \leftarrow x$, $\epsilon_0 \leftarrow 1$
 2: If targeted attack: $m \leftarrow -1$ else $m \leftarrow +1$
 3: for $k \leftarrow 1$ to $K$ do
 4:     $g \leftarrow m\nabla_{\tilde{x}_{k-1}} J(\tilde{x}_{k-1}, y, \theta)$
 5:     $g \leftarrow \alpha \frac{g}{\|g\|_2}$                ▷ Step of size $\alpha$ in the direction of $g$
 6:     $\delta_k \leftarrow \delta_{k-1} + g$
 7:     if $\tilde{x}_{k-1}$ is adversarial then
 8:         $\epsilon_k \leftarrow (1-\gamma)\epsilon_{k-1}$      ▷ Decrease norm
 9:     else
10:         $\epsilon_k \leftarrow (1+\gamma)\epsilon_{k-1}$      ▷ Increase norm
11:     end if
12:     $\tilde{x}_k \leftarrow x + \epsilon_k \frac{\delta_k}{\|\delta_k\|_2}$   ▷ Project $\delta_k$ onto an $\epsilon_k$-sphere around $x$
13:     $\tilde{x}_k \leftarrow \text{clip}(\tilde{x}_k, 0, 1)$        ▷ Ensure $\tilde{x}_k \in \mathcal{X}$
14: end for
15: Return $\tilde{x}_k$ that has lowest norm $\|\tilde{x}_k - x\|_2$ and is adversarial
```



(a) $\tilde{x}_k$ not adversarial (b) $\tilde{x}_k$ adversarial

Figure 2.2: Here is the illustration of an attack which is untargeted:We have a shaded region representing the portion of the input space directed as y_{true}.In scenario (a), $\tilde{x}_k$ for now is not considered adversarial yet.For the next iteration, we then increase the ϵ_{k+1} norm, or conversely, reduce it in scenario (b).In both scenario, we take a g steps starting from the point where are at right now represented as $\tilde{x}$, and x which is the center of ϵ_{k+1}-sphere we then project it back .This process involves adjusting the perturbation ϵ to iteratively explore the input space and find the adversarial example.

The complete procedure is outlined in Algorithm 1 and visually depicted in Fig. 2.2. We initiate the process with the original image x and progressively refine the noise δ_k.

During iteration k, if the sample $\tilde{x}_k = x + \delta_k$ is considered as adversarial yet, we then take a larger norm which is $\epsilon_{k+1} = (1+\gamma)\epsilon_k$. Conversely, if we see that sample is already adversarial, we opt for a smaller $\epsilon_{k+1} = (1-\gamma)\epsilon_k$. In both scenarios, we take a g steps(as shown in step 5 of Algorithm 1) from the point $\tilde{x}_k$ (indicated by the red arrow in Figure. 2.2) and directs it back onto an ϵ_{k+1} -sphere centered at x (following the direction represented by the blue line in Figure. 2.2). This projection yields $\tilde{x}_{k+1}$. Finally, $\tilde{x}_{k+1}$ is further projected onto the valid region of the input space . For instance, in the case of images which are normalized to the range [0, 1], we clip the value of each and every pixel to make sure that it fall within this range (which is the step 13 of Algorithm 1). Additionally, it's also important to take into consideration of the concept quantizing of the image during each iteration to make sure that the attack generates a valid image.

If we reach a point where we see that decision boundary is making tangent to the sphere as aforementioned, the direction of g aligns with δ_{k+1}. It then will be projected in the same direction as δ_k. Consequently, the norm will go back and forth between the two sides of the boundary in that direction. By multiplying ϵ by $1+\gamma$ and $1-\gamma$, we achieve a global reduction of the norm by $1-\gamma^2$, facilitating a perfect search for the optimal norm.

2.3.4 FMN(Fast Minimum-norm Attack)

In this section of the thesis, we focus on the FMN attack, also referred to as the Fast Minimum-Norm attack, to assess the robustness of the models employed in this study. As mentioned earlier, our model selection comprises three robust models and three non-robust models. The FMN attack operates based on the chosen norm and aims to identify perturbations within the input data, even when they are of minimal magnitude.

Within the realm of gradient-based minimum-norm attacks, there are three primary sub-categories:

- Soft-constraint attacks, such as CW, strike a balance between the confidence of misclassified samples and the perturbation size. These attacks require fine-tuning of a trade-off hyperparameter on a per-sample basis to find the smallest perturbation, necessitating numerous iterations for convergence.

- Boundary attacks, like BB and FAB, move along the decision boundary toward the nearest point to the input sample. These attacks converge relatively quickly. However, BB relies on an adversarial starting point, and both methods involve solving relatively complex optimization problems at each step.

- Recent minimum-norm projected-gradient attacks like DDN perform a maximum-confidence attack at each step while adhering to a specified perturbation budget ϵ. They iteratively adjust ϵ to reduce the perturbation size.It's important to note that DDN is specific to l_2 norm and can't diverse to others.

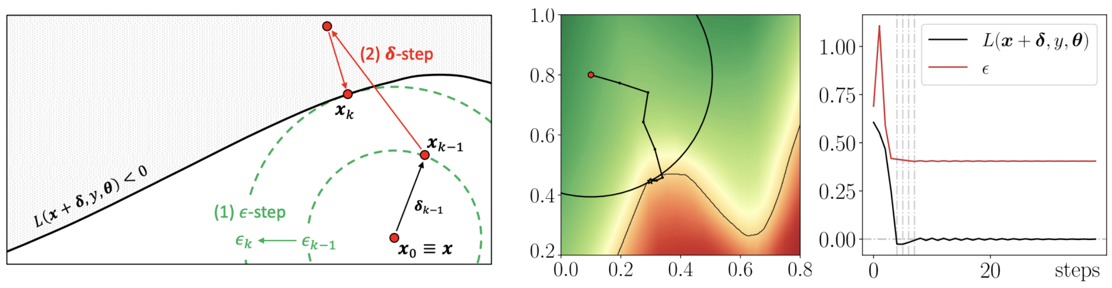


Figure 2.3: (a) The FMN attack algorithm is conceptually depicted in the leftmost plot. It involves two steps: the ϵ-step, which updates the constraint size ϵ to minimize its distance to the decision boundary, and the ϵ-step, which updates the perturbation δ using a projected-gradient step to maximize confidence in misclassification within the current ϵ-constraint.(b) An example of our attack on a two-dimensional problem is shown in the middle plot, that shows the loss function and the size of their constraint ϵ. The algorithm initiates by pushing the starting point (represented as a red dot) toward the adversarial region (highlighted in red). Subsequently, it perturbs the point around the decision boundary to enhance the current solution towards a local optimum.If we take a look at the vertical lines they are just showing the steps (characterized by a smaller $||\delta^*||$ and $L < 0$) is discovered.

To address these limitations, we introduce a novel and efficient minimum-norm attack method referred to as FMN (Fast Minimum-Norm) attack, as outlined in Section 2. FMN retains the key advantages of DDN while extending its applicability to various l_p norms, where p can take on values of $0, 1, 2$, or ∞. Our extensive experimentation on diverse datasets and models is detailed in Section 3, demonstrating that FMN offers substantial improvements in terms of convergence speed and computational efficiency when compared to existing minimum-norm attacks. This superior performance is particularly notable in scenarios involving l_0, l_1, and 'l_∞ norms, while FMN also achieves comparable results to l_p-norm attacks. Moreover, FMN consistently identifies equal or better optimal solutions across the majority of tested scenarios and l_p norms. In summary, FMN presents a comprehensive solution that possesses the essential attributes required for a robust adversarial attack, representing a significant advancement in the evaluation of adversarial robustness.

Minimum-Norm Adversarial Examples with Adaptive Projections

Problem formulation. In the context of an input sample x within the range of $[0,1]^d$, where x belongs to class y within the set $1,...,c$, the objective of an untargeted attack is to discover the minimum-norm perturbation denoted as δ^* This perturbation should be such that it transforms the original input x into an adversarial example $x^* = x + \delta^*$ leading to a misclassification of x. The problem can be mathematically expressed as follows:

$$\boldsymbol{\delta}^\star \in \arg\min_{\boldsymbol{\delta}} \|\boldsymbol{\delta}\|_p, \tag{2.3}$$

$$\text{s.t. } L(\boldsymbol{x}+\boldsymbol{\delta}, y, \boldsymbol{\theta}) < 0, \tag{2.4}$$

$$\boldsymbol{x}+\boldsymbol{\delta} \in [0,1]^d, \tag{2.5}$$

Algorithm 1 Fast Minimum-norm (FMN) Attack

Input: $\boldsymbol{x}$, the input sample; t, a variable denoting whether the attack is targeted ($t = +1$) or untargeted ($t = -1$); y, the target (true) class label if the attack is targeted (untargeted); γ_0 and γ_K, the initial and final ϵ-step sizes; α_0 and α_K, the initial and final $\boldsymbol{\delta}$-step sizes; K, the total number of iterations.

Output: The minimum-norm adversarial example $\boldsymbol{x}^\star$.

```
1: $x_0 \leftarrow x$, $\epsilon_0 = 0$, $\delta_0 \leftarrow 0$, $\delta^\star \leftarrow \infty$
2: for $k = 1, \dots, K$ do
3:     $g \leftarrow t \cdot \nabla_\delta L(x_{k-1} + \delta, y, \theta)$  // loss gradient
4:     $\gamma_k \leftarrow h(\gamma_0, \gamma_K, k, K)$  // $\epsilon$-step size decay (Eq. 6)
5:     if $L(x_{k-1}, y, \theta) \geq 0$ then
6:         $\epsilon_k = \|\delta_{k-1}\|_p + L(x_{k-1}, y, \theta)/\|g\|_q$ if adversarial not found yet else $\epsilon_k = \epsilon_{k-1}(1 + \gamma_k)$
7:     else
8:         if $\|\delta_{k-1}\|_p \leq \|\delta^\star\|_p$ then
9:             $\delta^\star \leftarrow \delta_{k-1}$  // update best min-norm solution
10:        end if
11:        $\epsilon_k = \min(\epsilon_{k-1}(1 - \gamma_k), \|\delta^\star\|_p)$
12:    end if
13:    $\alpha_k \leftarrow h(\alpha_0, \alpha_K, k, K)$  // $\delta$-step size decay (Eq. 6)
14:    $\delta_k \leftarrow \delta_{k-1} + \alpha_k \cdot g/\|g\|_2$  // gradient-scaling step
15:    $\delta_k \leftarrow \Pi_\epsilon(x_0 + \delta_k) - x_0$
16:    $\delta_k \leftarrow \mathrm{clip}(x_0 + \delta_k) - x_0$
17:    $x_k \leftarrow x_0 + \delta_k$
18: end for
19: return $x^\star \leftarrow x_0 + \delta^\star$
```

where $||.||_p$ indicates the l_p-norm operator. The loss L in the equation 2.4 is defined as :

$$L(\boldsymbol{x}, y, \boldsymbol{\theta}) = f_y(\boldsymbol{x}, \boldsymbol{\theta}) - \max_{j \neq y} f_j(\boldsymbol{x}, \boldsymbol{\theta}) \tag{2.6}$$

In this context, $f_j(x, \theta)$ represents the confidence score assigned by the model f to classify input x as class j, where θ represents the model's learned parameters. If we assume that the classifier assigns x to the class with the highest confidence, denoted as $y^* = arg\ max_{j \in 1,...,c} f_j(x, \theta)$, then if the x is not correctly classified it makes the loss function $L(x, y, \theta)$ negative.In the Eq. (2.5) the constraint keep sure the perturbed sample $x + \delta$ remains within the valid input space. It's important to note that this problem typically involves a non-convex loss function L (with respect to its first argument) due to the non-convex nature of the underlying decision function f. As a result, it can have multiple locally optimal solutions. Additionally, when the input sample x is already adversarial (i.e., $L(x, y, \theta) < 0$), the solution is trivial, with $\delta^* = 0$.

Extension to the targeted case. The objective of a targeted attack is to ensure that the input sample gets misclassified into a specific target class, denoted as y0. This is achieved by modifying the loss function in Eq. (2.6) to become $L^t(\boldsymbol{x}, y', \boldsymbol{\theta}) = \max_{j \neq y'} f_j(\boldsymbol{x}, \boldsymbol{\theta}) - f_{y'}(\boldsymbol{x}, \boldsymbol{\theta}) = -L(\boldsymbol{x}, y', \boldsymbol{\theta})$, which essentially changes the sign of the loss and replaces the true class label y with the target class label y'.

Solution Algorithm. To gain more insight to the problem (2.3)-(2.5), we rewrite it using the upper bound ϵ on $||\delta||_p$:

$$\min_{\epsilon, \boldsymbol{\delta}} \epsilon, \quad \text{s.t. } \|\boldsymbol{\delta}\|_p \leq \epsilon \tag{2.7}$$

We can outline the algorithm into two primary steps, akin to the DDN approach [43], illustrated in Fig. 2.3(a). These steps involve the adjustment of the maximum perturbation size, denoted as ϵ, independently of the perturbation itself, represented as δ. Specifically, the first step focuses on adapting the constraint size ϵ to reduce its distance from the decision boundary, referred to as the ϵ-step. Meanwhile, the second step updates the perturbation δ using a projected-gradient method to minimize the loss function L while adhering to the constraint of the given ϵ, known as the δ-step. This approach essentially translates into an iterative process resembling projected gradient descent, where ϵ is dynamically modified to seek the minimum-norm adversarial example. The complete algorithm, named Algorithm 1, is provided above, with a detailed explanation of the two principal steps which is explained below.

ϵ-step: In this phase (lines 4-12 in Algorithm 1), the objective is to refine the upper bound ϵ on the perturbation's norm. The underlying principle involves increasing ϵ if the current sample is not adversarial, which is indicated when $L(x_{k1}, y, \theta) \geq 0$. Conversely, if the sample is adversarial then $L(x_{k1}, y, \theta) < 0$, the aim is to reduce ϵ . When determining the increment or decrement of ϵ, the attack strategy depends on whether an adversarial example has been previously discovered. If no adversarial example has been found so far, an estimate of the distance to the decision boundary is made using a linear approximation, and ϵ_k is updated as follows:$\epsilon_k = \|\boldsymbol{\delta}_{k-1}\|_p + L(\boldsymbol{x}_{k-1}, y, \boldsymbol{\theta}) / \|\nabla L(\boldsymbol{x}_{k-1}, y, \boldsymbol{\theta})\|_q$,this equation help us to attack in a fast manner that leading to the boundary decision.In contrast, if a previous adversarial sample exists but the current one is not adversarial, it suggests that the current ϵ estimate is mannerly smaller than the solution with the minimum norm. In this scenario, ϵ is increased by a small fraction as $\epsilon_k = \epsilon_{k-1}(1 + \gamma_k)$, with γ_k representing a decaying step size.Furthermore, when decreasing ϵ, if the current sample is indeed adversarial i.e $L(\boldsymbol{x}_{k-1}, y, \boldsymbol{\theta})$, the algorithm aims to verify if the current solution can be improved. If the corresponding ϵ_k value is greater than the best-known $||\delta^*||_p$ found thus far, the best value is retained, and ϵ_k is set to $||\delta^*||_p$. These updates of ϵ, performed multiplicatively, this is because of the fluctuating fashion near to the boundary and seeking the points of adversarial. To make sure of convergence, the step size γ_k is subjected to cosine annealing.

