SCHOOL OF COMPUTATION, INFORMATION AND TECHNOLOGY — INFORMATICS

TECHNICAL UNIVERSITY OF MUNICH

Bachelor's Thesis in Informatics

Formal Verification of Neural Networks using Linearity Grafting

Felix Jonathan Brandis

SCHOOL OF COMPUTATION, INFORMATION AND TECHNOLOGY — INFORMATICS

TECHNICAL UNIVERSITY OF MUNICH

Bachelor's Thesis in Informatics

Formal Verification of Neural Networks using Linearity Grafting

Formale Verifikation Neuronaler Netze mittels Linearisierung

Author:	Felix Jonathan Brandis
Supervisor:	Prof. Dr. Jan Křetínský
Advisor:	Stefanie Mohr
Submission Date:	15.09.2023

I confirm that this bachelor's thesis is my own work and I have documented all sources and material used.

Munich, 15.09.2023 Felix Jonathan Brandis

Abstract

The increasing deployment of Neural Networks in safety-critical scenarios paired with their black-box nature and the existence of adversarial examples has generated growing academic and industry interest in the robustness of Neural Networks. As an alternative to the cycle of presenting empirical attack and defense strategies, the field of Neural Network Verification has emerged. Due to the complexity of verifying the robustness of a Neural Network, various abstraction and simplification techniques have been proposed. Linearity Grafting as such a technique aims to facilitate verification while preserving the classification performance of a given baseline model by replacing a portion of the nonlinear activation functions with linear approximations. We study the reproducibility of the results reported in the original work introducing Linearity Grafting and find ambiguities in the description of the methods that make the reproduction of results only partially successful. We find that there is a tradeoff between preserving classification performance and enabling a higher verified robustness rate in grafted models. This tradeoff tends to be controllable by the relative weighting of the two graft neuron selection heuristics *insignificance* and *instability*. Possible variations in the methods to calculate these heuristics not addressed in the original work cause big differences in the performance of the grafted models. Furthermore, we study a modification of the grafting technique that preserves the weights of the baseline model to evaluate the potential of transferring verification results between the baseline model and the more similar grafted model. This modification does not deliver promising results. We also evaluate the transferability of adversarial examples between baseline and grafted models. The results do not suggest that attacking a model via its grafted counterpart is a promising strategy.

We offer a different interpretation of the originally reported results regarding the potential of Linearity Grafting: A possible explanation for the increased verified robustness and preserved classification accuracy observed in grafted models is a mismatch between problem difficulty and architectural complexity of the baseline model. We speculate that grafted models can only achieve these results if baseline models are overparameterized and unnecessarily complex and hard to verify. We hypothesize that the truly reachable verified robust accuracy depends on problem difficulty and Linearity Grafting only improves the verifiability by aligning model complexity to problem difficulty.

Contents

1. Introduction

Neural Networks provide the backbone technology for a vast range of application fields. The technological advances enabled by their performance in areas such as Natural Language Processing, Computer Vision, and Medical AI have led to Neural Networks being responsible for state-of-the-art results across all these disciplines and dominating more traditional approaches.
Increasingly, systems based on this technology are deployed in the real world, influencing safety-critical decisions in scenarios such as medical diagnosis, self-driving vehicles, autonomous robots, facial recognition, drug discovery, and more. Due to the black-box nature of Neural Networks and their size, with some examples having more than a trillion parameters [1], the evaluation and traceability of their decisions are heavily limited. As a result of this, a significant amount of research focuses on enabling a higher degree of explainability and aims to find semantic patterns in Neural Networks. Apart from improving the understanding of where decisions are coming from, it is desirable to be able to verify their correctness. At the moment, empirical measures of performance on test data are the primary indicator of how confident one can be in a network's ability to make good decisions. While in many scientific and safety-critical fields, empirical data is heavily relied upon and considered sufficient to reject or prove hypotheses, e.g., when reasoning about drug efficacy, the mathematical and abstract nature of Neural Networks motivates incorporating a more formal approach to their evaluation. Considering the constant search for better network architectures, data preprocessing techniques, and other parameters governing the learning process, Machine Learning research is dominated by empirical methods. Still, an increasing amount of work is being done on more theoretical and formally verifiable properties of the technology.
In light of the demonstration of the existence of adversarial examples, security-related questions have gained importance, as in these cases, Neural Networks exhibit unintuitive and unwanted behavior. The notion of robustness has become a desirable property with respect to which networks are being evaluated. A potential escape from the cycle of presented adversarial defenses and attacks breaking them motivates formal reasoning about the robustness of a network to adversarial perturbations as opposed to solely empirical evaluation.
Verifying robustness to attacks on test images for a network involves solving a highly

nonlinear optimization problem and is shown to be NP-complete in general. This motivates the search for techniques to alter the structure of networks, making them more optimized for verification while aiming to maintain comparable performance. This paper seeks to present, evaluate and discuss **Linearity Grafting (LG)** as such a technique. First presented in [2], LG linearizes a trained network around certain neurons (selected in some informed way) to enable higher verified robust accuracy rates compared to baseline models. The parameters of the linear approximations are optimized in a learning process. The adopted weights of the original model are also optimized with a smaller learning rate in this process (referred to as weight retraining). Specifically, we aim to provide insights into the following three main matters of interest:

I Reproducibility of results presented in [2]

II Influence of neuron selection heuristics and other grafting parameters on the performance of grafted networks

III Properties of the relationship between baseline and grafted network:

a) Effectiveness of grafting without weight retraining

b) Changes in verification status for individual test inputs

c) Transferability of adversarial examples

Point III is motivated by the optimistic conclusions in [2] regarding the facilitation of highly improved verification rates using LG. If there exists a scenario where the original network is hard to verify but has to be used in application, are there possibilities of (a) verifying properties for the grafted network and transferring them to the original network or (b) disproving properties (e.g. robustness) for the grafted network and transferring the respective counterexamples to the original network? Figure 1.1 illustrates the properties of interest of the relationship between the two networks. Formally describing the relationship and therefore bounding the difference between the two networks benefits from minimizing this difference, which motivates evaluating whether networks grafted without retraining weights can achieve useful results while preserving more similarity to the original networks.

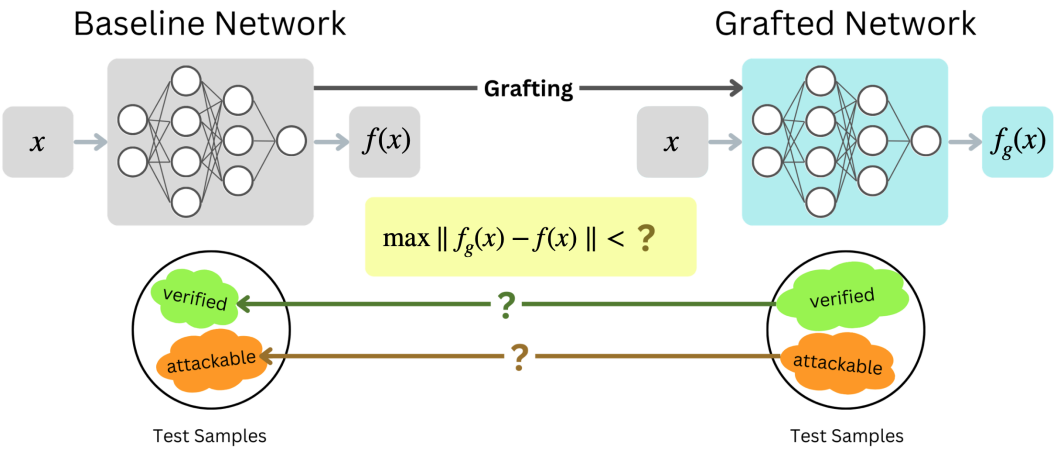

Figure 1.1.: Relationship between original and grafted network: Can the maximum difference of their output be bounded? Can successful verification or attack results be transferred back to the original network?

2. Preliminaries and Related Work

2.1. Neural Networks

Neural Networks are parametrized functions used in the field of Machine Learning (ML). They have proven to be hugely successful across a wide range of areas and are the underlying technology of a multitude of systems obtaining state-of-the-art results in fields associated with Artificial Intelligence (AI) such as computer vision and Natural Language Processing (NLP). Inspired by the brain neuron structure, a Neural Network consists of interconnected entities called neurons. Different architectural types describe the connection and dataflow patterns between neurons; architectures are adapted to the specific tasks at hand. Examples are Convolutional Neural Networks (CNN) in the context of computer vision and Recurrent Neural Networks (RNN) when handling sequential data, e.g. in NLP.

Considered is a canonical Feedforward Neural Network (FNN) classifier trained to differentiate between C classes. It consists of L layers of dimensionality d_j with $j \in \{1, ..., L\}$ and calculates a function $f : \mathbb{R}^n \to \mathbb{R}^C$, mapping the input vector $\boldsymbol{x} \in \mathbb{R}^n$ to a classification probability distribution over the classes. It holds that $d_0 = n$ and $d_L = C$. Let $k : \mathbb{R}^n \to [C]$ be the mapping of a given input to its true class. Let $\hat{k}(\boldsymbol{x})$ be the class assignment predicted by the classifier f, that is, the index of the highest component of $f(\boldsymbol{x})$.

A neuron's input value is calculated by forming a weighted sum over neuron output values of connected neurons from the preceding layer. The neuron's output value is calculated by passing the input value through a nonlinear activation function $\mathcal{A}(\cdot)$. In all cases considered here, the activation function used is the Rectified Linear Unit (ReLU) function, meaning $\mathcal{A}(x) = \max(0, x)$. Note that this notation will be abused to mean elementwise application in the case of a vector as an input.