$$\gamma_k = h\left(\gamma_0, \gamma_K, k, K\right) = \gamma_K + \frac{1}{2}\left(\gamma_0 - \gamma_K\right)\left(1 + \cos\left(\frac{k\pi}{K}\right)\right) \tag{2.8}$$

where k being the current step, K is number of steps in total, and γ_0 and γ_K initial and final steps.

δ-step. This step involves the update of δ (lines 13-17 in Algorithm 1) with the aim of finding the adversarial example that is misclassified with the utmost confidence, essentially minimizing the loss function L within the current constraint of ϵ (defined by Eq. 2.7) and subject to the bounds specified in Eq.2.5. This process is achieved through a projected-gradient step, where we traverse along the negative gradient of L. We employ a normalized steepest descent with a decreasing step size σ to handle potential issues arising from noisy gradients while ensuring convergence (line 14). It's worth noting that in this step, the gradient is rescaled by its l_2 norm, preserving its direction. The step size σ undergoes cosine annealing to facilitate convergence (Eq. 2.8). After updating δ, it is then projected onto the l_p-norm constraint of size ϵ, ensuring compliance with the constraint stated in Eq. (2.7). The projection operation Π_ϵ is straightforward for p = ∞ and p = 2. In the case of p = 1, we utilize an efficient algorithm proposed by Duchi et al. [8]. For p = 0, only the first ϵ components of δ with the largest absolute values are retained. Finally, any components of δ that violate the bounds specified in Eq. (2.5) are clipped (line 16).

Execution example. In Fig. 2.3(b), we provide an illustration of our algorithm's execution in a two-dimensional context. Initially, the sample is adjusted by following the negative gradient of the loss function L towards the decision boundary. As soon as an adversarial point is identified, the algorithm adapts by reducing the value of ϵ to seek an improved solution. Consequently, the point is projected back into the non-adversarial region, and ϵ is increased, albeit by a smaller, gradually diminishing quantity. These oscillations facilitate the movement of the point along the boundary, aiming to reach a local optimum—specifically, an adversarial point situated on the boundary. This situation is characterized because of the loss function of the gradient as some contraints of the norm has opposite directions.FMN would be converge relatively quickly to a favorable local optimum, given that the step size is steadily reduced to a suitably small value and a sufficient number of iterations are executed.

Adversarial initialization. Our attack offers the flexibility of being initiated either from the original input sample x or from a designated point x_{init}. In cases where the attack commences from x_{init}, whether for an untargeted or targeted attack, we employ a ten-step binary search procedure that operates between x and x_{init}. The primary objective of this search is to locate an adversarial point that is in closer proximity to the decision boundary. Specifically, we aim to determine the minimum value of ϵ for which the loss function $L\left(\boldsymbol{x} + \Pi_\epsilon\left(\boldsymbol{x}_{\text{init}} - \boldsymbol{x}\right), y, \boldsymbol{\theta}\right) < 0$ (or $L^t < 0$ for targeted attacks). Subsequently, our attack is launched with the corresponding values of x_k, ϵ_k, δ_k, and δ^*.

Differences with DNN.FMN introduces significant modifications to both the algorithm and the formulation originally presented in DDN. The key distinctions are as follows: **Perturbation Rescaling**: DDN always adjusts the perturbation to a fixed size of ϵ. However, this approach poses challenges when employing different norms, particularly sparser ones, as it limits the attack's ability to explore neighboring regions effectively. In contrast, FMN adopts a more flexible approach.**Choice of Loss Function:** While DDN utilizes the cross-entropy loss, FMN employs the logit difference as its loss function (L). This change is motivated by the fact that the logit difference is less susceptible to saturation effects.**Dynamic ϵ Estimation:** FMN dynamically estimates the value of ϵ and does not require an initial value. This adaptive approach contributes to the attack's effectiveness.**Decay of γ:** In FMN, the parameter γ is subject to decay to enhance convergence, particularly around better minimum-norm solutions. This adjustment helps dampen oscillations near the decision boundary.**Adversarial Initialization:** FMN introduces the option of initializing the attack from an adversarial point. This feature can significantly expedite the convergence process by utilizing a rapid line-search algorithm to locate the boundary, followed by subsequent queries to refine the outcome

Chapter 3

Pruning Methods

The recent growth of edge computing applications has led to an increased need for deep neural compression [14, 18, 20, 31, 32, 59, 61, 62, 64, 68]. Pruning is one of the most effective and practical network compression methods [6, 11, 27, 28, 40, 54, 56, 67], which aims to remove redundant parameters from a network to reduce its size and potentially speed up inference. There are two main types of pruning: structural [5, 25, 66] and unstructural [7, 12, 40]. Structural pruning physically removes grouped parameters, changing the structure of the neural network, while unstructural pruning zeroes out partial weights without modifying the network structure. Structural pruning is more widely applicable where we do not have to be dependent on artificial intelligence tools or related tools that helps us to eliminate the costs of computations and memory.[34, 63].

Nonetheless, the task of structural pruning itself presents a considerable challenge, particularly when dealing with modern deep neural networks that possess intricate and interconnected internal structures. This challenge arises due to the fact that deep neural networks are constructed using a multitude of fundamental components, such as convolution, normalization, and activation functions. These components, whether characterized by parameters or not, are inherently intertwined through complex connections. Consequently, even a seemingly simple task, like removing a single channel from a convolutional neural network as depicted in Figure 3.1(a), demands careful consideration of its interconnectedness across all layers. Neglecting this aspect could result in rendering the network inoperable. To elaborate, the concept of residual connections mandates that the output channels of two consecutive convolutional layers match, compelling them to be pruned jointly. This principle is equally applicable when structurally pruning other architectures like Transformers, Recurrent Neural Networks (RNNs), and Graph Neural Networks (GNNs), as shown in Figures 3.1(b-d).

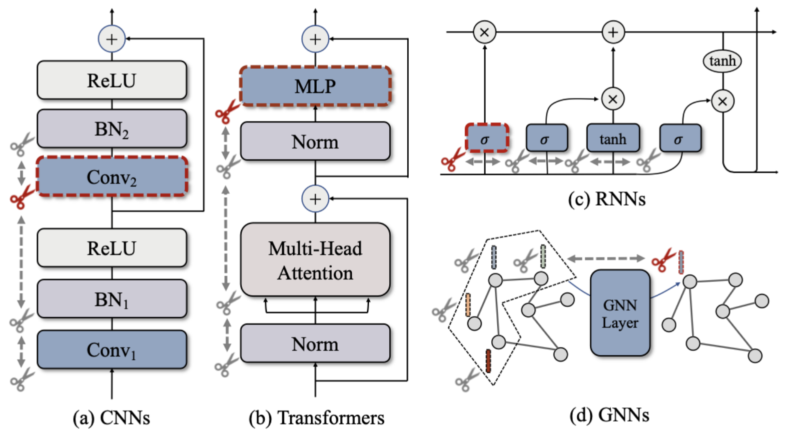


Figure 3.1: Parameters across various layers exhibit inherent interdependencies throughout diverse network architectures. As a result, the pruning of multiple layers simultaneously becomes necessary. For instance, when considering the pruning of Conv2 as shown in (a), it becomes imperative to concurrently prune all associated layers Conv1, BN1, BN2 within the same block.

3.1 Structural and Unstructural Pruning

Pruning has made remarkable advancements in the realm of network acceleration, evident from numerous studies found in existing literature [4, 17, 16, 25, 33, 35]. Predominant pruning techniques can be broadly divided into two categories: structural pruning [5, 25, 29, 65] and unstructural pruning [7, 24, 40, 49]. The objective of structural pruning is to eliminate a cluster of parameters, thereby reducing the dimensions of neural networks. Conversely, unstructured pruning involves nullifying specific weights while leaving the network's architecture intact. In practical terms, unstructured pruning is notably simple to implement and adaptable across diverse networks. Nonetheless, it often mandates specialized AI accelerators or software to facilitate model acceleration [13]. On the contrary, structural pruning enhances inference efficiency by physically eliminating parameters from networks, expanding its applicability across a broader spectrum [25, 34]. In existing literature, the scope of pruning algorithms encompasses various aspects, including pruning strategies [16, 35], parameter identification [49, 39, 40], layer sparsity [23, 48], and training methodologies [42, 55].

3.1.1 Pruning Grouped Parameters

Within intricate network configurations [25, 30, 34, 66, 69], interdependencies can emerge among clusters of parameters, demanding their concurrent removal. The pruning of interconnected parameters has garnered research attention from the early stages of structural pruning [25, 33, 35]. For example, in the scenario of pruning two consecutive convolutional layers, excising a filter from the initial layer mandates the simultaneous pruning of related kernels in the subsequent layer [25]. While manual assessment of parameter dependencies is feasible, this approach becomes exceedingly labor-intensive when applied to intricate networks, as evident in various prior investigations [25, 66, 69]. Additionally, such manual techniques lack transferability to novel architectures, significantly limiting the scope of pruning applications. Recently, some initial attempts have been made to decode the intricate interrelations among layers [30, 66]. Nonetheless, existing methods still rely on empirical guidelines or pre-defined architectural patterns, thus rendering

them inadequately versatile for a comprehensive range of structural pruning applications.
In this thesis we tried to solve all of the aforementioned issues regarding parameter dependency and furthermore, we will see how pruning effects the different models and their accuracies thus also diving into robustness against adversarial attacks.

3.2 Methods

3.2.1 Dependency in Neural Networks

This study is centered on the structural pruning of neural networks [10] while considering the constraint of parameter interdependency. To maintain a broad perspective, our method is developed using fully-connected (FC) layers. We commence by examining a linear neural network comprising three consecutive layers, as depicted in Figure 3.2(a). These layers are characterized by 2-D weight matrices, namely w_l, w_{l+1}, and w_{l+2}. The aim is to achieve network slimming through structural pruning by eliminating neurons. In this context, it becomes evident that certain dependencies arise among parameters, denoted as $w_l \Leftrightarrow w_{l+1}$. This linkage mandates the simultaneous pruning of w_l and w_{l+1}. Specifically, when pruning the k-th neuron bridging w_l and w_{l+1}, both $w_l[k, :]$ and $w_{l+1}[:, k]$ need to be removed. Existing literature tackles layer dependencies and facilitates structural pruning in deep neural networks through manually devised and model-specific strategies [16, 25]. However, as illustrated in Figure 3.2(b-d), there exists a multitude of dependencies. The manual analysis of each dependency on a case-by-case basis becomes impractical, especially considering that straightforward dependencies can be nested or combined to create more intricate patterns. To address the challenge of dependencies in structural pruning, this study introduces the concept of a Dependency Graph. This approach offers a comprehensive and fully automated mechanism for modeling dependencies.

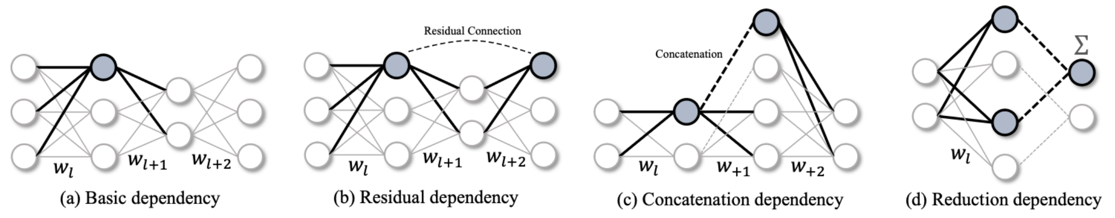


Figure 3.2: Parameters that are grouped together and exhibit interdependence across various structures necessitate concurrent pruning of the highlighted parameters.

3.2.2 Dependency Graph

Grouping

In order to facilitate structural pruning, the initial step involves categorizing layers based on their interdependencies [10]. In a formal sense, our objective is to determine a grouping matrix $G \in \mathbb{R}^{L\times L}$, where L denotes the count of layers within the network earmarked for pruning. Here, $G_{ij} = 1$ signifies the presence of a dependency between the i-th layer and the j-th layer. To ensure self-dependency for ease of computation, we set $\mathrm{Diag}(G) = \mathbf{1}_{1\times L}$. Leveraging this grouping matrix, the identification of all linked layers with interdependence to the i-th layer, denoted as $g(i)$, becomes straightforward:

$$g(i) = \{j \mid G_{ij} = 1\}$$

However, discerning the grouping patterns from a neural network is intricate, as contemporary deep networks could encompass numerous layers with intricate interconnections. Consequently, this results in a comprehensive and intricate grouping matrix G. Within this matrix, G_{ij} is influenced not solely by the i-th and j-th layers, but also by intermediate layers positioned between them. Consequently, such non-local and implicit relationships can't be addressed through simple rules in the majority of scenarios. To surmount this challenge, we avoid a direct estimation of the grouping matrix G and propose an alternative, yet easily estimable, method for modeling dependencies – the Dependency Graph. This Dependency Graph facilitates the derivation of G in an efficient manner.

Dependency Graph

We shall initiate our exploration by taking into considerations of a group denoted as $g = \{w1, w2, w3\}$, which exhibits several dependencies i.e $w1 \Leftrightarrow w2$, $w2 \Leftrightarrow w3$, and $w1 \Leftrightarrow w3$. With closer analysis of this structure [10], it becomes apparent that there exists some redundancy within it. For instance, the interdependency $w1 \Leftrightarrow w3$ can be deduced from the relationships $w1 \Leftrightarrow w2$ and $w2 \Leftrightarrow w3$ through a recursive process. Initially, we consider $w1$ as the starting point and assess its dependence on other layers, such as $w1 \Leftrightarrow w2$. Subsequently, $w2$ serves as a fresh starting point to recursively extend the dependency, thereby inducing the interconnection $w2 \Leftrightarrow w3$. This recursive iteration culminates in a transitive relation, $w1 \Leftrightarrow w2 \Leftrightarrow w3$. In this scenario, a mere two dependencies suffice to depict the relationships within group g. Likewise, the grouping matrix mentioned in Section 3.2 also harbors redundancy in terms of dependency modeling, permitting compression into a more concise structure with fewer edges while upholding the same information content. We substantiate that a novel graph D, which gauges the local interdependency between adjacent layers and is termed the Dependency Graph, can serve as a productive reduction of the grouping matrix G. Diverging from G, the Dependency Graph exclusively records dependencies between contiguous layers connected directly. This Graph D can be regarded as the transitive reduction [2] of G, encompassing identical vertices as G while minimizing the number of edges. Formally, the construction of D ensures that for all $G_{ij} = 1$, a path exists in D connecting vertex i and vertex j. Consequently, we can get G_{ij} by just looking at the vertices between the i and j.