The values in the last layer (also called logits) are fed into a softmax function to obtain a probability distribution as the output.

The function described by the Neural Network (NN) is parametrized by weight matrices $\boldsymbol{W}^{(i)} \in \mathbb{R}^{d_i \times d_{i-1}}$ and biases $\boldsymbol{b}^{(i)} \in \mathbb{R}^{d_i}$ with $i \in \{1, ..., L\}$. Graphically, the Structure of a FNN without the final application of a softmax function is illustrated in Figure 2.1. A

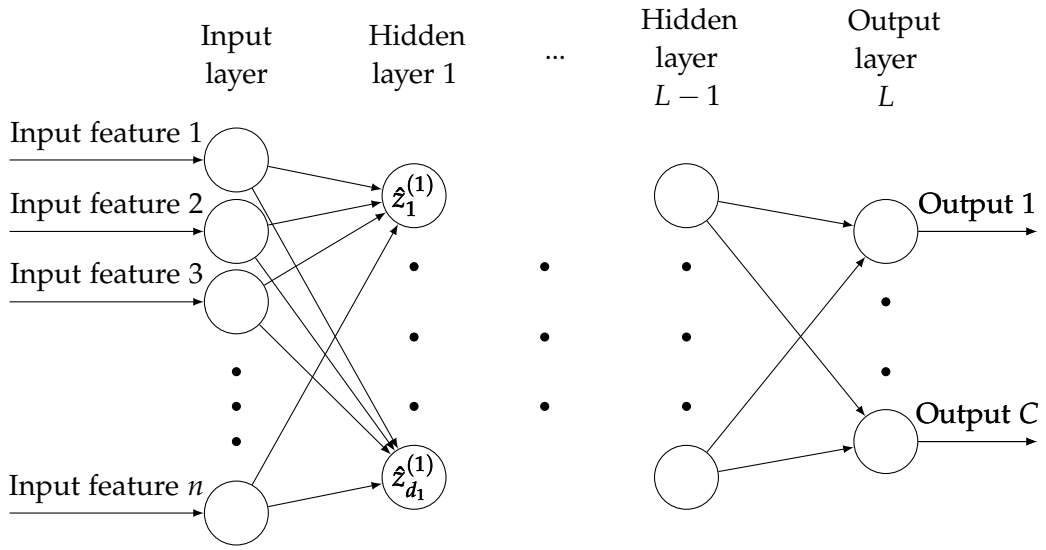

Figure 2.1.: Deep FNN structure excluding final application of softmax function, adapted from [3].

formal definition of the FNN is given in (2.1):

$$\begin{aligned} f(\boldsymbol{x}) &= softmax(\hat{\boldsymbol{z}}^{(L)}) \\ \hat{\boldsymbol{z}}^{(i)} &= \mathcal{A}(\boldsymbol{z}^{(i)}) \\ \boldsymbol{z}^{(i)} &= \boldsymbol{W}^{(i)}\hat{\boldsymbol{z}}^{(i-1)} + \boldsymbol{b}^{(i)} \\ \hat{\boldsymbol{z}}^{(0)} &= \boldsymbol{x} \end{aligned} \tag{2.1}$$

The power of a Deep Neural Network (DNN) (a Neural Network with $L > 2$ layers, i.e. more than one hidden layer) lies in its ability to model highly complex functions. The weights $\boldsymbol{W}^{(i)}$ are learned by minimizing a cost function J that compares the model's predictions with the true labels of training data. This minimization is often done using the technique of Stochastic Gradient Descent (SGD), as the contribution of the individual weights to the gradient of the cost function can be determined via backpropagation.
DNN models are able to autonomously identify important features from the data that help them make correct predictions. It is understood that earlier layers extract more basic features and later layers more complex ones, by detecting combinations of earlier layer's features.

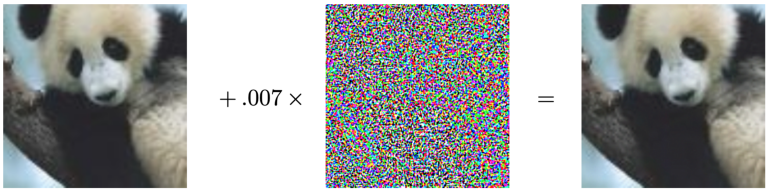

Figure 2.2.: Demonstration of an adversarial example attack on GoogLeNet [6] trained on the ImageNet dataset as shown in [5]

2.2. Adversarial Examples and Robust Training

An adversarial example in the context of a classification task is an incorrectly classified input created by applying small, intentional feature perturbations to an originally correctly classified input. The existence of adversarial examples and a simple and efficient technique to construct them were first shown in [4] and [5], as depicted in Figure 2.2. Changes imperceptible to the human eye leading to a changed and incorrect classification, often with very high confidence, is unwanted behavior, as intuitively, a classification model should have smooth decision boundaries and be robust to these changes.

Generally, adversarial examples have been shown to occur in various fields such as computer vision, NLP and cyberspace security as well as in the physical world [7]. Successful attacks have been performed on systems trained to handle tasks like image classification [5], [8], [9], semantic segmentation [10] and object detection [11], machine translation [12], text generation [13] and malware detection [14]. Furthermore, attacks have been transferred to the real world to successfully fool road sign and facial recognition models [15]–[18]. A multitude of methods to construct adversarial examples and strategies to create defense mechanisms have been presented and extensive reviews of those are done in [19], [20], [21] and [7].

Formally, an untargeted attack to construct an adversarial example in a classification task where a trained function f calculates class probabilities and a corresponding function $\hat{k}$ infers the class of a sample $\boldsymbol{x}$ can be described as an optimization problem [4]:

$$\boldsymbol{x}^* = \boldsymbol{x} + \arg\min_{\delta \boldsymbol{x}} \{ \|\delta \boldsymbol{x}\| \mid \hat{k}(\boldsymbol{x} + \delta \boldsymbol{x}) \neq \hat{k}(\boldsymbol{x}) \} \tag{2.2}$$

The norm $\|\cdot\|$ measures the difference between two points in the input space, often the l_p norm is chosen with $p \in \{1, 2, \infty\}$. The highly nonlinear nature of deep neural

networks makes the exact solution to this problem in most cases unfeasible to obtain. Many approximation techniques that generate satisfying results have been shown, of which a selection is briefly outlined in the Appendix.
The presented attack strategies assume a white-box setting in which the attacker has access to all information about the target model, such as the model weights, the training algorithm and the data distribution. Black-box settings assume a zero-knowledge or at least limited-knowledge attacker. A standard attack strategy in these scenarios is the creation of a local surrogate model, on which attacks are performed using white-box methods. Due to the transferability of adversarial examples first shown in [5], the obtained local adversarial examples are often effective in leading to misclassification by the black-box model. Although black-box settings are generally harder to tackle, successful attacks have been demonstrated in all settings and security through obscurity as a protection mechanism against the danger of adversarial examples should not be considered an adequate solution.

While research on attacking strategies has been continuously producing more powerful techniques, advancements in defense development have been relatively slow. This is by no means the result of a lack of defense strategy proposals, as the demonstration of adversarial examples has led to a long line of research into this area. Defense strategies can be categorized into three approaches: model modification, data modification, and the use of auxiliary tools [7].
Examples of model modification are defensive distillation [22], [23] feature squeezing [24] and mask defense [25]. Auxiliary tools are used in the examples of a defense-GAN [26], the proposed MagNet framework [27] and a high-level representation guided denoiser [28]. Data modification is done in the cases of data compression [29], data randomization [30] and different techniques of adversarial training.
Adversarial Training augments the dataset by creating adversarial examples and trains the classifier to classify them correctly. A challenge of generalizing models to be more adversarially robust is that usually more data and more complex architectures are needed to achieve good results [31]. Furthermore, the adversarially trained models often do not reach the generalization capabilities of traditionally trained models, as there seems to exist a tradeoff between maximizing robustness and accuracy. On the other hand, adversarially trained models tend to exhibit desirable properties other than robustness: they seem to learn representations that align more with human perception and are more semantically meaningful [32].
Many of the proposed defense techniques have failed to withstand slightly optimized attacks assuming knowledge of the defense strategy by the attacker. Examples are [33], [34], [35] and [36]. The fact that the majority of proposed defense strategies are not evaluated in a standardized manner putting them under the attack of state-of-the-art,

defense-informed attacks of different types is concerning [37] and further motivates formal reasoning about the robustness properties of neural networks.