Network Decomposition

However, we discover that constructing the dependency graph [10] on a per-layer basis can present challenges in practical implementation. This is due to the fact that certain fundamental layers, like fully-connected layers, can have multiple distinct pruning strategies such as $w[k,:]$ and $w[:,k]$, as elaborated in Section 3.1, which involve reducing the dimensions of inputs and outputs, respectively. Furthermore, networks also encompass non-parameterized operations such as skip connections, which also influence the interdependencies between layers [36]. To address these complexities, we introduce a fresh notation to disassemble a network $F(x; w)$ into more granular and fundamental constituents, denoted as $F = \{f_1, f_2, ..., f_L\}$, in which all the { indicated the layers that can either be the layers of convolution or they can be either be related to operations perhaps non-parameterized.We can do the modeling connections at the layer level, we direct our attention towards the dependencies between the inputs and outputs of these constituents. Specifically, we represent the input and output of constituent f_i as f_i^- and f_i^+, respectively. For any given network, this final disassembly can be formulated as $\mathcal{F} = \left\{f_1^-, f_1^+, \ldots, f_L^-, f_L^+\right\}$. This notation simplifies dependency modeling and accommodates distinct pruning approaches for the same layer.

Algorithm 1 Layer Grouping Algorithm

```
Require: Dependency Graph $D(F, E)$
Ensure: Groups $G$
 1: $G = \{\}$
 2: for $i \in \{1, 2, ..., 2 \cdot |F|\}$ do
 3:     $g = \{i\}$
 4:     repeat
 5:         UNSEEN $= \{1, 2, ..., 2 \cdot |F|\} - g$
 6:         $g' = \{j \in$ UNSEEN $\mid \exists k \in g, D_{kj} = 1\}$
 7:         $g = g \cup g'$
 8:     until $g' = \emptyset$
 9:     $G = G \cup \{g\}$
10: end for
11: return $G$
    =0
```

Dependency Modeling

By leveraging this notation, we reformulate the neural network as depicted in Equation 3.1, revealing two primary categories of dependencies: inter-layer dependency and intra-layer dependency [10], demonstrated as follows:

$$(f_1^-, \underbrace{f_1^+) \leftrightarrow (f_2^-}_{\text{Inter-layer Dep}}, f_2^+) \cdots \leftrightarrow \underbrace{(f_L^-, f_L^+)}_{\text{Intra-layer Dep}} \tag{3.1}$$

The symbol $\leftrightarrow$ indicates the connection between two consecutive layers. Analysis of these two types of dependencies leads to straightforward yet comprehensive principles for dependency modeling:

- Inter-layer Dependency: A dependency $f_{i-} \Leftrightarrow f_{j+}$ consistently emerges when layers $f_{i-} \leftrightarrow f_{j+}$ are connected.

Algorithm 2 Dependency Graph Generation Algorithm

```
Require: A neural network $F(x; w)$
Ensure: Dependency Graph $D(F, E)$
1: $f^- = \{f_1^-, f_2^-, ..., f_L^-\}$ decomposed from $F$
2: $f^+ = \{f_1^+, f_2^+, ..., f_L^+\}$ decomposed from $F$
3: Initialize Dependency Graph $D = \mathbf{0}_{2L \times 2L}$
4: for $i \in \{0, 1, ..., L\}$ do
5:     for $j \in \{0, 1, ..., L\}$ do
6:         $D(f_i^-, f_j^+) = D(f_j^+, f_i^-) = \underbrace{\mathbb{Z}\left[f_i^- \leftrightarrow f_j^+\right]}_{\text{Inter-layer Dep}} \vee \underbrace{\mathbb{Z}\left[i = j \wedge \text{sch}\left(f_i^-\right) = \text{sch}\left(f_j^+\right)\right]}_{\text{Intra-layer Dep}}$
7:     end for
8: end for
9: return $D$
   =0
```

- Intra-layer Dependency: A dependency $f_{i-} \Leftrightarrow f_{i+}$ exists if f_{i-} and f_{i+} share the same pruning scheme, denoted as $sch(f_{i-}) = sch(f_{i+})$.

Starting with inter-layer dependency, its estimation becomes straightforward if the network's topological structure is known. For connected layers with $f_{i-} \leftrightarrow f_{j+}$, a dependency invariably exists since f_{i-} and f_{j+} correspond to the same intermediate features within the network. The intra layer depicts us that in order to maintain the neural network authenticity the input and output later must be pruned together. Numerous network layers adhere to this criterion, such as batch normalization, where inputs and outputs share the same pruning scheme ($sch(f_{i-}) = sch(f_{i+})$), leading to simultaneous pruning, as depicted in Figure 3.3. On the contrary, layers like convolutions exhibit distinct pruning schemes for their inputs and outputs (e.g., $w[:, k, :, :] \neq w[k, :, :, :]$), as demonstrated in Figure 3.3. Consequently, $sch(f_{i-}) \neq sch(f_{i+})$ and no dependency exists between the input and output of a convolution layer.

With these established principles, we can formally define dependency modeling as follows:

$$D\left(f_i^-, f_j^+\right) = \underbrace{\mathbb{Z}\left[f_i^- \leftrightarrow f_j^+\right]}_{\text{Inter-layer Dep}} \vee \underbrace{\mathbb{Z}\left[i = j \wedge \text{sch}\left(f_i^-\right) = \text{sch}\left(f_j^+\right)\right]}_{\text{Intra-layer Dep}} \tag{3.2}$$

Here, $\vee$ and $\wedge$ denote logical "OR" and "AND" operations, respectively, and $\mathbb{Z}$ is an indicator function that returns "True" if the condition holds. The first term addresses inter-layer dependency arising from network connectivity, while the second term handles intra-layer dependency introduced by shared pruning schemes between input and output layers. Notably, DepGraph, being a symmetric matrix, adheres to $D(f_{i-}, f_{j+}) = D(f_{j+}, f_{i-})$. In Figure 3.3, the DepGraph of a CNN block with connections is visually depicted. Algorithms 1 and 2 outline the procedures for dependency grouping and also modeling.

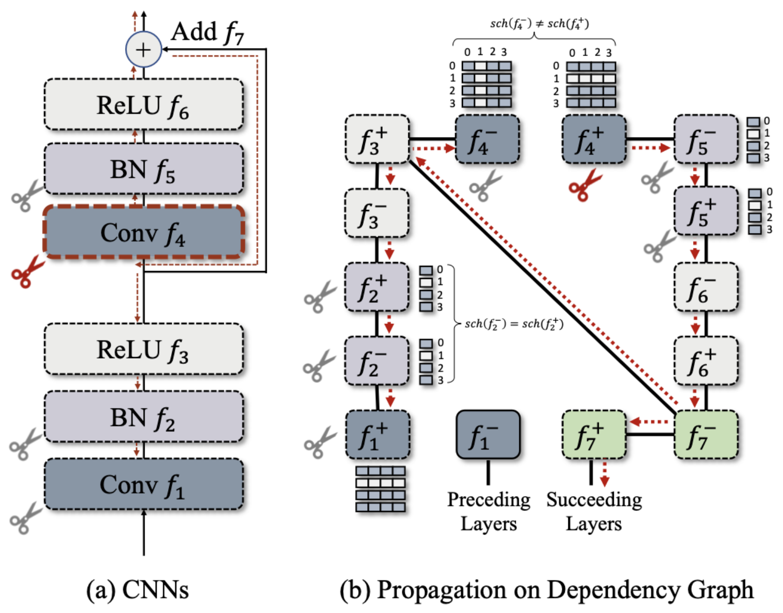


Figure 3.3: Layer grouping is established through a recursive propagation process on the Dependency Graph (DepGraph), commencing from f_{4+}. In this specific illustration, no intra-layer dependency exists between the input f_{4-} and the output f_{4+} of the convolutional layer due to the divergent pruning schemes highlighted earlier.

3.2.3 Magnitude Pruners

Up to this point in our thesis, we have employed two distinct pruning techniques: the L1 pruner[26] and the Random pruner [26]. Both of these methods fall under the category of magnitude pruners, as they rely on assessing the average magnitude of their kernel weights. It is worth noting that the library we have utilized offers a wide array of pruning strategies, including MetaPruner, Magnitude Pruner, BN-ScalePruner, GroupNorm Pruner, and GrowingRegPruner. However, our thesis primarily concentrates on the magnitude pruner approach.

Our approach focuses on the elimination of less essential filters from a pre-trained model to enhance computational efficiency while keeping the decrease in accuracy to a minimum. To assess the relative importance of a filter within each layer, we compute the sum of its absolute weights, denoted as $\sum |\mathcal{F}_{i,j}|$, which is essentially its l_1-norm, represented as $|\mathcal{F}_{i,j}|_1$. We represents the number of channels with n_i which are the input to the filter, $\sum |\mathcal{F}_{i,j}|$, and this sum of $\mathcal{F}$ is calculating average magnitude by considering the kernel weights, the resulting value provides an estimate of the magnitude of the resulting feature map. Filters with smaller kernel weights typically generate feature maps with less pronounced activations when compared to other filters within that layer.

The process of removing m filters from the ith convolutional layer follows these steps:

- Calculate the sum of absolute kernel weights, denoted as $s_j = \sum_{l=1}^{n_i} \sum |\mathcal{K}_l|$, for each filter $\mathcal{F}_{i,j}$
- Arrange the filters in ascending order based on their s_j values.
- Eliminate **m** filters with the smallest s_j values along with their associated

feature maps. Additionally, remove the corresponding kernels in the subsequent convolutional layer.

- Create new kernel matrices for both the ith and $i+1th$ layers, and copy over the remaining kernel weights to the new model.

Relationship To Pruning Weights

The process of pruning filters based on the sum of their absolute weights is akin to magnitude-based weight pruning, as outlined by Han et al [14]. Magnitude-based weight pruning removes entire filters when all the kernel weights within a filter fall below a specified threshold. However, this method necessitates meticulous threshold tuning, making it challenging to predict the precise number of filters that will be pruned. Additionally, it results in sparse convolutional kernels, which can be challenging to accelerate due to the absence of efficient sparse libraries, particularly in cases of low sparsity.

Relationship To Group-sparse Regularization On Filters

Recent research, such as the work by Zhou et al [70] and Wen et al [58], has employed group-sparse regularization, typically using $l_{2,1}$-norm ($\sum_{j=1}^{n_i} \|\mathcal{F}_{i,j}\|_2$ for each filter) on convolutional filters. This regularization technique also encourages filters with small l_2-norms to approach zero, effectively eliminating them ($\mathcal{F}_{i,j} = 0$). In practical applications, we have not observed a significant distinction between the l_2-norm and l_1-norm when it comes to filter selection, as both measures tend to highlight important filters with relatively large values. The process of zeroing out weights in multiple filters during training achieves a similar outcome to the strategy of iterative pruning and retraining, as discussed in a later section.

Pruning Filters Across Multiple Layers

Now, let's delve into the process of pruning filters across the network. In previous research, the typical approach involved pruning weights layer by layer, followed by iterative retraining to compensate for any potential loss of accuracy, as outlined in the work by Han et al [14]. However, there are compelling reasons to explore the pruning of filters across multiple layers simultaneously:

1. Efficiency in Deep Networks: Pruning and retraining on a per-layer basis can be exceedingly time-consuming in deep neural networks.

2. Holistic Network View: Pruning layers across the entire network provides a more comprehensive perspective on the network's robustness, resulting in a more compact network architecture.

3. Complexity Considerations: In the case of intricate networks like ResNet, taking a holistic approach may be essential. For instance, when pruning ResNet, we have to take into consideration that if we pruned the indentity maps or either ith layer of the any residual block, then we have to pruned some more of the other layers.

This broader pruning strategy offers advantages in terms of efficiency, overall network evaluation, and the management of complex network architectures.

When it comes to pruning filters across multiple layers, we explore two distinct strategies for layer-wise filter selection:

- **Independent Pruning:** This approach independently determines which filters to prune at each layer, without taking into account the status of filters in other layers.
- **Greedy Pruning:** In contrast, the greedy pruning strategy considers filters that have already been removed in previous layers. However, it does not consider the kernels associated with previously pruned feature maps when calculating the sum of absolute weights.

Figure 3.4 visually illustrates the difference between these two approaches in terms of calculating the sum of absolute weights. While the greedy approach may not be globally optimal, it offers a holistic perspective and often results in pruned networks with higher accuracy, particularly when a significant number of filters are being pruned.

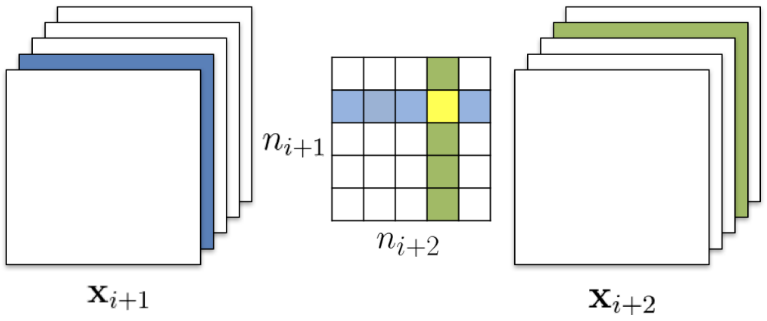


Figure 3.4: When it comes to pruning filters across consecutive layers, it's essential to understand the difference between the two strategies:Independent Pruning, filter sums (represented by the green columns) are calculated without considering feature maps that have been removed in the previous layer (depicted in blue). As a result, the kernel weights indicated in yellow are still included in the pruning process. This approach ultimately leads to a kernel matrix with dimensions of $(n_{i+1} - 1) \times (n_{i+2} - 1)$.Conversely, the greedy pruning strategy does not take into account the kernels associated with previously pruned feature maps. Therefore, it calculates filter sums only for the active feature maps in the current layer. This approach also results in a kernel matrix with dimensions of $(n_{i+1} - 1) \times (n_{i+2} - 1)$.Both of these approaches have their implications and outcomes in the pruning process.

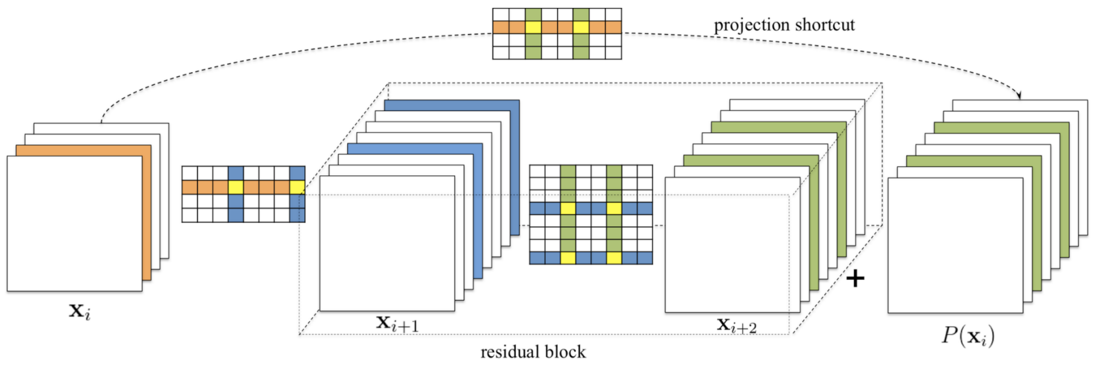


Figure 3.5: The figure represents blocks of residual blocks with the shortcut to projections,which is indicated in green, can be found by the results of pruning the projection shortcut. However, the first layer of the residual block can be pruned without any specific restrictions. This strategy ensures that pruning decisions for the second layer are influenced by the preceding shortcut projection, optimizing the pruning process within the residual block.