2.3. Neural Network Verification

With the rising deployment of DNN's in safety-critical scenarios, being able to guarantee desirable properties of the systems in use becomes increasingly important. Examples of wanted properties are robustness, correctness, fairness, and monotonicity. In the following, robustness to adversarial examples will be the focus.
Evaluating a system by empirically measuring the success rate of state-of-the-art adversarial attacks can not give guarantees about the robustness, in particular not towards potential new and more powerful attacks. As outlined above, many heuristic defenses were shown to collapse under attacks targeted to break them. Verification aims to prove the impossibility of any successful attack under certain assumptions like the maximum perturbation magnitude.
Given a norm $\|\cdot\|$ (if not stated otherwise, the l_∞ norm is used) and a maximum distance ϵ we can define a perturbation set $\mathcal{P}$ around the sample $\boldsymbol{x} \in \mathbb{R}^d$ as shown in (2.3):

$$\mathcal{P} = \{\boldsymbol{x'} \in \mathbb{R}^d \mid \|\boldsymbol{x'} - \boldsymbol{x}\| \leq \epsilon\} \tag{2.3}$$

In a simple 2-class classification setting with a classifying function $f : \mathbb{R}^d \to \mathbb{R}$ and the decision boundary at $f = 0$, assuming that the sample $\boldsymbol{x}$ is correctly classified to be in class 1 with $f(\boldsymbol{x}) > 0$, then f^* as defined in (2.4) gives a definitive answer about the robustness of f around $\boldsymbol{x}$: If $f^* > 0$, robustness is verified. If $f^* < 0$, a counterexample exists and f is attackable for the input $\boldsymbol{x}$ within the ϵ-radius. For $f^* = 0$, the result depends on the specific implementation for decision boundaries.

$$f^* = \min_{\boldsymbol{x'} \in \mathcal{P}} f(\boldsymbol{x'}) \tag{2.4}$$

The verification problem is extended to a multiclass setting in (2.5) with classes C and the class probability prediction f_c for class c. Similarly, robustness is verified if $g^* > 0$ holds and vice versa:

$$g^* = \min_{\boldsymbol{x'} \in \mathcal{P}} \left(f_{c_{true}}(\boldsymbol{x'}) - \max_{\substack{c \neq c_{true} \\ c \in C}} f_c(\boldsymbol{x'}) \right) \tag{2.5}$$

For the sake of simplicity, in the following, f^* is used to describe the minimum value of interest for verification problems in all settings (2-class, multiclass, etc.). Complete verifiers aim to find a definitive answer to the question of whether $f^* > 0$,

while incomplete verifiers compute a lower bound ${f^*}_{lower} \leq f^*$ for f^*. In the case of ${f^*}_{lower} > 0$, the input is verified. The case ${f^*}_{lower} \leq 0$ however, makes no statement possible.
Verification algorithms can be categorised into different categories: Initial approaches of solving the presented highly non-linear optimization problems involved reformulating them into Mixed Integer Linear Programming (MILP) [38], [39] or Satisfiability Modulo Theories (SMT) [40]–[42] problems and using off-the-shelf solvers to tackle them. While these solvers enable complete verification, their limit is the exponentially rising cost with an increased number of neurons. As shown in [43] and [42], verifying the robustness of a ReLU network in general is NP-complete. Other approaches like CROWN [44] and DeepPoly [45] are based on the concept of bound propagation, enabling efficient incomplete verification. CROWN uses linear bounds to relax activation functions and propagate these bounds backward through the network. While these incomplete approaches significantly reduce the cost, they often produce vacuous and unusable bounds.
In this work, the state-of-the-art α-β-CROWN complete verifier is used. It combines a heavily GPU-parallelizable Branch and Bound (BaB) approach with an optimizable bound propagation technique (α-CROWN [46]), optimizable constraints (β-CROWN [47]) and a strong BaB-based falsifier. Due to the parallelization and optimizable bounds and constraints, this verifier is often able to produce useful bounds for making verifiability statements.

2.4. Pruning

Pruning is a technique originally explored to compress NNs and get rid of redundant parameters with the goal of reducing computational and storage cost while maintaining model performance [48], [49]. This can be done in an unstructured way, eliminating individual neurons and parameters that are insignificant according to some heuristic. Alternatively, structured pruning aims to get rid of whole components of the model to achieve hardware-friendly networks with reduced memory and computation needs.
Model compression and enforced weight sparsity have been proposed as adversarial defense strategies [25], [50], [51]. In [52], a pruning technique aware of the robust training objective is presented. Similarly, using pruning to reduce the resource intensity of a model while preserving robustness is investigated in [53]–[57].

2.5. Unstable Neurons

The non-linearity of activation functions is the reason for both the expressiveness of Neural Networks and the complexity of verifying them. In the verification process, for an input $\boldsymbol{x} \in \mathcal{P}$, lower and upper bounds $l_j^{(i)}$ and $u_j^{(i)}$ for the pre-activation value $z_j^{(i)}$ of a neuron are propagated and tightened. Given a ReLU neuron where $\mathcal{A}(z_j^{(i)}) = \max(0, z_j^{(i)})$ it is called unstable under input $\boldsymbol{x}$ if $l_j^{(i)} \leq 0 \leq u_j^{(i)}$. This is due to the unpredictability of its output value $\hat{z}_j^{(i)}$. For $u_j^{(i)} < 0$ or $l_j^{(i)} > 0$ on the other hand, the input lies in one of the linear ReLU regions only and it holds that $\hat{z}_j^{(i)} = 0$ or $\hat{z}_j^{(i)} = z_j^{(i)}$ respectively.

2.6. Linearity Grafting

Linearity Grafting [2] is a technique to modify a trained neural network. There are two main goals: Preserving the performance of the network (measured in terms of classification accuracy) and altering its structure in a way that makes it easier to be verified. As outlined in the section above, the complexity of the verification process comes from the nonlinear nature of Neural Networks and resulting necessary case distinctions when propagating bounds through the layers. These case distinctions grow exponentially in number and occur at the nonlinear parts of the Neural Network: the activation functions. To reduce the number of case distinctions, Linearity Grafting replaces a certain fixed percentage of these activations (default value is 50%) with linear approximations. If after this modification, the output of a neuron is passed through a linear approximation of the nonlinear activation function, we say that the neuron has been grafted.
Selecting which neurons to graft aligns with the earlier mentioned goals: Preserving the model performance is best achieved if neurons that don't contribute much to the final performance are chosen. The heuristics to measure this contribution and create a significance score will be discussed later. On the other hand, the verification process is simplified most if neurons that lead to case distinctions most often (unstable neurons) are chosen. Figure 2.3 illustrates Linearity Grafting graphically.
In [2], Linearity Grafting is proposed as a multistep process: As a first step, a DNN is trained using a relatively cheap adversarial training technique to robustify the network empirically without increasing the training cost significantly. Specifically, the Fast Adversarial Training (FAT) method proposed in [58] and further improved in [59] is used. Secondly, given a grafting ratio, the neurons to prune are identified. The goal is to graft insignificant and unstable neurons; different heuristics to measure these criteria

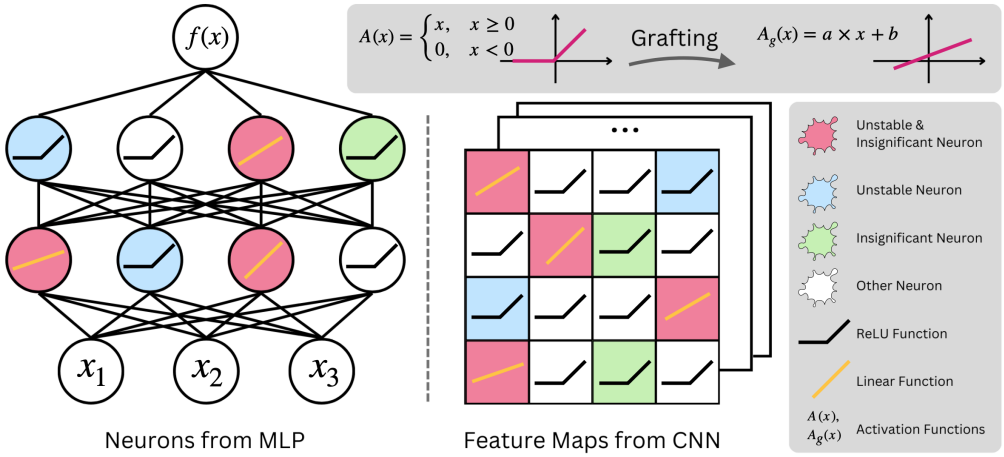

Figure 2.3.: Illustration of *Linearity Grafting*, which replaces nonlinear activation functions of certain neurons with linear approximations. Adapted from [2]

and weigh them are considered. After linearizing the identified neurons, the slope and intercept parameters of the linearizations are tuned to optimize model performance. The model weights are also retrained with a smaller learning rate in this process. The last step is verifying the resulting network with a complete verifier. Experimentally it can be shown that the injected linearity helps to improve certifiable robustness ratios. Linearity Grafting can be seen as a generalization of Neuron Pruning: pruned neurons behave the same way as grafted neurons with the restriction of slope a and intercept b of the linear approximation being equal to 0.

3. Experiments

Generally, the selection of datasets, architectures and training and verification hyperparameters was guided by the original experimental design in [2]. The evaluation metrics have also been adopted. The main differences are adjustments in the timeout length in verification experiments as well as the experiments on grafting without retraining. Further details can be found in the respective sections.

3.1. Datasets and Architectures

Experiments are done on the MNIST [60] and CIFAR10 [61] datasets on two different architectures: (i) a 4-layer CNN on CIFAR10 referred to as **CNN-B** proposed in [62], and (ii) a 7-layer convolutional network on both MNIST and CIFAR10 adapted from [63] called **ConvBig**.

3.2. Evaluation Metrics

Results are evaluated by means of the following metrics: **Standard Accuracy (SA)**, **Robust Accuracy (RA)**, **Verified Accuracy (VA)** and **Unstable Neuron Ratio (UNR)**.
SA% is the percentage of correctly classified samples by the model evaluated on the complete test set. This is often the single most important indicator for the quality of a trained model and is often just referred to as Accuracy.
RA% is the percentage of samples that were classified correctly and attacked unsuccessfully evaluated on the complete test set. The attack method is PGD-100 [64] with 100 restarts and the attack radius (maximum l_∞ perturbation) is $\epsilon = \frac{2}{255}$ for CIFAR10 and $\epsilon = 0.1$ for MNIST.
VA% is the percentage of verifiably robust samples evaluated on the first 1000 samples due to its high computational cost. Generally, RA is an upper bound for VA, although there might be slight deviations due to the difference in the set of samples that the metrics are calculated on. Figure 3.1 displays the individual contribution to these metrics by the size of the four mutually exclusive sets that an input sample can be added to in the verification procedure.
The **UNR%** is the mean of the percentage of unstable neurons over all input samples

that reach the BaB verification stage. The unstable neurons are determined in the verification process and therefore no instability masks for the network are created for samples that are misclassified, successfully attacked in the first place or verified by the incomplete verifier.