When dealing with simpler CNNs like VGGNet or AlexNet, the process of pruning filters in any convolutional layer is relatively straightforward. However, in the case of complex network architectures like Residual networks (He et al [15]), the task of filter pruning becomes more intricate. The architecture of ResNet introduces certain constraints, requiring a careful approach to filter pruning.
In Figure 3.4, we illustrate the process of filter pruning within residual blocks featuring projection mapping. In this context, the filters within the 1st layer of the residual block can be pruned without constraints, as such pruning doesn't alter the count of output feature maps generated by the block.However, pruning filters in the second convolutional layer of the residual block is more challenging. This is because there is a correspondence between the output feature maps of the second layer and the identity feature maps. Therefore, pruning this layer necessitates careful planning. To ascertain the identity feature maps eligible for pruning, we employ a selection criterion rooted in the characteristics of the filters found within the shortcut convolutional layers, which are typically with 1×1 kernels. The pruning of the second layer of the residual block is aligned with the filter index chosen during the pruning of the shortcut layer.This approach ensures that the pruning process in Residual networks accounts for the importance of identity feature maps and maintains the integrity of the network's structure.

Retraining Pruned Networks To Regain Accuracy

Following the pruning of filters, it's crucial to address the performance degradation by retraining the network. There are two main strategies for pruning filters across multiple layers [26]:

1. **Prune Once and Retrain:** In this strategy, filters from multiple layers are pruned simultaneously, and the network is retrained until it regains its original accuracy.

2. **Prune and Retrain Iteratively:** This approach involves pruning filters either layer by layer or filter by filter, followed by iterative retraining. The model is retrained before proceeding to prune the next layer. This allows the model's weights to adapt to the changes introduced during the pruning process.

We have observed that for layers that are resilient to pruning, the "prune once and retrain" strategy can effectively remove significant portions of the network, and any loss in accuracy can typically be recovered through a relatively short retraining period (which is shorter than the original training time). However, in cases where sensitive layers lose some filters or substantial portions of the network are pruned, it may not be possible to fully restore the original accuracy. In such scenarios, the iterative "prune and retrain" approach may yield better results. It's important to note that the iterative process requires more training epochs, especially for very deep networks.

3.3 Methodologies

We utilized three resilient models, namely, Addepalli2022Efficient(ResNet18)[1], Sehwag2021Proxy(ResNet18)[52], and Engstrom2019Robustness(ResNet50)[9].In parallel, we employed three models lacking robustness:

Wang2023Better-WideResNet-28-10 [57], ResNet-50[9], and VGG-19 [60], each featuring distinctive parameter settings for a comprehensive comparative analysis.

Below we briefly explain all the models, including their architectures, parameters, activation function and training of these models.

Addepalli2022Efficient(ResNet18)

This model is recently used by Addepalli [1] from effective and efficient data augmentation technique in adversarial training.Addepalli2022Efficient-ResNet18 is likely based on the ResNet-18 architecture, which is a popular convolutional neural network (CNN) architecture. It consists of multiple residual blocks.For the reminder the figure 3.6 below illustrate a simple ResNet-18 model:

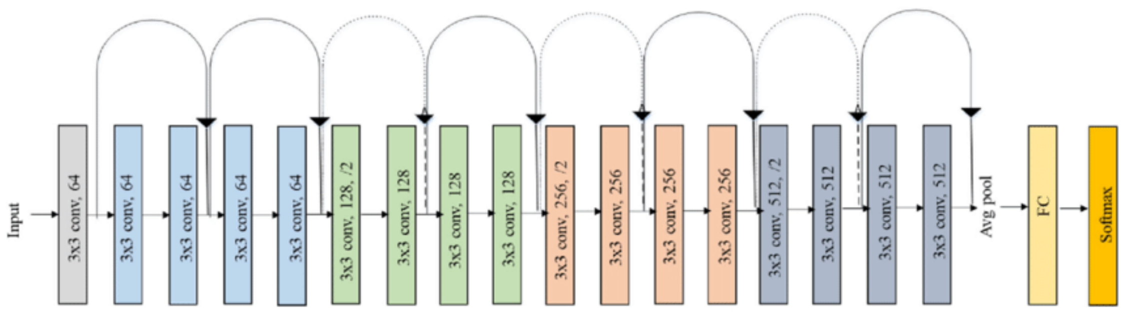


Figure 3.6: ResNet-18 Architecture

Moreover,in the paper [1] they purposed a new approach called Diverse Augmentation based joint Adversarial Training as known as DAJAT for the better data augmentation.The model is trained on CIFAR-10, CIFAR-100 and ImageNette.For improving the quality of the training data they used empirical risk minimization(ERM) which is based on the random transformation.In their research, they initiated an examination of the underlying factors contributing to the divergent patterns observed in Standard and Adversarial Training. Subsequently, they demonstrated the feasibility of harnessing intricate augmentations within Adversarial training by concurrently training models on both straightforward and intricate data augmentations. This approach involved the utilization of distinct batch-normalization layers for each augmentation type, as visually depicted in Figure 3.7.

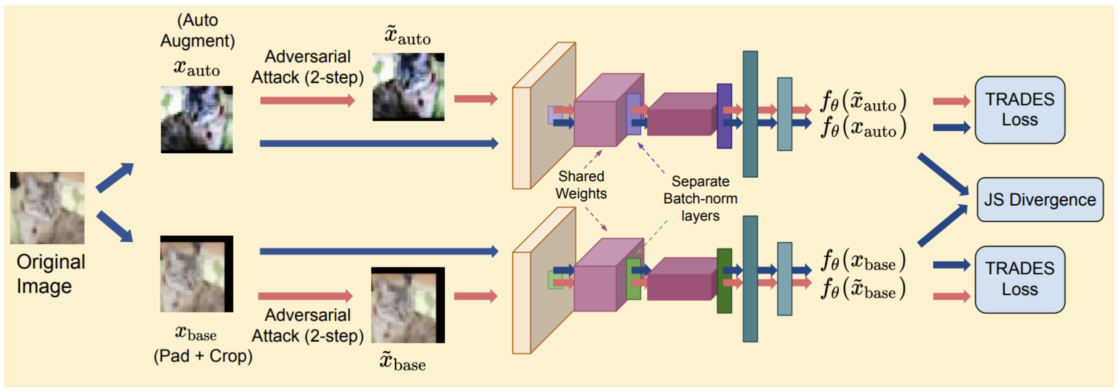

(Auto Augment)
x_{auto}
Adversarial Attack (2-step)
$\tilde{x}_{\text{auto}}$
Original Image
x_{base}
(Pad + Crop)
Adversarial Attack (2-step)
$\tilde{x}_{\text{base}}$
Shared Weights
Separate Batch-norm layers
$f_\theta(\tilde{x}_{\text{auto}})$
$f_\theta(x_{\text{auto}})$
$f_\theta(x_{\text{base}})$
$f_\theta(\tilde{x}_{\text{base}})$
TRADES Loss
JS Divergence
TRADES Loss

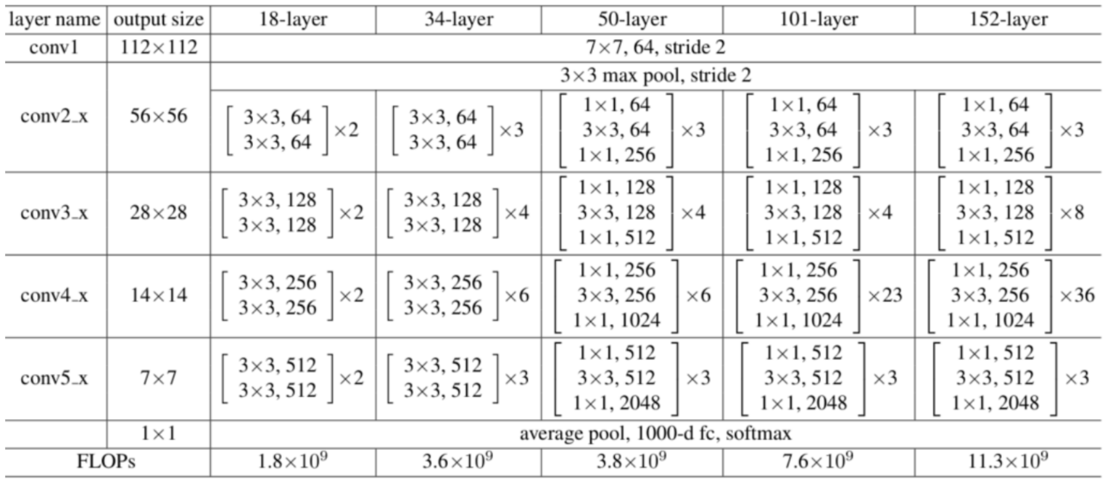
Figure 3.7: DAJAT Architecture

Sehwag2021Proxy(ResNet18)

This model is also based on the ResNet18 Architecture as explained in Figure 3.6, In their research [52] they try to make the model more robust by using vast data generated by the advanced generative models from the proxy distributions.Initially, their aim is to establish a formal comprehension of the transference of robustness from the classifiers that are trained on proxy distributions to the actual data. Their analysis leads to the proof that the contrast in robustness between a classifier on these two distributions can be bounded from above by the conditional Wasserstein distance existing between them. Subsequently, they leverage proxy distributions to yield substantial enhancements in the efficacy of adversarial training across five distinct datasets. Notably, on the CIFAR-10, they achieve notable enhancements in robust accuracy, boasting improvements of up to 7.5% and 6.7% in the l_∞ model and l_2 threat model, respectively, compared to baselines not utilizing proxy distributions. Additionally, they enhance the robust accuracy on CIFAR-10 by 7.6%. Furthermore, they illuminate how various generative models offer varied enhancements in robust training performance. Introducing a robust discrimination approach, they delve into the characterization of the individual impacts of generative models, shedding light on why diffusion-based generative models, as opposed to generative adversarial networks, constitute a superior choice for proxy distribution.

Engstrom2019Robustness(ResNet50)

Engstrom2019Robustness is basically a robustness library which is based on ResNet-50 architecture,many of the researchers used this library for making the model more robustness, this includes [9, 50, 45, 51, 47, 46] research papers.Resnet-50 architecture is moreorless the same resnet but with 50 layers,In Figure 3.8 below I tried to briefly explain this architecture.



layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix}3\times3, 64\\3\times3, 64\end{bmatrix}\times2$	$\begin{bmatrix}3\times3, 64\\3\times3, 64\end{bmatrix}\times3$	$\begin{bmatrix}1\times1, 64\\3\times3, 64\\1\times1, 256\end{bmatrix}\times3$	$\begin{bmatrix}1\times1, 64\\3\times3, 64\\1\times1, 256\end{bmatrix}\times3$	$\begin{bmatrix}1\times1, 64\\3\times3, 64\\1\times1, 256\end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix}3\times3, 128\\3\times3, 128\end{bmatrix}\times2$	$\begin{bmatrix}3\times3, 128\\3\times3, 128\end{bmatrix}\times4$	$\begin{bmatrix}1\times1, 128\\3\times3, 128\\1\times1, 512\end{bmatrix}\times4$	$\begin{bmatrix}1\times1, 128\\3\times3, 128\\1\times1, 512\end{bmatrix}\times4$	$\begin{bmatrix}1\times1, 128\\3\times3, 128\\1\times1, 512\end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix}3\times3, 256\\3\times3, 256\end{bmatrix}\times2$	$\begin{bmatrix}3\times3, 256\\3\times3, 256\end{bmatrix}\times6$	$\begin{bmatrix}1\times1, 256\\3\times3, 256\\1\times1, 1024\end{bmatrix}\times6$	$\begin{bmatrix}1\times1, 256\\3\times3, 256\\1\times1, 1024\end{bmatrix}\times23$	$\begin{bmatrix}1\times1, 256\\3\times3, 256\\1\times1, 1024\end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix}3\times3, 512\\3\times3, 512\end{bmatrix}\times2$	$\begin{bmatrix}3\times3, 512\\3\times3, 512\end{bmatrix}\times3$	$\begin{bmatrix}1\times1, 512\\3\times3, 512\\1\times1, 2048\end{bmatrix}\times3$	$\begin{bmatrix}1\times1, 512\\3\times3, 512\\1\times1, 2048\end{bmatrix}\times3$	$\begin{bmatrix}1\times1, 512\\3\times3, 512\\1\times1, 2048\end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

Figure 3.8: ResNet-50 Architecture

The ResNet architecture, comprising 50 layers, incorporates various key components, as depicted in the table below:

- **Initial Convolutional Layer:** It consists of a 7×7 kernel convolution operation with 64 distinct kernels, all executed with a stride of 2.

- **Max Pooling Layer:** Subsequently, a max-pooling layer with a 2-sized stride is applied.

- **Stacked Layers:** The network then proceeds with a series of layers, each composed of a 3×3 convolution operation employing 64 kernels, followed by two more layers: one with 1×1 convolutional layers having 64 kernels and another with 1×1 convolutional layers utilizing 256 kernels. This trio of layers is recurrently applied three times.

- **Additional Stacked Layers:** The architecture further incorporates 12 layers, consisting of 1×1 convolutional layers with 128 kernels, 3×3 convolutional layers with 128 kernels, and 1×1 convolutional layers with 512 kernels. This set of layers is iterated four times.

- **Further Layers:** Following that, there are 18 more layers, encompassing 1×1 convolutional layers employing 256 cores, 3×3 convolutional layers with 256 cores, and 1×1 convolutional layers utilizing 1024 cores. These 18 layers are repeated six times.

- **Continuation of Layers:** Subsequently, there are 9 additional layers, featuring 1×1 convolutional layers with 512 cores, 3×3 convolutional layers with 512 cores, and 1×1 convolutional layers with 2048 cores, repeated three times.

- **Final Layers:** The network concludes with an average pooling layer, followed by a fully connected layer containing 1000 nodes. The softmax activation function is employed in this final layer for classification purposes.

VGG19

In order to do the better comparison between the robust models we need to use other non-robust models.We start with the VGG19 [60] model,VGG19 consist of 19 layers, below Figure explain the VGG19 architecture:

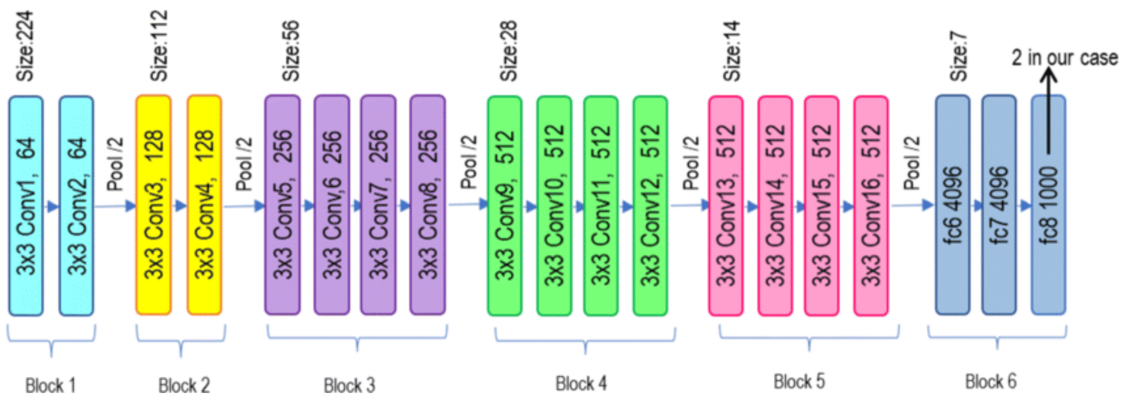


Figure 3.9: VGG19 Architecture

Let's explore the architecture of VGG in a concise manner:

- VGGNet takes image of size 224×224 as input. As for ImageNet the model's creators consistently cropped the center 224×224 patch from each image to maintain uniform input dimensions.
- VGG's convol layers employ a minimal receptive field, specifically 3×3, which is the smallest size capable of capturing vertical and horizontal information. Additionally, 1×1 convolution filters are utilized for linear transformations of the input. Following these convolutions, a ReLU (Rectified Linear Unit) activation function is applied.
- Hidden layers within the VGG use the ReLU activation function. VGG generally avoids using Local Response Normalization (LRN) as it tends to increase memory usage and training time without delivering significant improvements in overall accuracy.
- VGGNet incorporates three fully connected layers. Among these layers, the first two consist of 4096 channels each, while the third comprises 1000 channels, corresponding to the number of classes for classification tasks.

Wang2023Better-WideResNet-28-10

In the research paper [57], the authors used WideResNet-28-10 architecture to make the model more robust, in this paper they use DDPM (denoising diffusion probablistic model) to generate data that made improvements in adversarial training.Before going further in details lets discuss about the WideResNet.Wide Residual Networks (WRNs) involve the exploration of various parameters, including the architecture of the ResNet block, depth (referred to as the deepening factor l), and width (referred to as the widening factor k) within the ResNet block.

In the context of WRNs, when k equals 1, it maintains the same width as a standard ResNet. Conversely, for k greater than 1, it becomes k times wider compared to the ResNet.

The notation **WRN-d-k** signifies a WRN with a depth of d and a widening factor k.It's worth noting that Pre-Activation ResNet is utilized for CIFAR-10, CIFAR-100, and SVHN datasets, while the original ResNet is employed for the ImageNet dataset. A key distinction lies in their architectural order, Pre-Activation ResNet follows a structure of batch normalization and ReLU activation before convolution, whereas the original ResNet adheres to a structure of Convolution-Batch Normalization-ReLU. Generally, Pre-Activation ResNet tends to outperform the original variant, although the advantage is less pronounced for ImageNet when the number of layers remains around 100.

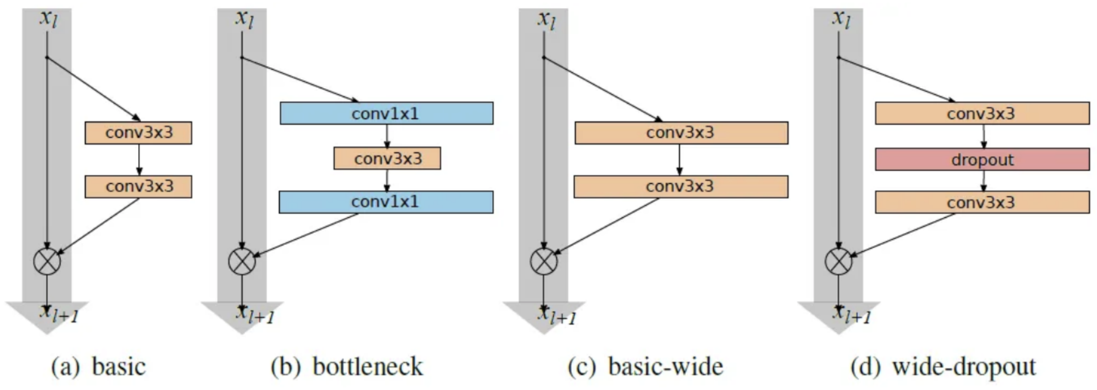
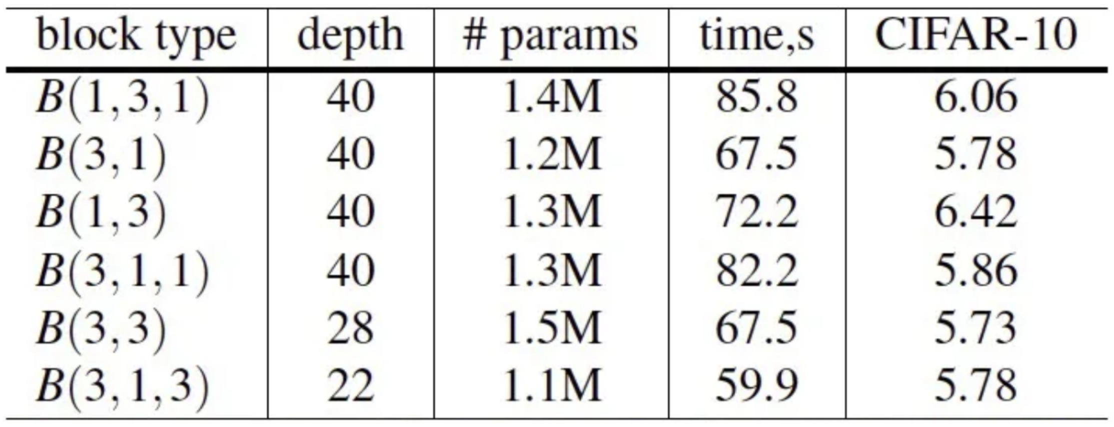


Figure 3.10: ResNet Blocks

These ResNet blocks are designed as follows:



block type	depth	# params	time,s	CIFAR-10
$B(1,3,1)$	40	1.4M	85.8	6.06
$B(3,1)$	40	1.2M	67.5	5.78
$B(1,3)$	40	1.3M	72.2	6.42
$B(3,1,1)$	40	1.3M	82.2	5.86
$B(3,3)$	28	1.5M	67.5	5.73
$B(3,1,3)$	22	1.1M	59.9	5.78

WRN-d-2 (k=2), Error Rate (%) in CIFAR-10 Dataset

- B(3;3): This is the original basic block depicted in the figure 3.10(a).
- B(3;1;3): It features an additional 1×1 layer positioned between two 3×3 layers.
- B(1;3;1): This block maintains the same dimensionality for all convolutions, creating a bottleneck.
- B(1;3): This network employs alternating 1×1 and 3×3 convolutions.
- B(3;1;1): This block is structured in a Network-in-Network style.

- B(3;1): This network follows a pattern of alternating 3×3 and 1×1 convolutions.

Going back to the research paper that how they used this WideResNet architecture to increase the robustness of their model.To begin with, their research paper provides a resounding affirmation through the utilization of the latest diffusion model, as introduced by Karras et al.[21] in 2022. This model, boasting superior efficiency and image quality in comparison to DDPM, forms the cornerstone of their investigation. Remarkably, their adversarially trained models establish a new benchmark for performance on RobustBench, solely relying on generated data. When subjected to the l_∞-norm threat model with ϵ set at 8/255, their models deliver remarkable results, achieving a robust accuracy of 70.69% on CIFAR-10 and 42.67% on CIFAR-100. This marks a substantial improvement over previous models, showcasing an impressive increase of +4.58% and +8.03%, respectively. to mention, even with the l_2-norm threat model with ϵ set to 128/255, their models excel, achieving an accuracy of 84.86% on CIFAR-10, representing a impressive gain of +4.44%. What's more, these outcomes outperform prior endeavors that incorporated external data sources.

Chapter 4

Experiments

In our research project, we conducted a series of experiments to assess the efficacy of various pruning algorithms and evaluated their impact on model accuracy. We used the CIFAR-10 dataset for both the pruning processes and adversarial attacks across six distinct trained models. Initially, our focus was on two pruning techniques: L1 [26] and random pruning [26]. To gauge the effectiveness of these pruning methods, we manipulated a single hyperparameter, sparsity, which ranged from 0.1 to 0.9, thereby pruning channels with varying probabilities.

Subsequently, we introduced a DDN (Decoupling Direction and Norm [43]) attack with a duration of 1000 steps, targeting the best-performing model after the pruning phase. The objective was to assess the robustness of these models in the face of adversarial attacks. We selected three models that exhibited robustness and three that did not, facilitating a comparative analysis of the attack's impact on different model types.

To facilitate transparency and reproducibility, we implemented our research using PyTorch, and the complete source code is available on our GitHub repository: https://github.com/Cinofix/pruning-robustness-2023.

4.1 Experimental Setup

Datasets

In our experimental design, we deliberately chose one distinct dataset with varying characteristics to introduce diversity and complexity into our setup. This approach aligns with the methodology proposed in previous research on poisoning attacks [37], [38], [44]. One of the datasets we used is the CIFAR-10 dataset [22], which is notable for its differences in data dimensionality, the number of classes, and class balance.

The CIFAR-10 dataset comprises 60,000 color images, each measuring 32x32 pixels, equally distributed across ten distinct classes. Our data split consists of 50,000 images allocated for training, 10,000 for validation, and another 10,000 for testing. During the training phase, we introduced data augmentation techniques such as random cropping and random rotation to enhance the model's robustness and generalization capabilities.

Models and Training phase

We assessed the performance of our pruning algorithms and the DDN attack across neural networks of varying sizes in our experiments. Specifically, we employed three robust models, Addepalli2022Efficient-ResNet18 [1], Sehwag2021Proxy-ResNet18 [52], and Engstrom2019Robustness-ResNet50[9], and for comparison, three non-robust models: Wang2023Better-WideResNet-28-10[57], ResNet-50 [9], and VGG-19 [60], each with different parameter configurations. Addepalli-Resnet18 and Sehwag-ResNet18 have approximately 11 million parameters, while Engstrom-ResNet50, similar to ResNet-50 (non-robust), boasts 23 million parameters.
Wang2023Better-WideResNet-28-10 is the largest among them, with 36 million parameters, and it is also non-robust. Lastly, VGG-19 comprises 20 million parameters.
All models underwent training on the CIFAR-10 dataset for 50 epochs, utilizing the SGD optimizer with a momentum of 0.9, weight decay set at 5e-4, and a batch size of 256. The optimization process focused on minimizing the cross-entropy loss, denoted as **L**. We also incorporated an exponential learning rate scheduler with an initial learning rate of 0.001 and a decay rate of 0.1 to facilitate the training process.
Furthermore, we subjected all models to the training phase employing two distinct pruning techniques: L1 and Random pruning. With each of these pruning methods, we systematically varied the sparsity level, spanning from 0.05 to 0.9, resulting in a total of ten models. During the training process, we implemented a checkpoint system, saving model states every five epochs. Subsequently, when it came to launching the attack, we selected the best-performing model based on its performance at a specific epoch in order to ensure rigorous evaluation and comparison.

Attack Setup

We conducted DDN (Decoupling Direction and Norm [43]) attacks, each comprising 1000 steps, on all six models, both robust and non-robust. Within each model, we performed attacks on various sub-models generated by varying the sparsity. It's essential to note that DDN relies on the l_2 norm. For the scope of this thesis, we kept the hyperparameters of DDN unchanged, with our primary objective being to assess the robustness of both pruned and non-robust models. Following the attacks on all models, we generated a JSON file that encompasses diverse metrics, including hashes, information about targeted attacks, accuracy, original success, adversarial success, ASR (Attack Success Rate), execution times, the number of forward and backward passes, distances, and instances of box failures. This comprehensive data set allowed us to thoroughly evaluate and compare the performance and robustness of the various models.To facilitate a more comprehensive comparison and gain deeper insights into the impact of different attack types on both robust and non-robust models, we conducted experiments involving the FMN attack, known as the Fast Minimum Norm attack [41]. This approach allowed us to delve further into how the behaviors of these models evolve when subjected to this specific attack type.

Performance Metrics

Following the training of our six models, each pruned at different levels, our next step involved subjecting these models to adversarial attacks to evaluate their robustness. We specifically focused on measuring two key aspects: distances and ASR (Attack Success Rate). By analyzing these metrics, we gained insights into the effectiveness of the attacks and the models' resilience to them.
Subsequently, we created plots that illustrate the relationship between sparsity levels and the distances observed across all models. These plots serve as a valuable visualization, offering insights into the success of the attacks and helping us identify which model exhibited the least resistance to adversarial attacks. Detailed results and graphical representations can be found in the experimental results section of our study.

4.2 Experimental Results

Params and Flops

Considering the model parameters and floating-point operations is crucial in understanding the impact of pruning on different models. Sparsity, in essence, represents a probability assigned to pruning methods to remove channels. For instance, a sparsity of 0.1 corresponds to cutting 10 percent of the channels, inevitably leading to a reduction in model accuracy. Our primary focus revolves around eliminating unnecessary neurons that contribute minimally to computations.
As we increase the sparsity level, more neuron layers are pruned, rendering the model more susceptible to vulnerabilities. Concurrently, with increasing sparsity, the parameters and FLOPs (Floating-Point Operations per Second) are also likely to decrease. The figures below illustrate the variations in parameters and FLOPs for the two pruning methods, L1 and Random, highlighting the distinct behaviors of each approach.
Furthermore, as depicted in Figure 4.1, we present the parameters (Params) and Floating Point Operations (Flops) of the robust models. As previously mentioned, our analysis involves three robust models: Addepalli (ResNet18), Sehwag (ResNet18), and Engstrom (ResNet50). Interestingly, the figure illustrates that the Random Pruning method outperforms L1 pruning. In theory, pruning more channels should lead to a decrease in Params and Flops. However, this behavior is notably observed in the Random method, whereas the L1 method tends to exhibit over-pruning.
Moving on to Figure 4.2, we shift our focus to the non-robust models (VGG19, ResNet50, and WideResNet) and their response to channel pruning. Remarkably, both methods exhibit similar behaviors in these non-robust models. However, it's worth noting an unusual behavior with the L1 method, particularly after pruning approximately 30% of the channels. This peculiar behavior can also be attributed to over-pruning.