3.3. Verification Process and Parameters

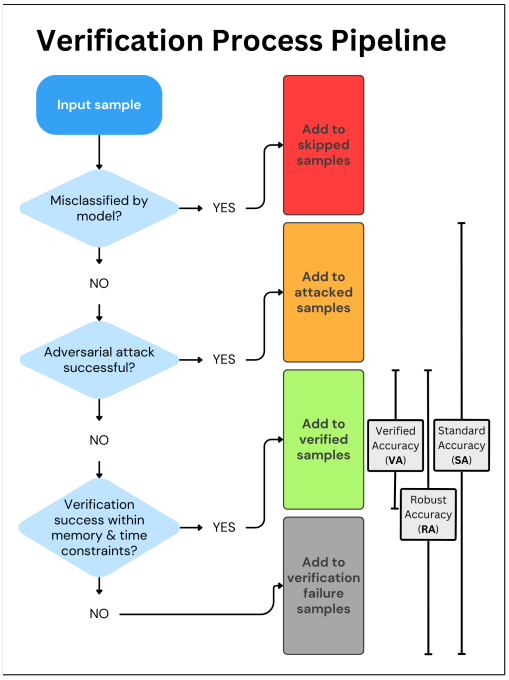

Figure 3.1.: Visualisation of the verification process and the evaluation metrics. The third step is in reality subdivided into the incomplete verifier and BaB stage.

As in [2], the current state-of-the-art α,β-CROWN complete verifier ([44], [46], [47], [65]), was used to verify the trained networks. In some places, modifications to the code were made to allow for the extraction of more detailed results from the process.

Verifying the robustness of a network refers to a procedure done repeatedly on the individual elements of a set of test inputs. For every input, the verifier will output one of the four possible results: **(1) Skipped Sample** - the verifier will abort the verification process for this sample, as the original input was already misclassified by the model. **(2) Attacked Sample** - the verifier will abort the verification process for this sample, as an adversarial example was successfully found, making robustness verification impossible. **(3) Verification Success** - the verifier was able to verify robustness within the given time and memory constraints. This is usually first tried by the means of an incomplete verifier and, if unsuccessful then continued in the BaB stage. **(4) Verification Failure** - the verifier was unable to find an adversarial example or to verify robustness for the input within the given time and memory constraints. The described 4-step process is depicted in Figure 3.1.

A quantitative, summarising analysis of the individual results is then used to deduce or hypothesize about the general behavior of the network concerning robustness. Whether

the conclusions drawn from these individual results are meaningful heavily depends on the size of the test set and its ability to accurately reflect the normally occurring input distribution.
Many hyperparameters govern and influence the verification process. In the presented results, the timeout length for the BaB stage was set to 180s (as opposed to 300s in [2]) to facilitate bigger test sets and more experiments. This is justified by results presented in [2] (Figure A9) showing that due to the exponential complexity of the verification problem, improvements in VA stagnate quickly with the augmentation of timeout length, especially for grafted networks.
If not indicated otherwise, verification results refer to a test set size of 1000. Other parameters like maximum domain size, branching heuristics, PGD-100 attack restarts, verification abortion bound threshold and many more were left unchanged compared to [2].
A more exhaustive search for the optimal parameters for the respective architecture, network size and dataset might have led to improved or different verification results.

3.4. Grafting Mask Creation

To indicate the neurons whose activation functions will be replaced by linear approximations, grafting masks need to be created. These masks and the weights of the trained net are passed into the grafting process, which creates the modified grafted architecture, learns the best parameters for the linear approximations and readjusts the given weights.
The selection of neurons to graft depends on the following parameters: grafting mode (global or local), grafting ratio, importance indicators, instability indicators, the normalization procedure of scores and their combination into the grafting score via a weighting value γ.

The **grafting mode** specifies whether for each layer the specified ratio of neurons is grafted (local) or whether the selection is done on a global level. This distinction is traditionally done for pruning techniques and while global pruning sometimes achieves better results, it risks the degeneration of the network architecture. Although implemented in the provided code, this distinction is not mentioned in [2]. As an inspection of the provided mask reveals varying per-layer grafting ratios, it is assumed that the presented experiments were conducted using the global grafting mode.

In [2], different **grafting ratios** and their effect on the tradeoff between SA and VA are evaluated, often finding the best results at a ratio of 50%. This was adopted and is the

standard grafting ratio for all presented results if not mentioned otherwise.

Importance indicators are heuristically calculated, unnormalized per-neuron values that aim to reflect the importance of a neuron for the network to function well. Three different heuristics to determine the importance inspired by pruning neuron selection algorithms are presented in [2]: learned importance scores (Hydra) [52], mean absolute neuron activation magnitude (SAP) [51] and mean absolute neuron activation gradient (GAP) [66].
An implementation to calculate these indicators given a trained model is provided for two of them: SAP and GAP. The indicators are calculated on a randomly selected subset of the training set. The size of the subset for all conducted experiments is 1000, it is assumed to have been the same for the experiments in the original publication as this is the value that can be found hardcoded in the implementation. Additionally, with the goal of comparing the difference in performance of grafted networks depending on the importance heuristic, a new importance heuristic was implemented in this paper: neuron activation magnitude variance, which will be referred to as VAP.
For a subset S of the training data with elements $\boldsymbol{x}_j$, the SAP importance indicator value v_i for neuron i is calculated as shown in (3.1). Note that the notation $z_i(\boldsymbol{x}_j)$ is abused to indicate the scalar pre-ReLU neuron activation values for neuron i on input $\boldsymbol{x}_j$, not to be confused with $\boldsymbol{z}^{(l)}$ referring to the activation value vectors of layer l as defined in (2.1).

$$v_i = \frac{1}{|S|} \sum_{\boldsymbol{x}_j \in S} |z_i(\boldsymbol{x}_j)| \tag{3.1}$$

GAP and VAP are calculated accordingly. The effect of varying the importance heuristic is discussed in the respective results section.

In [2], no implementation to calculate **instability indicators** of neurons is provided. This limits the reproducibility of the results. It is specified that the indicators are calculated from the number of inputs for which a given neuron is unstable. It is assumed that the total number of inputs is 1000, equal to the default verification test set size. For the presented results here, instability masks created during the BaB stage of verification were saved and later utilized to calculate the number of unstable neurons per image. As only some of the inputs go to the BaB stage, the instability indicator is only based on this fraction of the test set, whose size lies between 20% and 40% of the test set size for the presented experiments.
The UNR of the provided CIFAR10 **CNN-B** network calculated with this method matches the one presented in [2] exactly (15.85%). This and the fact that the UNR of the self-trained MNIST and CIFAR10 **ConvBig** take on similar values as the models using

	Description	**Example** (ranks, normalized scores)
min-ranking	Ties are assigned equal ranks, ranks are the minimum rank for the whole tied group	$[1, 2, 2, 4, 4, 4]$, $[0, 0.33, 0.33, 1, 1, 1]$
dense-ranking	Ties are assigned equal ranks, ranks are assigned without gaps	$[1, 2, 2, 3, 3, 3]$, $[0, 0.5, 0.5, 1, 1, 1]$
ordinal-ranking	Ties are assigned different ranks in order of appearance	$[1, 2, 3, 4, 5, 6]$, $[0, 0.2, 0.4, 0.6, 0.8, 1]$
norank	Normalization of scores without ranking	– , $[0, 0.4, 0.4, 1, 1, 1]$

Table 3.1.: Different ranking algorithms. The Example column shows ranks and normalized scores for the exemplary indicator values $[10, 20, 20, 50, 50, 50]$.

the same architecture presented in [2] (ours: 33.41%, 17.27% respectively, [2]: 31.27% and 17.7% respectively) gives somewhat confidence that our method for extracting unstable nodes in the verification process matches the one used in [2].

The **normalization process** described in [2] consists of two steps: First, the indicator values (for importance and instability) are each ranked and then their rank is normalized to the range $[0, 1]$. This results in significance scores $r_s^{(i)}$ and instability scores $r_u^{(i)}$ where i is the neuron index. The code provided by [2] does not include an implementation for this procedure. Specifically, it is not defined how indicators are ranked and why this is done. Ranking indicators leads to an information loss about their relative difference. Furthermore, while importance indicators take on many distinct values, instability indicators only take on integer values as they represent a count of inputs for which the respective neuron is unstable. For a verification test set size of 1000 and 40% of inputs reaching the BaB stage, instability indicators can only take on 400 distinct values. For networks with several thousand neurons (e.g. CIFAR10 **ConvBig** with 62464 neurons), the specific ranking algorithm has a big influence of the distribution of scores, as the treatment of ties becomes very important. Four different ranking algorithms and their effect on neuron selection are evaluated. The different algorithms are outlined in Table 3.1.

Normalization of ranks is done by linear rescaling to the range $[0, 1]$, i.e. unnormalized ranks $r^{(i)} \in R$ are rescaled as shown in (3.2).

$$r^{(i)} \mapsto \frac{r^{(i)} - \min(R)}{\max(R) - \min(R)} \tag{3.2}$$

Differences in neuron selection based on importance heuristic and ranking algorithm

are presented in the results section.