Conversely, when examining the Random pruning method closely, one can observe that both Params and Flops tend to stabilize after pruning around 30% to 40% of the channels. Beyond this point, the neurons exhibit an unusual behavior as aforementioned in the over pruning, the neurons are now over pruned which means most of the layers that play important role in the computation are now pruned out, therefore models is corrupted and and shows unreliable results. In the case of

the Random method, they either remain relatively constant or, in the L1 method, they might even increase, which is quite unexpected. Ideally, one would expect either a constant or an exponentially decreasing trend.

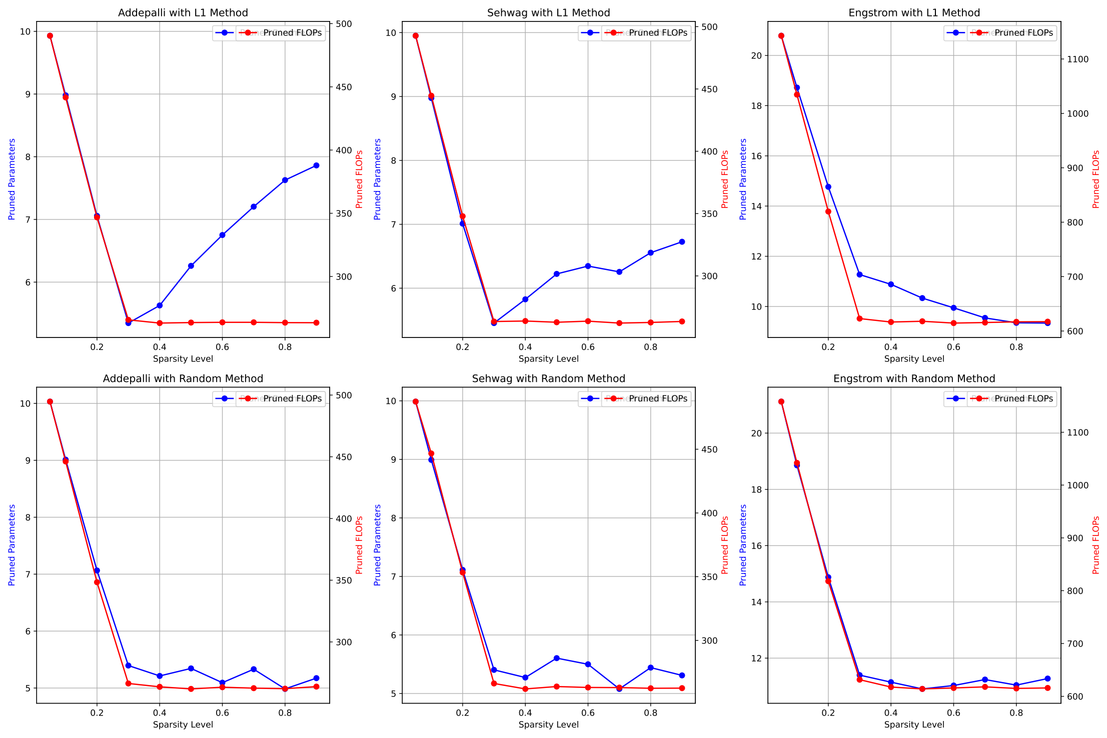


Figure 4.1: Params and Flops of Robust models

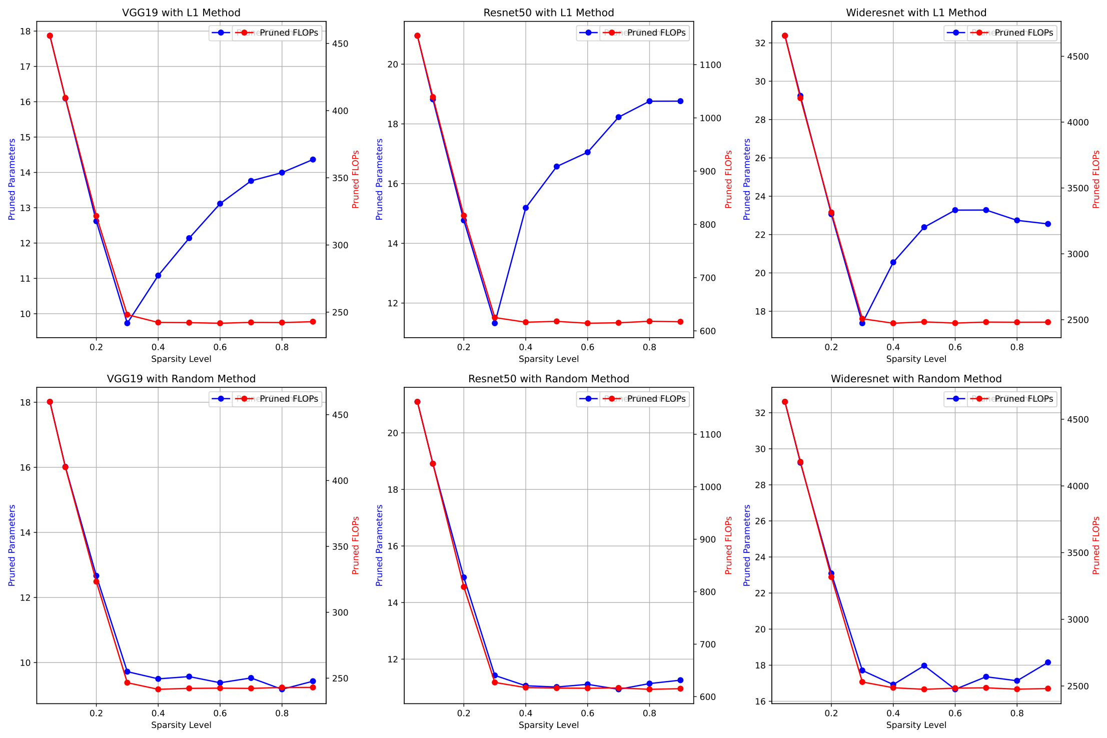


Figure 4.2: Params and Flops of Non-Robust models

Over-Pruning

In our previous experiment, we explored how the network's size impacts the prams and flops of two pruning methods. Now, we aim to investigate how these pruning methods perform when the network is pruned to an extreme degree, resulting in a substantial reduction in size, up to 99.6%. As before, we will compare the performance of L1 pruning and Random pruning. It's challenging to predict precisely how Random and L1 pruning will behave under such severe pruning conditions. However, we anticipate that Random pruning may perform slightly better than L1 pruning. Additionally, we suspect that Random pruning might lead to a decrease in accuracy due to the emergence of a bottleneck in the network. This bottleneck can occur when the random nature of pruning leads to the removal of most or all nodes in a layer, severely limiting the information flow to the subsequent network layer.

Figure 4.3 provides a comparison of the performance of Random pruning and L1 pruning. The left plot displays all 10 pruning iterations, ranging from 0.05 to 0.9, using the L1 method, while the right plot presents the same pruning iterations with the Random method. We exclude the last pruning iteration (1.0) to facilitate a more direct performance comparison, as the final drop in accuracy affects the y-axis scale. Initially, the performance of both pruning methods is highly similar, and for the first four pruning iterations (from 0.05 to 0.4), their performance remains nearly identical. Beyond this point, some variations emerge, though the performance still closely aligns. Notably, Random pruning exhibits a small accuracy peak at around 60% pruning. Further investigation is required to determine whether this is an anomaly or a characteristic of Random pruning.

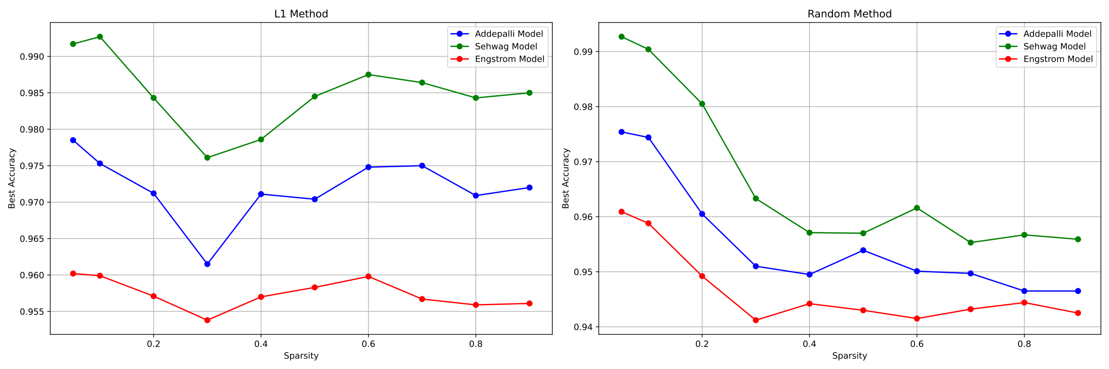


Figure 4.3: All Robust Model Best Accuracies against sparsity

In these experiments, we continue to work with a total of six models. In the figure above, we compare three robust models using two pruning methods. We train these models while increasing the sparsity and plot the best achieved accuracies. To maintain clarity, the green plot represents the Sehwag model, the blue plot corresponds to the Addepalli model, and the red plot represents the Engstrom model. It is evident that the Random pruning method consistently outperforms the L1 method. As expected, with increasing sparsity, the accuracy of the Random model decreases. However, it's worth noting that beyond a sparsity of 0.4, a bottleneck phenomenon occurs, causing unusual behavior in the model's accuracy. Despite this, it still outperforms the L1 method. On the other hand, the L1 method exhibits significant variation in accuracy beyond a sparsity of 0.4, resulting in poor performance.

In Figure 4.4, we present all non-robust models along with their best achieved accuracies across varying sparsity levels, ranging from 0.05 to 0.9. The red plot represents the WideResNet, the green plot corresponds to ResNet50, and the blue plot indicates VGG19. WideResNet achieves the highest accuracy, while VGG19 achieves the lowest. Comparing Figures 4.3 and 4.4, we can clearly observe the differences between robust and non-robust models. Their behavior is influenced by various factors, including the training process, parameters, and the number of layers used in the models.More specifically, how well their layers and neurons are dependent of each other.

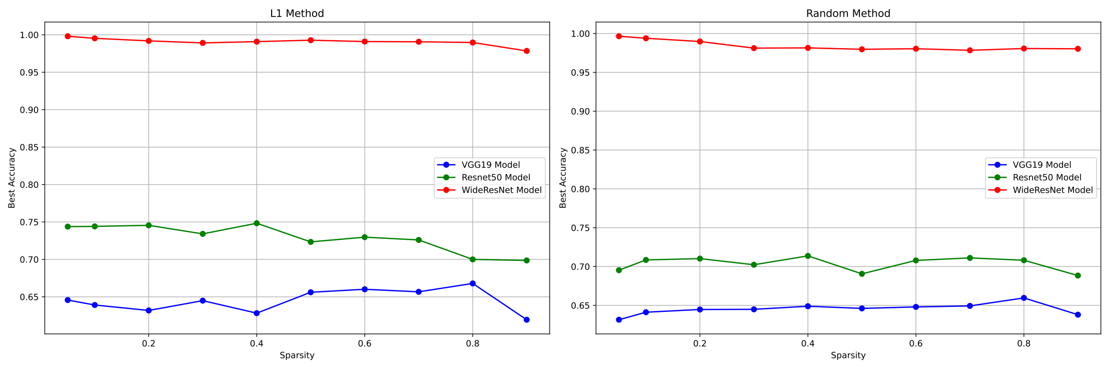


Figure 4.4: All Non-Robust Model Best Accuracies against sparsity

The consequences of over-pruning on these models are significant, as discussed in section 3.2.2 on Dependency Modeling. This section illustrates how the layers within these models are intricately interconnected. When we consider cutting off certain layers or, in essence, reducing the depth of the model by removing neurons or layers, it's crucial to understand that these layers and neurons depend on both their assessors and predecessors. If we decide to prune a neuron or layer, it entails cutting off its assessors and predecessors. However, these assessors and predecessors are also linked to their neighboring neurons and layers, which then necessitates attention to these neighboring components as well.

In essence, over-pruning has a profound drawback: it can permanently damage the models by severing critical connections and removing essential layers that contribute to the model's vulnerability and overall performance. This interdependence between layers and neurons underscores the importance of careful pruning techniques to maintain model robustness and functionality.

Performing DDN attack on Robust Models

One of the pivotal aspects of this thesis involves assessing the robustness of the pruned models. After subjecting all the models to L1 pruning and random pruning, the next step is to evaluate their robustness. Initially, we employ the DDN attack, also known as the "Decoupling Direction and Norm" attack, as elucidated in Section 2.3.3. This attack is conducted through 1000 iterations, and we closely examine its behavior by analyzing various distance metrics. Following the attack, we obtain four distinct distances for each sparsity level: l_0, l_1, l_2, and l_∞. We then calculate the median distance for each of these metrics and plot them to gain insights.

What's particularly noteworthy is that L1 and random pruning methods exhibit distinct behaviors when applied to robust models versus non-robust models. The figures presented below offer a clear visual representation of these differences between the two categories of models, which is explained below.

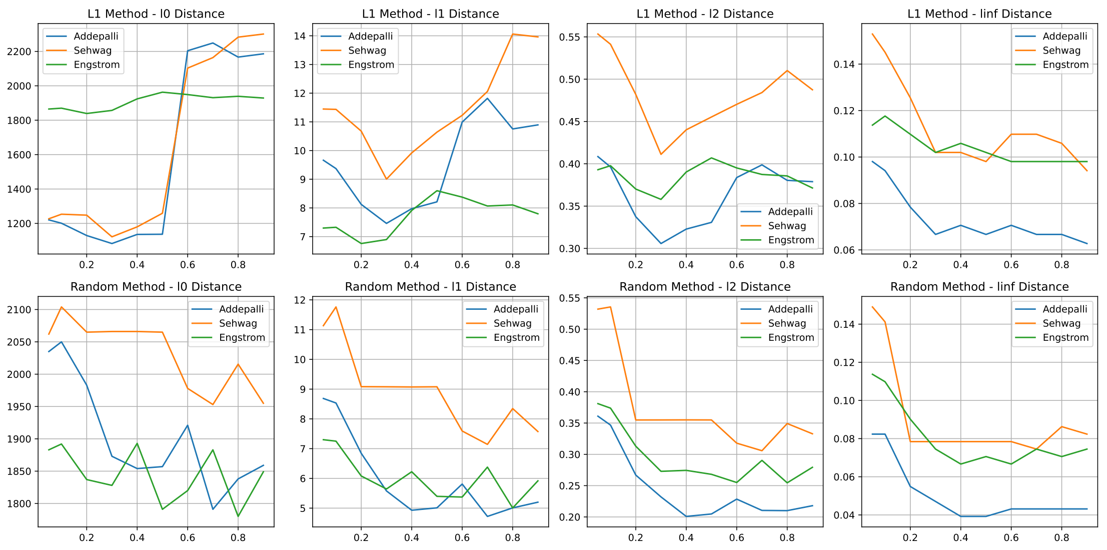


Figure 4.5: All Robust Model with L1 and Random Methods

Distances metrics: Distances like l_0, l_1, l_2, and l_∞ play a crucial role in the realm of adversarial attacks and defenses in machine learning, particularly with deep learning models. These metrics serve as valuable tools for quantifying the disparities between an initial input and a perturbed input, which is meticulously engineered to deceive a machine learning model. These distance measures are frequently employed to assess the effectiveness of adversarial attacks, such as the DDN and FMN attack, on a model's performance. Below, we'll delve into what each of these distances reveals about the nature of such an attack:
L_0 Distance (Hamming Distance): The L_0 distance serves as a gauge for quantifying the number of elements that exhibit disparities between two inputs. In the context of adversarial attacks, it essentially tells us the count of individual features that have been altered to construct an adversarial example. When the L_0 distance is small, it signifies that only a limited number of features have undergone modification. This can make the perturbation less conspicuous to human observers, yet it remains effective in misleading the model.The L_0 distance is simply a count of the number of differing elements between two inputs. Mathematically, it can be expressed as:

$$L_0(x, x') = \sum_{i=1}^{n} \mathbb{I}(x_i \neq x'_i) \tag{4.1}$$

Where:

- $L_0(x, x')$ represents the L_0 distance between input x and perturbed input x'.