Combining the significance and instability scores $r_s^{(i)}$ and $r_u^{(i)}$ into a **grafting score** is done with a decaying weighting factor γ. Neurons are picked via evaluating the expression $\arg\max_i \gamma \times r_u^{(i)} - r_s^{(i)}$. In [2] it is stated that the best results are achieved in a multistage process where γ takes on different, decaying values. For a given grafting ratio g, n stages and different values $\gamma_1, ..., \gamma_n$, in every stage l, the top $\frac{g}{n} \times 100\%$ of remaining neurons according to the grafting score $\gamma_l \times r_u^{(i)} - r_s^{(i)}$ are picked. The percentage refers to the total neuron count. According to [2], best results are achieved in a 4-stage process where γ takes on the values $[2, 1.5, 1, 0]$. A 'linear decay' is mentioned, but the above values that do not decay linearly are explicitly stated, which introduces room for ambiguities. No implementation, which could clear these ambiguities, is provided. For all presented results, unless stated otherwise, the explicitly stated values from above for γ are used for mask creation. The effect of different γ values on neuron selection is discussed in the results section.

Generally, the above-mentioned uncertainties hindered the reproduction of results significantly. Upon requesting clarification of the precise procedure of creating grafting masks from the authors of [2] we received no answer. The fact that some crucial information necessary to accurately replicate the mentioned processes was missing from the paper introduced a lot of work overhead to reverse-engineer the methods applied in [2]. The missing information could not fully be deduced from implementations, as some processes lacked implementation in the provided code (instability score extraction, ranking and score combination). This was further complicated by the sparse documentation and the fact that only one of the trained models was provided with its grafting mask and grafted counterpart. Furthermore, the provided example scripts to self-train and then graft a network relied on a fixed grafting mask. Grafting a self-trained network with a hardcoded, pre-provided grafting mask renders all considerations regarding the effect of instability and importance heuristics on graft neuron selection useless. The neuron behavior of the self-trained network will be different from the one in the pre-trained network for which the mask was created according to these considerations.

3.5. Training and Grafting

As in [2], models are trained using a cheap adversarial training method to improve empirical robustness. The fast adversarial training method presented in [58] with gradient alignment regularization is used, adopting the default hyperparameter value

choices as in [59].
For training baseline models, unless stated otherwise, we follow the choices in [2]: 200 epochs of training with a batch size of 128 and a decaying learning rate with the initial value 0.1 which is reduced by a factor of ten in epochs 100 and 150. An SGD-Optimizer with 0.0005 weight decay and 0.9 momentum is used.
For optimizing the grafted networks, the slope and intercept parameters are refined in 100 epochs with an initial learning rate of 0.1 that decays with a cosine annealing schedule. Depending on whether weights are retrained or not, we set the initial learning rate for them to be 0.001 or 0. All other parameters from the baseline model training process are adopted.

3.6. Hardware and Experimental Setup

Early experiments were conducted in a Google Colab Environment using NVIDIA Tesla T4 GPUs with 16 GB memory. All results presented were obtained in experiments run on an NVIDIA A40 GPU with 48 GB memory.

4. Results and Discussion

4.1. Heuristics for Graft Neuron Selection

This section is mainly concerned with analyzing the differences between grafting masks created depending on the choice of γ values, ranking methods and neuron significance heuristics. Comparing the indices of grafted neurons allows first interpretations of the importance and effect of varying the three mentioned parameters before comparing the performance of models grafted with the different masks.

Figure 4.1 displays the effect of using different ranking methods to convert importance and instability indicator values to their respective normalized score values. The **Norank** row gives an idea about the original distribution of the indicators, as the scores are just min-max normalized as defined in (3.2). The following observations can be made regarding the effect of different ranking algorithms:
(i) If there are few or no ties between indicator values, the choice of ranking method is irrelevant. This is clear from the fact that these methods only differ in their tie treatment, but still important to note. We can see that in the case of Magnitude (SAP) and Variance (VAP) heuristics, the score distributions do not change with changing the ranking method. In the case of the Gradient (GAP) heuristic, there seem to be some equal zero values leading to a tie, but apart from that values are different. Note that normalizing the rank leads to an information loss about the relative distance of scores and the original distribution.
(ii) If there are many ties between indicator values, the ranking method significantly influences the score distribution: Instability indicator values are integers ranging between zero and less than 1000 while there are often several thousand neurons to graft, leading to many ties. Min ranking of values leads to a fairly uniform distribution, as the rank and therefore score incorporates the number of tied neurons for a certain value. Dense ranking leads to nearly the same distribution as no ranking because nearly all integer values in the given range are taken on and therefore act like ranks. Ordinal ranking forces some order upon tied neurons leading to a uniform distribution - this leads to a distortion of the data, as it creates a score difference between neurons that do not differ in their indicator value.

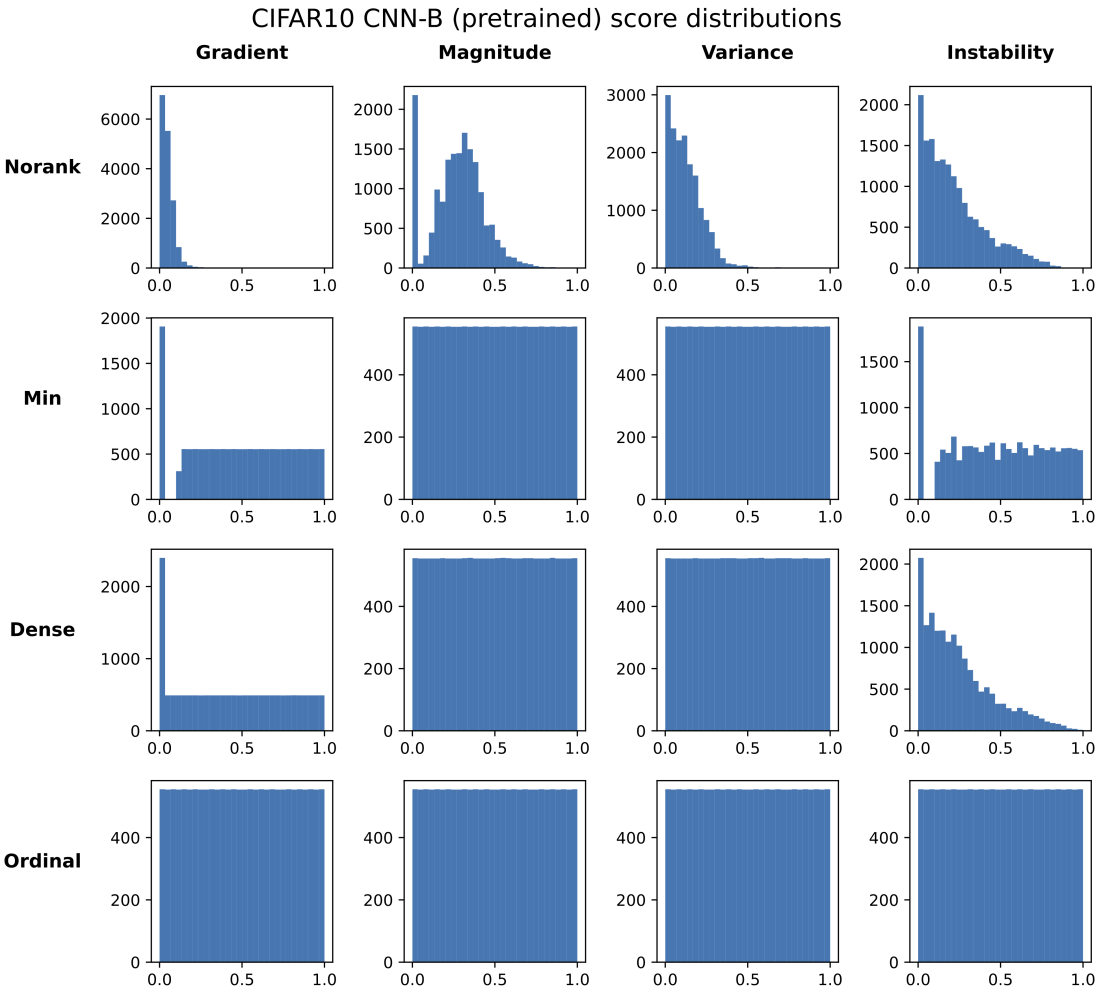

Figure 4.1.: Histograms depicting score distributions depending on heuristic and ranking method

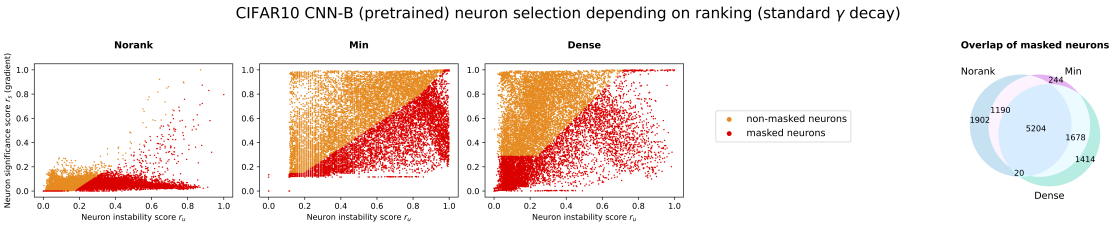

Figure 4.2.: Left: Visualization of neuron score distribution and neuron selection depending on ranking, Right: Overlap of selected neurons

One can already suspect that the result of linearly combining one of the three importance scores with the instability score heavily depends on the ranking algorithm used to create the scores from the indicator values. This is confirmed by the data displayed in Figure 4.2. Visually, the difference in the distribution of neurons plotted on the r_u - r_s plane can be seen. The Venn diagram on the right further confirms that the actual neurons selected differ significantly depending on the ranking algorithm chosen.