- n is the total number of elements in the inputs.
- $\mathbb{I}(\cdot)$ is the indicator function, which returns 1 if the condition inside the parentheses is true and 0 otherwise.

- x_i and x'_i are the individual elements of the inputs x and x', respectively.

L_1 **Distance (Manhattan Distance):** L_1 distance calculates the absolute difference between each element of two inputs, aggregating the sum of these absolute differences for corresponding elements. In the context of adversarial attacks, a small L_1 distance indicates that the perturbed input closely resembles the original input in terms of the cumulative absolute differences between their elements.The L_1 distance is the sum of the absolute differences between corresponding elements of two inputs. It is represented as:

$$L_1\left(x, x'\right) = \sum_{i=1}^{n} |x_i - x'_i| \tag{4.2}$$

Where:

- $L_1(x, x')$ represents the L_1 distance between input x and perturbed input x'.
- n is the total number of elements in the inputs.
- x_i and x'_i are the individual elements of the inputs x and x', respectively.

L_2 **Distance (Euclidean Distance):** L_2 distance computes the square root of the sum of squared differences between corresponding elements of two inputs. In adversarial attacks, a small L_2 distance suggests that the perturbed input is akin to the original input concerning pixel-wise disparities. However, it is important to note that even when the L_2 distance is small, the perturbed input can still be adversarial.The L_2 distance is the square root of the sum of squared differences between corresponding elements of two inputs:

$$L_2\left(x, x'\right) = \sqrt{\sum_{i=1}^{n} \left(x_i - x'_i\right)^2} \tag{4.3}$$

Where:

- $L_2(x, x')$ represents the L_2 distance between input x and perturbed input x'.
- n is the total number of elements in the inputs.
- x_i and x'_i are the individual elements of the inputs x and x', respectively.

L_∞ **Distance (Infinity or Chebyshev Distance):** The L_∞ distance quantifies the maximum absolute difference between corresponding elements of two inputs. In the context of adversarial attacks, a small L_∞ distance indicates that the most significant individual difference between the original and perturbed inputs is relatively minor. This implies that the adversarial perturbation is not readily perceptible.The L_∞ distance is the maximum absolute difference between corresponding elements of two inputs:

$$L_\infty\left(x, x'\right) = \max_{i=1}^{n} |x_i - x'_i| \tag{4.4}$$

Where:

- $L_\infty(x, x')$ represents the L_∞ distance between input x and perturbed input x'.

- n is the total number of elements in the inputs.

- x_i and x'_i are the individual elements of the inputs x and x', respectively

When evaluating the impact of a DDN attack or any other adversarial attack on a machine learning model, these distance metrics come into play to assess how closely the perturbed input resembles the original input. Smaller distance values suggest that the attack has successfully generated an adversarial example that closely resembles the original input both visually and semantically. This is achieved while causing a misclassification or model failure. The use of these distances aids in quantifying the attack's effectiveness in terms of evasion and its potential implications for the model's reliability and security.

Figure 4.5 provides an overview of all the robust models employed in this thesis, emphasizing the repeated usage of L1 and Random pruning methods. In the figure, the first row is dedicated to L1 methods, while the second row corresponds to the Random methods. Each column signifies the distances l_0, l_1, l_2, and l_∞, respectively. Our observations indicate that as the distance metrics become smaller, signifying the increasing effectiveness of the attack, the L1 pruning method exhibits a peculiar behavior. Initially, the attack proves successful up to 0.4 sparsity, but beyond this point, it displays a resurgence, suggesting a weakening of the attack. Notably, the DDN attack exhibits considerable variation in results and loses its consistent trend.

Conversely, the Random method consistently delivers robust results. It adheres to a clear trend: as more channels are pruned, the DDN attack becomes progressively more potent. It is essential to highlight that while certain portions of the graph maintain constant trends, increasing sparsity ultimately weakens the model, rendering it more vulnerable to attacks.

Performing DDN attack on Non-Robust Models

Figure 4.6 illustrates, Non-robust models shows distinct trends compared to robust models. Due to their inherent vulnerability to adversarial attacks, non-robust models behave differently when subjected to the DDN attack. For example, the WideResNet models, under both L1 and Random pruning methods, follow a consistent trend of decreasing robustness as sparsity levels increase, albeit with occasional fluctuations. This pattern indicates that these non-robust models lack resilience even after pruning, making them highly susceptible to adversarial attacks.

Conversely, VGG19 and ResNet50 models demonstrate similar trends in distances l_1, l_2, and l_∞. However, they exhibit some variation in distance l_0. These observations underscore the extreme vulnerability of non-robust models to adversarial attacks, with their distances being consistently low and nearly constant with increasing sparsity. In summary, non-robust models are exceptionally susceptible to attacks and require effective defense mechanisms to bolster their security.

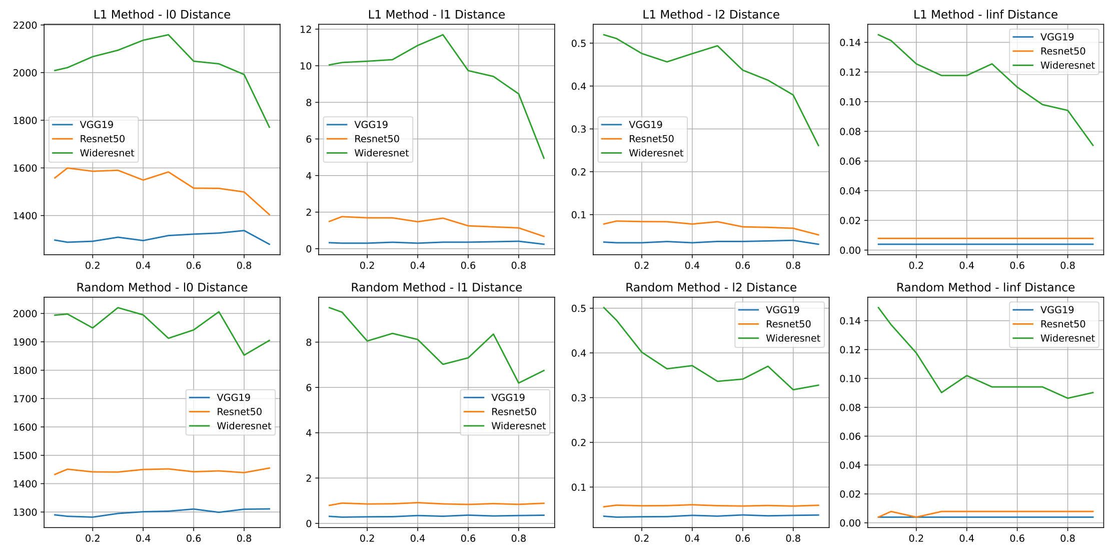


Figure 4.6: All Non-Robust Model with L1 and Random Methods

Performing FMN attack on Robust Models

For a more meaningful comparison, we also conduct an attack using FMN, which stands for Fast Minimum Norm attack. We use a specific hyperparameter setting where norm is set to $Float("inf")$. The FMN (Fast Minimum Norm) attack is a method used in adversarial attacks on machine learning models, especially deep neural networks. It aims to create adversarial examples that can mislead a model by perturbing the input data in a way that is imperceptible to humans but confuses the model's predictions.
When we consider infinity norm in the FMN attack, it means the attack focuses on maximizing changes within the l_∞ norm. This norm measures the maximum absolute difference between elements of two vectors, ensuring that no single change in the perturbation exceeds a certain threshold.
Here's how the FMN attack with $norm = Float("inf")$ works.

1. Start with the original input data.

2. Calculate the gradient of the model's loss concerning the input data.

3. For each input element, compute a perturbation that maximizes the l_∞ norm while keeping it small. Typically, this involves multiplying the gradient's sign by a small value (e.g., ϵ).

4. Add the calculated perturbation to the original data, creating an adversarial example.

5. Feed the adversarial example into the model and observe its prediction, aiming to create a different prediction from the correct classification.

6. The FMN attack can be iterative, applying smaller perturbations to make the adversarial example less noticeable while still causing misclassification.

By focusing on the l_∞ norm, the FMN attack aims to make minimal, imperceptible changes to the input data while effectively confusing the model. This helps assess model robustness and security.

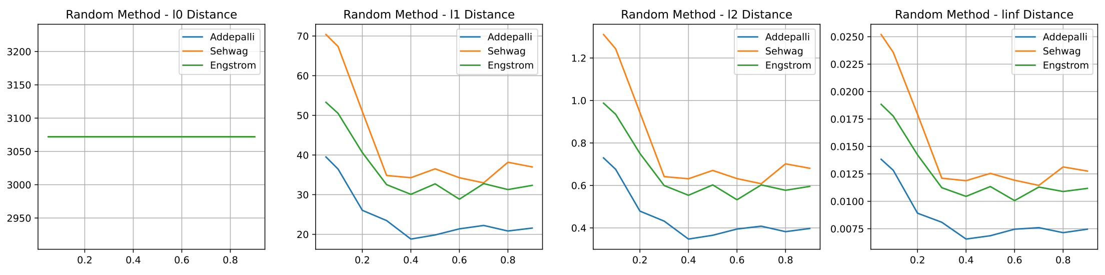


Figure 4.7: All Robust Model with Random Method

As depicted in Figure 4.7, the impact of Random pruning on model robustness when subjected to FMN attacks is evident. Notably, the trends in distances, specifically l_1, l_2, and l_∞, appear quite similar. This similarity can be attributed to the attack's norm configuration, which predominantly emphasizes the l_∞ distance. Interestingly, as we progressively prune more channels, the FMN attack grows in potency, consequently rendering the initially robust models more susceptible to adversarial attacks.FMN attack didn't consider the l_0 distance therefore, it didn't effect any of the model.

Performing FMN attack on Non-Robust Models

Comparing the outcomes of attacks on non-robust models using DNN and FMN approaches reveals an intriguing similarity. It appears that both DNN and FMN attacks exhibit analogous behavior when targeting non-robust models. In Figure 4.8, we observe that the WideResNet model continues to respond to the attack, but after pruning around 40%, its response stabilizes, indicating a consistent pattern. It's essential to acknowledge the influence of over-pruning in this context. In contrast, VGG19 and ResNet50 display no significant variation, highlighting the FMN attack's effectiveness in compromising non-robust models.

However, it's noteworthy that a substantial distinction emerges in the context of the l_0 distance when comparing robust and non-robust models. Each model exhibits its unique but consistent trend in this regard. From this observation, it can be inferred that the FMN attack does not give significant consideration to the l_0 distance.

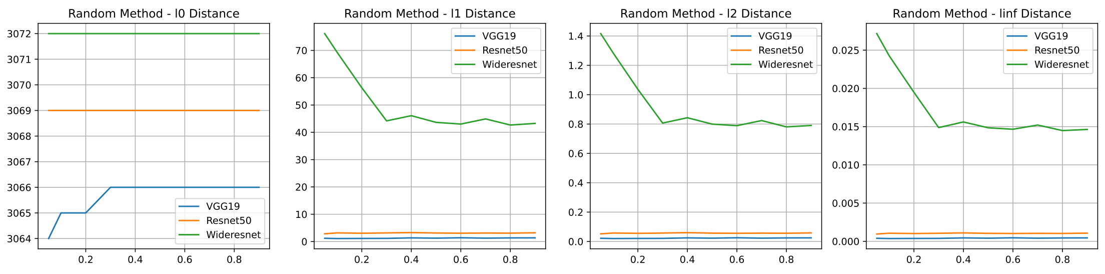


Figure 4.8: All Non-Robust Model with Random Method

Chapter 5

Conclusions and Future Work

In this research, we have delved into the realm of adversarial attacks, specifically focusing on the FMN (Fast Minimum Norm) attack, and its impact on both robust and non-robust machine learning models, all while considering the effects of pruning techniques and the DDN (Decoupling Direction and Norm) attack.Our study encompassed two key pruning methods, L1 pruning and Random pruning, which served as the foundation for exploring model robustness. Pruning filters from neural networks offers computational efficiency, but it also introduces vulnerabilities that can be exploited by adversarial attacks.The FMN attack, characterized by its emphasis on the l_∞ norm, was employed to craft adversarial examples, thereby evaluating model susceptibility to subtle input perturbations. By systematically adjusting sparsity levels through pruning, we scrutinized the ensuing effects on model robustness under the FMN attack.Our findings illuminate distinct behaviors between robust and non-robust models when subjected to the FMN assault. Robust models, designed to withstand adversarial attacks, exhibited varying trends in l_1, l_2, and l_∞ distances. Intriguingly, the l_0 distance displayed consistent trends, suggesting that the FMN attack may not prioritize it as a critical metric. This highlights the complexity of robust model evaluation in the presence of pruning.Conversely, non-robust models, inherently vulnerable to adversarial perturbations, showed more uniform trends in l_1, l_2, l_∞, and l_0 distances when confronted with the FMN attack. This observation underscores the efficacy of the FMN attack in targeting non-robust models and emphasizes the need for robustness-enhancing techniques, especially in safety-critical applications.In parallel, the study also shed light on the DDN (Decoupling Direction and Norm) attack, which is a crucial component in assessing model security. Future research endeavors should continue to explore pruning techniques, defense strategies, and advanced attack methodologies to further our understanding of model robustness and security.For the **future work** we can consider the exploration of adversarial attacks and model robustness that remains an evolving field with numerous avenues for future investigation.To enhance model security, it is imperative to develop and evaluate advanced adversarial defense mechanisms. Investigating the interplay between various attacks and defenses could yield insights into more robust models.Future research could delve into novel attack techniques beyond the FMN attack. Understanding the limitations and vulnerabilities of machine learning models under different attack paradigms is crucial. Assessing the transferability of adversarial attacks across different models and domains can provide valuable insights into the generalization of attack techniques. Applying adversarial robustness research to real-world applications, such as autonomous vehicles, medical diagnosis, and cybersecurity, is

an essential step toward deploying secure machine learning systems.Collaborations between machine learning experts, cybersecurity professionals, and domain-specific researchers can lead to more comprehensive and robust defense strategies against adversarial attacks.
In summary, the study of adversarial attacks and model robustness is pivotal for advancing the reliability and security of machine learning systems. Future work should continue to push the boundaries of knowledge in this domain and explore practical applications across various industries.

Bibliography

[1] Sravanti Addepalli, Samyak Jain, and R. Venkatesh Babu. Efficient and effective augmentation strategy for adversarial training. In *Proceedings of the 36th Conference on Neural Information Processing Systems (NeurIPS)*, 2022.

[2] Alfred V. Aho, Michael R. Garey, and Jeffrey D. Ullman. The transitive reduction of a directed graph. *SIAM Journal on Computing*, 1(2):131–137, 1972.

[3] N. Carlini and D. Wagner. Towards evaluating the robustness of neural networks. In *IEEE Symposium on Security and Privacy (SP)*, pages 39–57, 2017.

[4] Ting-Wu Chin, Ruizhou Ding, Cha Zhang, and Diana Marculescu. Towards efficient model compression via learned global ranking. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 1518–1528, 2020.

[5] Xiaohan Ding, Guiguang Ding, Yuchen Guo, and Jungong Han. Centripetal sgd for pruning very deep convolutional networks with complicated structure. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4943–4953, 2019.

[6] Xiaohan Ding, Tianxiang Hao, Jianchao Tan, Ji Liu, Jungong Han, Yuchen Guo, and Guiguang Ding. Resrep: Lossless cnn pruning via decoupling remembering and forgetting. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 4510–4520, 2021.

[7] Xin Dong, Shangyu Chen, and Sinno Pan. Learning to prune deep neural networks via layer-wise optimal brain surgeon. In *Advances in Neural Information Processing Systems*, volume 30, 2017.

[8] J. Duchi, S. Shalev-Shwartz, Y. Singer, and T. Chandra. Efficient projections onto the l1-ball for learning in high dimensions. In *Proceedings of the 25th International Conference on Machine Learning*, pages 272–279, 2008.

[9] Logan Engstrom, Andrew Ilyas, Shibani Santurkar, Dimitris Tsipras, Brandon Tran, and Aleksander Madry. Adversarial robustness as a prior for learned representations. *Machine Learning (stat.ML); Computer Vision and Pattern Recognition (cs.CV); Machine Learning (cs.LG); Neural and Evolutionary Computing (cs.NE)*, 2019.

[10] Gongfan Fang, Xinyin Ma, Mingli Song, Michael Bi Mi, and Xinchao Wang. Depgraph: Towards any structural pruning. *CVPR'23*, 2023.

[11] Shangqian Gao, Feihu Huang, Weidong Cai, and Heng Huang. Network pruning via performance maximization. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 9270–9280, 2021.

[12] Yiwen Guo, Anbang Yao, and Yurong Chen. Dynamic network surgery for efficient dnns. In *Advances in Neural Information Processing Systems*, volume 29, 2016.

[13] Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*, 2015.

[14] Song Han, Jeff Pool, John Tran, and William Dally. Learning both weights and connections for efficient neural network. In *Advances in Neural Information Processing Systems (NIPS)*, 2015.

[15] Song Han, Huizi Mao, and William J. Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. In *International Conference on Learning Representations (ICLR)*, 2016.

[16] Yihui He, Xiangyu Zhang, and Jian Sun. Channel pruning for accelerating very deep neural networks. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1389–1397, 2017.

[17] Yihui He, Ji Lin, Zhijian Liu, Hanrui Wang, Li-Jia Li, and Song Han. Amc: Automl for model compression and acceleration on mobile devices. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 784–800, 2018.

[18] Geoffrey Hinton, Oriol Vinyals, Jeff Dean, and et al. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2(7), 2015.

[19] P. A. Jensen and J. F. a. Bard. Operations research models and methods. 2003.

[20] Yongcheng Jing, Yiding Yang, Xinchao Wang, Mingli Song, and Dacheng Tao. Meta-aggregator: Learning to aggregate for 1-bit graph neural networks. In *ICCV*, 2021.

[21] Tero Karras, Miika Aittala, Timo Aila, and Samuli Laine. Elucidating the design space of diffusion-based generative models. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2022.

[22] A. Krizhevsky. Learning multiple layers of features from tiny images. Technical report, Technical Report, 2009.

[23] Jaeho Lee, Sejun Park, Sangwoo Mo, Sungsoo Ahn, and Jinwoo Shin. Layer-adaptive sparsity for the magnitude-based pruning. *arXiv preprint arXiv:2010.07611*, 2020.

[24] Namhoon Lee, Thalaiyasingam Ajanthan, Stephen Gould, and Philip HS Torr. A signal propagation perspective for pruning neural networks at initialization. In *arXiv preprint arXiv:1906.06307*, 2019.

[25] Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. Pruning filters for efficient convnets. *arXiv preprint arXiv:1608.08710*, 2016.

[26] Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. Pruning filters for efficient convnets. In *International Conference on Learning Representations (ICLR)*, 2017.

[27] Tailin Liang, John Glossner, Lei Wang, Shaobo Shi, and Xiaotong Zhang. Pruning and quantization for deep neural network acceleration: A survey. *Neurocomputing*, 461:370–403, 2021.

[28] Mingbao Lin, Rongrong Ji, Yan Wang, Yichen Zhang, Baochang Zhang, Yonghong Tian, and Ling Shao. Hrank: Filter pruning using high-rank feature map. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 1529–1538, 2020.

[29] Liyang Liu, Shilong Zhang, Zhanghui Kuang, Aojun Zhou, Jing-Hao Xue, Xinjiang Wang, Yimin Chen, Wenming Yang, Qingmin Liao, and Wayne Zhang. Group fisher pruning for practical network compression. In *International Conference on Machine Learning*, pages 7021–7032, 2021.

[30] Liyang Liu, Shilong Zhang, Zhanghui Kuang, Aojun Zhou, Jing-Hao Xue, Xinjiang Wang, Yimin Chen, Wenming Yang, Qingmin Liao, and Wayne Zhang. Group fisher pruning for practical network compression. In *International Conference on Machine Learning*, pages 7021–7032. PMLR, 2021.

[31] Songhua Liu, Kai Wang, Xingyi Yang, Jingwen Ye, and Xinchao Wang. Dataset distillation via factorization. In *Conference on Neural Information Processing Systems*, 2022.

[32] Songhua Liu, Jingwen Ye, Runpeng Yu, and Xinchao Wang. Slimmable dataset condensation. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2023.

[33] Zhuang Liu, Jianguo Li, Zhiqiang Shen, Gao Huang, Shoumeng Yan, and Changshui Zhang. Learning efficient convolutional networks through network slimming. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2736–2744, 2017.

[34] Jian-Hao Luo and Jianxin Wu. Neural network pruning with residual-connections and limited-data. In *The IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1458–1467, June 2020.

[35] Jian-Hao Luo, Jianxin Wu, and Weiyao Lin. Thinet: A filter-level pruning method for deep neural network compression. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 5058–5066, 2017.

[36] Xiaolong Ma, Geng Yuan, Sheng Lin, Zhengang Li, Hao Sun, and Yanzhi Wang. Resnet can be pruned 60×: Introducing network purification and unused path removal (p-rm) after weight pruning. In *2019 IEEE/ACM International Symposium on Nanoscale Architectures (NANOARCH)*, pages 1–2. IEEE, 2019.

[37] T. A. Nguyen and A. Tran. Input-aware dynamic backdoor attack. In *Advances in Neural Information Processing Systems, NeurIPS 2020*, 2020.

[38] T. A. Nguyen and A. T. Tran. Wanet - imperceptible warping-based backdoor attack. In *9th International Conference on Learning Representations, ICLR 2021*, 2021.

[39] Laurent Orseau, Marcus Hutter, and Omar Rivasplata. Logarithmic pruning is all you need. In *Advances in Neural Information Processing Systems*, volume 33, pages 2925–2934, 2020.

[40] Sejun Park, Jaeho Lee, Sangwoo Mo, and Jinwoo Shin. Lookahead: a far-sighted alternative of magnitude-based pruning. *arXiv preprint arXiv:2002.04809*, 2020.

[41] Maura Pintor, Fabio Roli, Wieland Brendel, and Battista Biggio. Fast minimum-norm adversarial attacks through adaptive norm constraints. In *Proceedings of the 35th Conference on Neural Information Processing Systems (NeurIPS)*, 2021.

[42] Alex Renda, Jonathan Frankle, and Michael Carbin. Comparing rewinding and fine-tuning in neural network pruning. *arXiv preprint arXiv:2003.02389*, 2020.

[43] Jérôme Rony, Luiz G. Hafemann, Luiz S. Oliveira, Ismail Ben Ayed, Robert Sabourin, and Eric Granger. Decoupling direction and norm for efficient gradient-based l2 adversarial attacks and defenses. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4322–4330. IEEE, 2019.

[44] A. Salem, R. Wen, M. Backes, S. Ma, and Y. Zhang. Dynamic backdoor attacks against machine learning models. *arXiv preprint arXiv:2003.03675*, 2020.

[45] Hadi Salman, Andrew Ilyas, Logan Engstrom, Ashish Kapoor, and Aleksander Madry. Do adversarially robust imagenet models transfer better? In *NeurIPS*, 2020.

[46] Hadi Salman, Andrew Ilyas, Logan Engstrom, Sai Vemprala, Aleksander Madry, and Ashish Kapoor. Unadversarial examples: Designing objects for robust vision. 2020.

[47] Hadi Salman, Saachi Jain, Eric Wong, and Aleksander Madry. Certified patch robustness via smoothed vision transformers. 2021.

[48] Victor Sanh, Thomas Wolf, and Alexander Rush. Movement pruning: Adaptive sparsity by fine-tuning. In *Advances in Neural Information Processing Systems*, volume 33, pages 20378–20389, 2020.

[49] Victor Sanh, Thomas Wolf, and Alexander Rush. Movement pruning: Adaptive sparsity by fine-tuning. In *Advances in Neural Information Processing Systems*, volume 33, pages 20378–20389, 2020.

[50] Shibani Santurkar, Dimitris Tsipras, Brandon Tran, Andrew Ilyas, Logan Engstrom, and Aleksander Madry. Image synthesis with a single (robust) classifier. *Computer Vision and Pattern Recognition (cs.CV); Machine Learning (cs.LG); Neural and Evolutionary Computing (cs.NE); Machine Learning (stat.ML)*, 2019.

[51] Shibani Santurkar, Dimitris Tsipras, and Aleksander Madry. Breeds: Benchmarks for subpopulation shift. 2020.

[52] Vikash Sehwag, Saeed Mahloujifar, Tinashe Handina, Sihui Dai, Chong Xiang, Mung Chiang, and Prateek Mittal. Robust learning meets generative models: Can proxy distributions improve adversarial robustness? In *Conference Paper at the International Conference on Learning Representations (ICLR)*, 2022.

[53] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. In *International Conference on Learning Representations*, 2014.

[54] Huan Wang, Can Qin, Yulun Zhang, and Yun Fu. Neural pruning via growing regularization. *arXiv preprint arXiv:2012.09243*, 2020.

[55] Huan Wang, Can Qin, Yulun Zhang, and Yun Fu. Neural pruning via growing regularization. *arXiv preprint arXiv:2012.09243*, 2020.

[56] Wenxiao Wang, Minghao Chen, Shuai Zhao, Long Chen, Jinming Hu, Haifeng Liu, Deng Cai, Xiaofei He, and Wei Liu. Accelerate cnns from three dimensions: A comprehensive pruning framework. In *International Conference on Machine Learning*, pages 10717–10726. PMLR, 2021.

[57] Zekai Wang, Tianyu Pang, Chao Du, Min Lin, Weiwei Liu, and Shuicheng Yan. Better diffusion models further improve adversarial training. In *International Conference on Machine Learning (ICML)*, 2023.

[58] Wei Wen, Chunpeng Wu, Yandan Wang, Yiran Chen, and Hai Li. Learning structured sparsity in deep learning. In *Advances in Neural Information Processing Systems (NIPS)*, 2016.

[59] Jiaxiang Wu, Cong Leng, Yuhang Wang, Qinghao Hu, and Jian Cheng. Quantized convolutional neural networks for mobile devices. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4820–4828, 2016.

[60] Jian Xiao, Jia Wang, Shaozhong Cao, and Bilong Li. Application of a novel and improved vgg-19 network in the detection of workers wearing masks. *Journal Name*, 2021.

[61] Tien-Ju Yang, Andrew Howard, Bo Chen, Xiao Zhang, Alec Go, Mark Sandler, Vivienne Sze, and Hartwig Adam. Netadapt: Platform-aware neural network adaptation for mobile applications. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 285–300, 2018.

[62] Yiding Yang, Jiayan Qiu, Mingli Song, Dacheng Tao, and Xinchao Wang. Distilling knowledge from graph convolutional networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020.

[63] Lewei Yao, Renjie Pi, Hang Xu, Wei Zhang, Zhenguo Li, and Tong Zhang. Joint-detnas: Upgrade your detector with nas, pruning and dynamic distillation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10175–10184, 2021.

[64] Jingwen Ye, Songhua Liu, and Xinchao Wang. Partial network cloning. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2023.

[65] Zhonghui You, Kun Yan, Jinmian Ye, Meng Ma, and Ping Wang. Gate decorator: Global filter pruning method for accelerating deep convolutional neural networks. In *Advances in neural information processing systems*, volume 32, pages 1–10, 2019.

[66] Zhonghui You, Kun Yan, Jinmian Ye, Meng Ma, and Ping Wang. Gate decorator: Global filter pruning method for accelerating deep convolutional neural networks. In *Advances in Neural Information Processing Systems*, volume 32, 2019.

[67] Ruichi Yu, Ang Li, Chun-Fu Chen, Jui-Hsin Lai, Vlad I Morariu, Xintong Han, Mingfei Gao, Ching-Yung Lin, and Larry S Davis. Nisp: Pruning networks using neuron importance score propagation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 9194–9203, 2018.

[68] Ruonan Yu, Songhua Liu, and Xinchao Wang. Dataset distillation: A comprehensive review. *arXiv preprint arXiv:2301.07014*, 2023.

[69] Yulun Zhang, Huan Wang, Can Qin, and Yun Fu. Aligned structured sparsity learning for efficient image super-resolution. In *Advances in Neural Information Processing Systems*, volume 34, pages 2695–2706, 2021.

[70] Hao Zhou, Jose Alvarez, and Fatih Porikli. Less is more: Towards compact cnns. In *European Conference on Computer Vision (ECCV)*, 2016.