To quantify the difference between graft index sets (sets containing the indices of grafted neurons), we use the average pairwise Jaccard distance $\bar{d}_J$. For a set M containing graft index sets I_i, $\bar{d}_J$ is defined in (4.1). Per definition, it holds that $0 \leq \bar{d}_J \leq 1$. A larger value for $\bar{d}_J$ implies a larger dissimilarity between the index sets I_i in M and vice versa. A larger dissimilarity implies a greater effect on graft neuron selection of the respective parameter.

$$\bar{d}_J = \frac{1}{\binom{|M|}{2}} \sum_{\substack{I_i, I_j \in M, \\ i<j}} \frac{|I_i \cup I_j| - |I_i \cap I_j|}{|I_i \cup I_j|} \tag{4.1}$$

Figures 4.3, 4.4 and 4.5 visually and numerically compare the dissimilarity under varied parameter choices for importance heuristics, ranking algorithms and γ values. Ordinal ranking was left out due to its nature of distorting the data by artificially ordering values as outlined above.

As can be seen in Figure 4.4 on the left, when using the GAP heuristic (as stated as the default in [2]), varying the ranking algorithm has a big influence on the actual neuron selection. Numerically, the dissimilarity $\bar{d}_J$ between the selected index sets caused by using different ranking algorithms is greater than any dissimilarity achieved by varying the importance heuristic (under any fixed ranking) as shown in Figure 4.3 ($\bar{d}_J = 0.404$ vs $\bar{d}_J \in \{0.253, 0.278, 0.303\}$). This further illustrates the difficulty in reproducing the results from [2], as there is no detailed description of the ranking algorithm despite its clear impact on neuron selection.

Efforts to reverse-engineer the precise ranking method used in [2] were not fruitful.

As there is only one pair of grafting mask and pre-trained model (CIFAR10 **CNN-B**) provided, this is the only indication of how well our implementations resemble the processes described in [2]. After calculating instability indicator values and importance indicator values for all three presented heuristics on the provided model, masks for all combinations of ranking methods and importance heuristics were created using the default decaying γ values. The similarity of the created masks to the provided mask was evaluated by calculating the Jaccard Similarity s_J value (defined in (4.2)) between the sets containing the indices of selected neurons. Per definition, it holds that $0 \leq s_J \leq 1$ and a larger value of s_J implies a higher degree of similarity.

$$s_J(A, B) = \frac{|A \cap B|}{|A \cup B|} \tag{4.2}$$

For random neuron selection and grafting ratio 0.5, one would expect a Jaccard Similarity value s_J of a third and 50% of the selected neurons matching up with the provided mask. The highest Jaccard Similarity value s_J reached under all combinations was 0.6097, meaning that 75.7% of the 8317 selected neurons to graft were equivalent.
This was achieved for four parameter settings: In two cases the instability indicator value ranking was *norank* and the corresponding significance indicator rankings were *min* and *dense*. In two cases the instability indicator value ranking was *dense* and the corresponding significance indicator rankings were *min* and *dense*. It is interesting to note that in all four cases, the importance heuristic was SAP, although the default significance heuristic used for mask creation according to [2] is GAP.
Furthermore, in three of the four cases achieving the highest similarity, a different ranking method has to be used for ranking instability indicator values and significance indicator values.
Although the different nature of indicator value distributions (importance heuristics: few ties, majority of values often within a small range with few outliers; instability heuristic: many ties, fairly even distribution of values) would be a reason to motivate the use of differing ranking algorithms, there is no mention of this in [2].
Visually, the distribution of neurons on the r_u - r_s plane shown in [2] looks most similar to a distribution resulting from normalizing scores without ranking, as depicted in Figure 4.2 on the very left. This raises the question of whether a universal approach for choosing the γ values makes sense, as for different models the distribution of neuron scores on the r_u - r_s plane will be very different. As shown in Figure 4.5, the difference in neuron selection caused by varying the gamma value used is heavily dependent on the ranking algorithm used: Under dense ranking, this variation only obtains a $\bar{d}_J$ value of 0.194, while no ranking leads to a $\bar{d}_J$ value of 0.448.
The fact that only around 75% of neurons of the most similar mask we can create

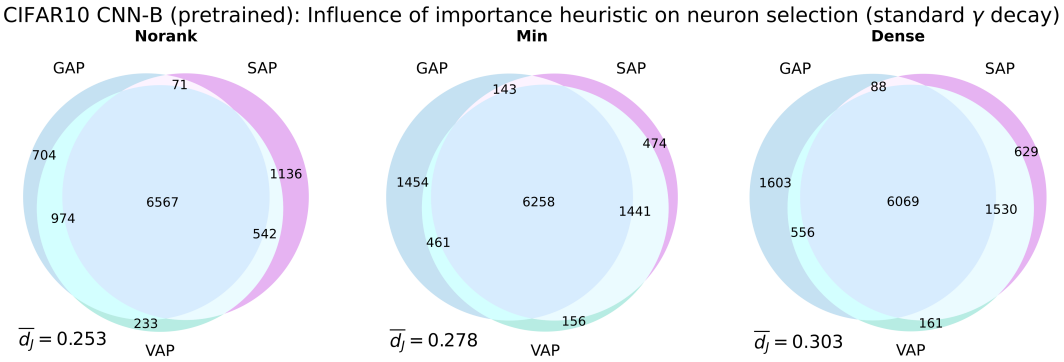

Figure 4.3.: Overlap of selected neurons under varied importance heuristics

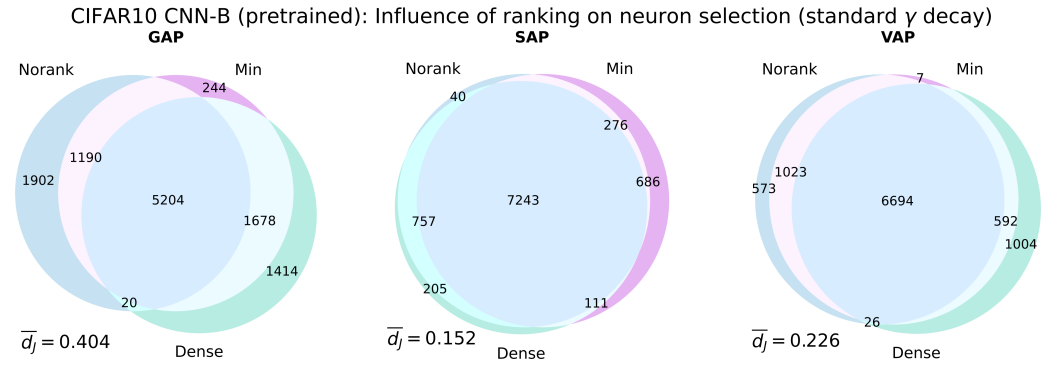

Figure 4.4.: Overlap of selected neurons under varied ranking methods

with our methods match up with the provided mask limits the statements we can make about Linearity Grafting on other models than the pre-trained one. Our own implementations clearly do not exactly resemble the processes used to create the results presented in [2].

A general tendency regarding the newly implemented VAP importance heuristic can be observed in Figure 4.3: The neurons selected under this heuristic would have either been selected under the GAP or the SAP heuristic as well. There are hardly any new neurons chosen under the VAP heuristic. This and the discovered importance of the choice of the ranking algorithm are the reasons why the effect of the VAP importance heuristic on the performance of grafted networks was not the focus of further experiments.
More results regarding the influence of the mentioned parameters on neuron selection in self-trained models are presented in the Appendix.

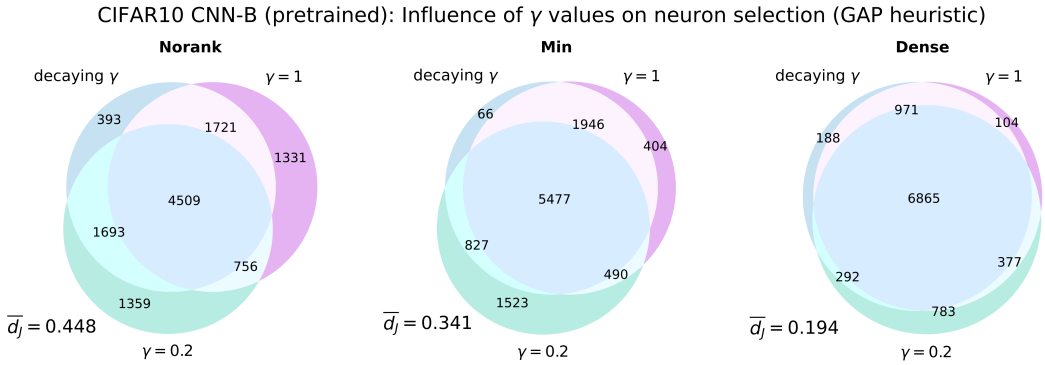

Figure 4.5.: Overlap of selected neurons under varied γ values

4.2. Reproduction of Results

4.2.1. CIFAR10 ConvBig

As shown in 4.1, the results for our self-trained baseline CIFAR10 ConvBig model are very similar to the ones reported in [2]. Comparing our different grafting approaches and their respective results, the following observations can be made:
(i) Not retraining weights in the grafting process leads to consistently worse results when comparing the models grafted with the same masks (created with no ranking and min-ranking): SA, RA and VA values are lower than their counterparts with retrained weights in all cases.
(ii) Min-ranking grafting mask creation leads to significantly better SA and RA results than no ranking, both when retraining weights and when not retraining weights. On the other hand, VA values for the grafted models using min-ranking do not improve significantly compared to the baseline model, rendering grafting with this mask useless. Grafted models using masks created with no ranking reach significantly better VA results similar to the ones reported in [2]. This performance difference depending on the ranking algorithm used illustrates how the trade-off between the neuron selection criteria instability and insignificance influences the performance trade-off between VA and SA/RA. In our experiments, the neurons selected in the mask created from min-ranked indicator values seem to have mainly been chosen with respect to the insignificance criterion, preserving SA and RA but hardly facilitating verification and boosting VA.
(iii) In all cases, the calculated UNR value differs considerably from the reported results, in two of the four cases it is even increased compared to the baseline model. This is either caused by the difference between our grafting mask creation algorithm and the one used in [2] or by a difference in UNR value calculation. The results imply the latter,

	Baseline		Grafted				
	own	reference	wrt		nwrt		reference
			norank	min	norank	min	
SA %	84.85	84.90	74.09	83.34	59.19	73.08	62.23
RA %	68.93	68.10	57.07	65.54	44.84	54.02	47.73
VA %	1.40	1.30	37.8	5.6*	35.2	5.0*	39.12
UNR %	17.27	17.75	20.97	19.10*	16.9	17.21*	4.32

Table 4.1.: Results for self-trained CIFAR10 ConvBig model, (n)wrt stands for (no) weights retrained, UNR and VA results marked with an asterisk come from a test set size of 500, reference values taken from [2]

as in our experiments, VA values similar to the ones reported can be reached, although our calculated UNR is significantly higher.

4.2.2. MNIST ConvBig

Table 4.2 shows the numerical results for our self-trained MNIST ConvBig model. Some observations from Section 4.2.1 made on the same architecture still hold, some are different:
(i) Similarly to the ConvBig model trained on the CIFAR10 dataset, not retraining weights in the grafting process leads to worse performance in terms of SA and RA. In terms of VA, however, it is the other way around, contrary to the behavior observed in the CIFAR10 case.
(ii) As observed above, min-ranking seems to choose the neurons mainly based on insignificance, leading to models preserving their SA and RA performance, but not improving VA rates. But even when not ranking indicator values, our grafted models do not even remotely reach the VA values reported in [2].
(iii) The models grafted using a mask created by not ranking indicator values are the only ones exhibiting significantly increasing VA rates while experiencing worse RA rates than the grafted models using min-ranking.
(iv) As observed above, we do not see a reduction of the UNR value for grafted models, contrary to the results reported in [2]. The two explanation approaches from above hold here as well.
(v) Our baseline model exhibits a big difference in empirical robustness: although only evaluated on a test set size of 1000, the difference in RA value between our model and the reported model (44.6% vs 97.14%) questions the comparability of the results, as our grafted networks use quite a dissimilar baseline model.

	Baseline		Grafted				
	own	reference	wrt		nwrt		reference
			norank	min	norank	min	
SA %	97.95	99.29	98.19	98.32	95.14	98.01	98.68
RA %	44.6**	97.14	64.41	97.71	53.79	97.5	92.73
VA %	0.0	0.10	11.01	0.0*	32.38	0.2*	82.30
UNR %	33.41	31.27	42.95	34.86*	39.75	35.6*	5.85

Table 4.2.: Results for self-trained MNIST ConvBig model, (n)wrt stands for (no) weights retrained, UNR and VA results marked with an asterisk come from a test set size of 500, RA results marked with a double asterisk come from a test set size of 1000, reference values taken from [2]

Generally, we are not able to reproduce the results reported in [2] for the MNIST ConvBig model. This might be due to a too different baseline model or due to the difference between the grafting mask creation algorithms used. Especially the impressive VA gain from 0.1% to 82.3% while only dropping in SA by 0.61pp reported in [2] can not be reproduced. Our grafted model exhibiting the largest VA gain (0.0% to 32.38%) also experiences the largest drop in SA (2.81pp).

4.2.3. CIFAR10 CNN-B

Table 4.3 shows our results for the CIFAR10 CNN-B model. For this architecture, a pre-trained model, a grafting mask and a grafted model were provided. Comparing the provided and the self-trained baseline model, we can see that the results from [2] can be reproduced with a slightly higher VA value and a slightly lower SA value. The following observations on the grafted models can be made:
(i) The ranking method of indicator values significantly influences the grafting mask creation and the performance of the grafted models. In this case, min-ranking achieves better SA, RA and VA values. It seems to prioritize insignificant neurons more in the grafting process, preserving SA. Although its RA value is higher, the attackability of correctly classified images by the grafted network using min-ranking is very similar to the one using no ranking: $1 - \frac{RA}{SA} = 1 - \frac{51.46\%}{66.88\%} = 23.06\%$ vs $1 - \frac{56.97\%}{73.22\%} = 22.19\%$. The latter one even verifies a larger part of empirically robust samples ($\frac{VA}{RA} = \frac{47.0\%}{51.46\%} = 91.33\%$ vs $\frac{49.2\%}{56.97\%} = 86.36\%$), but this is hidden by the drop in SA. This again hints at the tradeoff between unstable and insignificant neuron selection influencing the tradeoff between classification performance and verifiability of the grafted network.
(ii) Grafting the pre-trained model with the provided grafting mask without retraining

	Baseline		Grafted			
	self-trained	provided	self-trained		provided mask	
			norank	min	wrt	nwrt
SA %	78.1	*79.95*	66.88	73.22	*74.08*	64.01
RA %	61.19	*61.4* (62.23)*	51.46	56.97	*57.2* (58.76)*	45.2*
VA %	41.5	*36.04 (37.40)*	47.0	49.2	*49.65 (50.40)*	38.74
UNR %	13.28	*15.85 (15.85)*	14.23	15.45	*13.44 (5.36)*	14.17

Table 4.3.: Results for CIFAR10 CNN-B model, (n)wrt stands for (no) weights retrained, results of pre-trained models in italic with reference values in brackets, RA results marked with an asterisk come from a test set size of 1000

weights achieves worse results than the provided grafted model with retrained weights across all evaluation metrics: RA, SA and VA values are all more than 10pp lower.
(iii) Our calculated UNR value for the provided grafted model differs significantly from the value reported in [2]. This, as suspected above, further hints at a difference in the calculation algorithm of this metric between our implementation and the one used in [2]. Similarly to the observations made above, the UNR values of the grafted models are also not significantly lower compared to their respective baseline models.

4.3. Changes in Verification Status and Subset Properties

To understand the relationship between a trained model and its grafted counterpoint better, we take a closer look at the results obtained when verifying the networks. Instead of just comparing the numeric values of the mentioned evaluation metrics, an empirical analysis of the four result sets depicted in Figure 3.1 of both networks after their verification and the relationship between the sets is done as motivated in Figure 1.1.
For the original network, let the sets created in the verification process containing skipped, successfully attacked, successfully verified and verification failure inputs be called SK, AT, VS and VF respectively. It holds that these sets are mutually exclusive and that they cover all inputs of the test set - in the case of the conducted experiments the cardinality of their union is 1000. Similarly, let the respective sets for the grafted network be called SK_g, AT_g, VS_g and VF_g.
As LG usually augments the Verified Robust Accuracy (VA), and diminishes the Standard Accuracy (SA), the empirical Robust Accuracy (RA) and the portion of

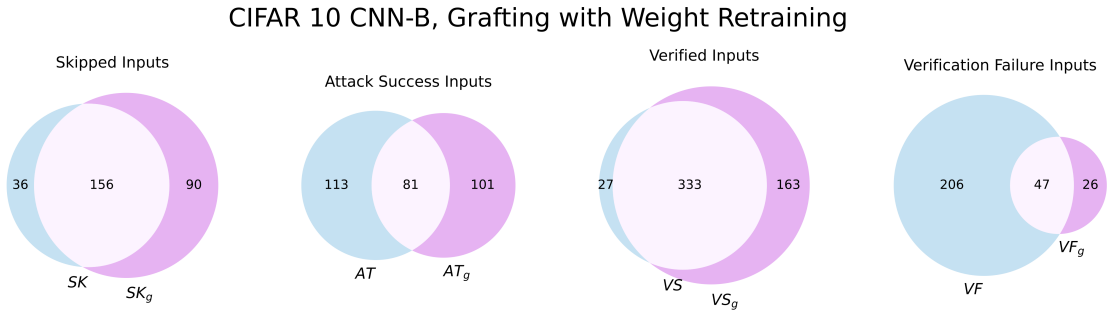

Figure 4.6.: Venn Diagram visualization of overlap of verification result sets. Produced on provided models from [2]

verification failure inputs, the following hypotheses are investigated:

$$\begin{aligned} VS &\overset{?}{\subseteq} VS_g \\ SK &\overset{?}{\subseteq} SK_g \\ AT &\overset{?}{\subseteq} AT_g \\ VF_g &\overset{?}{\subseteq} VF \end{aligned} \tag{4.3}$$

Empirical results of the overlap between the sets defined above for the provided CIFAR10 **CNN-B** model and its grafted counterpart are shown in Figure 4.6. More results for MNIST and CIFAR10 on self-trained **ConvBig** networks are provided in the Appendix. Due to the above-mentioned ambiguities and difficulties in reproducing results, we mainly discuss the results displayed for the provided pre-trained models. It is immediately visible that none of the hypothesized properties in (4.3) strictly hold. Still, some tendencies can be observed:
Firstly, although $VS \not\subseteq VS_g$, we observe that 92.5% of elements (333 out of 360) in VS are contained in VS_g. Inputs verified on the baseline model tend to also be verified on the grafted model and the gain in VA is mainly due to additional verified inputs. A weaker tendency holds for misclassified inputs: 81.25% (156 out of 192) of elements in SK are contained in SK_g. This confirms the expectation that a grafted network performs worse than its baseline counterpart in terms of SA. The majority of inputs misclassified by the baseline model will also be misclassified by the grafted model, as it does not improve classification performance.
Secondly, the hypothesized properties $AT \subseteq AT_g$ and $VF_g \subseteq VF$ or similar tendencies can not be supported by the data. Only 41.75% (81 out of 194) elements in AT are contained in AT_g. Successfully attacked inputs on the baseline model seem to not

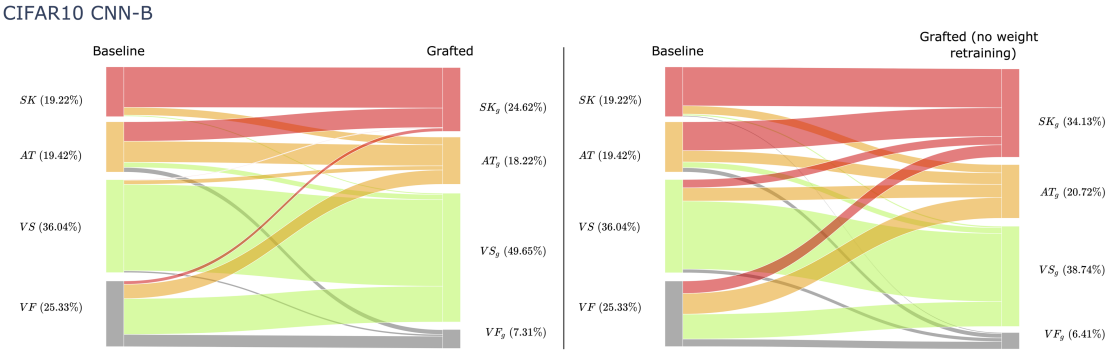

Figure 4.7.: Verification result transitions of test inputs between baseline model and grafted counterpart. Left/Right: Grafting with/without weight retraining

necessarily behave the same way on the grafted model. Of the elements in VF_g, 64.38% (47 out of 73) are contained in VF_g. More than a third of inputs for which the verification failed in the grafted model did not have this status in the baseline model. As LG is primarily motivated by improvements of verification behavior, this result and the other above-mentioned discrepancies from the hypothesized subset behavior motivate a closer look at the transitions of inputs from one verification result set to another between the baseline and the grafted model.

Figure 4.7 (left) visualizes these transitions for the same data as used in Figure 4.6. The following observations can be made:
(i) The majority of additionally skipped examples in the grafted model stem from formerly attacked inputs. Grafting seems to tip over model predictions that were already prone to perturbations. Hardly any formerly non-attackable inputs are misclassified in the grafted model, especially not formerly verified inputs.
(ii) The difference between AT and AT_g mainly comes from two contributions: The flow of formerly attackable, but correctly classified inputs to misclassified inputs in the grafted model (mentioned in (i)) and the flow of verification failure inputs to attackable inputs.
(iii) Hardly any inputs with verification failure in the grafted model were successfully verified in the baseline model. The discrepancy between VF_g and VF noted above mainly comes from formerly attackable inputs. The non-deterministic nature of PGD-100 attacks can explain some formerly attackable inputs leading to a verification failure in the grafted model. The fact that the flow from VF to AT_g is significantly bigger in magnitude than the flow from AT to VF_g implies that grafting makes the network more attackable.
(iv) The gain in VA mainly comes from inputs that formerly led to verification failure

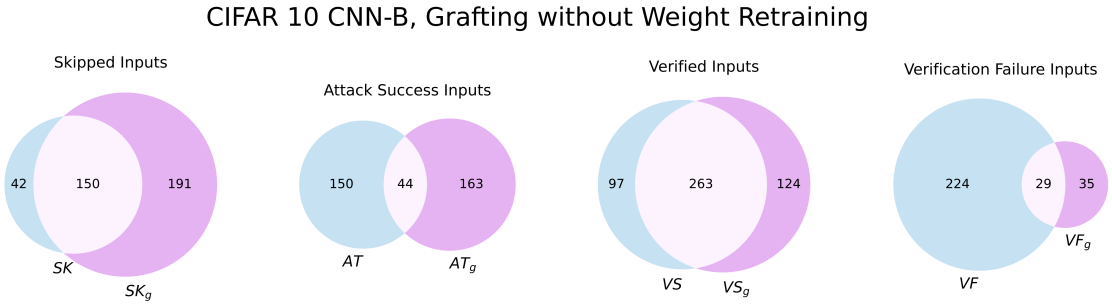

Figure 4.8.: Venn Diagram visualization of overlap of verification result sets. Produced on provided baseline model and self-grafted model using mask provided in [2]

now being verified. Behind the portion of verification failure inputs lies a 'hidden' ratio of attackable and verifiable inputs. LG seems to uncover a big part of this 'hidden' ratio, but it also alters this ratio in the process.

Grafting without retraining weights is motivated by minimizing the difference between the baseline and grafted model. Contrary to the expectation that this higher degree of similarity might lead to a tighter fulfillment of the hypothesized subset properties, Figure 4.8 and 4.7 (right) show that grafting without retraining weights seems to exhibit similar tendencies while amplifying unwanted behavior:
(v) Less elements of VS are contained in VS_g; this discrepancy comes from a significant amount of inputs transitioning from a verified status to being attackable or even misclassified.
(vi) As observed in the numeric results, significantly more inputs are misclassified in the grafted network. While in the case of grafting with weight retraining, this difference was mainly due to a flow from AT to SK_g, we can now observe a significant portion of formerly verified inputs being misclassified in the grafted network as well as a significantly larger flow from VF to SK_g compared to grafting with weight retraining.

Leaving the weights fixed in the grafting process seems to eliminate the possibility for the network to compensate for its loss in expressiveness that comes from the structural simplification by means of linearisation of activation functions. While in the case of retraining weights, the tasks and importance of a neuron might change and adapt to the fact that some crucial neurons have been grafted, this adjustment seems to be limited when only the slope and intercept of the linear approximations can be optimized. This expectably hurts performance in terms of SA. Contrary to expectation, however, although the network is intuitively more similar to its baseline counterpart when grafted without weight retraining, we observe that it tends to fulfill

the hypothesized subset properties less than when grafted with weight retraining.

4.4. Transferability of Adversarial Examples

Adversarial examples have been shown to often be transferable between similar networks [5]. In Section 4.3 it was shown that between the baseline and its grafted model, the individual inputs on which attacks are successful have some overlap. This motivates an investigation into the actual adversarial examples created on the respective inputs and their transferability between the grafted and the original network. As outlined at the end of Chapter 1, scenarios in which the original network has to be used in application would profit from a high transferability of adversarial examples. As it tends to be harder to make statements on the baseline network (usually a higher portion of verification failure inputs), information created in the verification process of the grafted counterpart could be transferred back to the original network.
Figure 4.9 (left) shows the results of using the baseline model to classify the adversarial examples created in the verification process of the grafted model and vice versa. While the former case is more relevant for the reasons outlined above, both directions were studied to better understand the relationship between the two models. The following observations can be made:
(i) The results of transferring adversarial examples from the grafted model to the baseline model exhibit a behavior one would expect from randomly chosen input samples: SA and RA metrics approximately match the ones reported on the test set (here: 83.5%, 65.9% reported: 79.95%, 62.23%). Note the two counteracting reasons why the results should differ from the reported values: (a) the transferred adversarial examples are not created specifically for the attacked network, which makes us expect a lower attack success rate (b) the transferred adversarial examples were created on a similar network (the grafted counterpart) and 100% of them successfully attacked it, this makes us expect a higher attack success rate. Apart from (a), another reason for expecting the adversarial examples to not perfectly transfer is the overlap size of the respective successful attack input sets: As shown in Figure 4.6, only 41.75% (AT) / 44.5% (AT_g) of inputs are contained in the respective corresponding set.
(ii) Transferring successful attacks from the baseline model to the grafted model, however, does not lead to results in terms of SA and RA one would expect on a random selection of inputs (here: 61.3%, 43.8% reported: 74.08%, 58.76%). The grafted network performs worse, misclassifying more unperturbed inputs than in the average case. This is in line with observation (i) from section 4.3. Furthermore, of the correctly classified unperturbed images, the corresponding attack is transferable for a larger portion than one would expect on a random sample set: $1 - \frac{43.8\%}{61.3\%} \approx 28.5\%$ vs $1 - \frac{58.76\%}{74.08\%} \approx 20.7\%$.

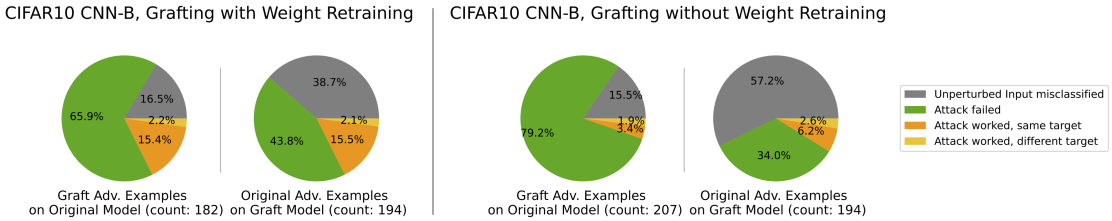

Figure 4.9.: Transferability of adversarial examples created in the verification process. Pre-trained and grafted model/mask from [2]. Left/Right: Grafting with/without retraining weights.

This implies that the similarity of the networks does have a positive effect on the transferability of attack images, but in this case only in the direction not relevant for application (from the baseline model to the grafted model).

The results lend themselves to an intuitive interpretation of baseline and grafted models as a 'strong' and a 'weak' model with corresponding 'strong' and 'weak' attacks. The successful classification rate of the weak model is lower, but of those correctly classified inputs a higher portion is successfully attacked, as we use the 'strong' attacks on it. These two effects counteract each other to produce a similar rate of transferable adversarial examples in both scenarios. The baseline model, however, exhibits the more desirable properties across all comparisons: (a) of the unperturbed images, it classifies a higher portion correctly and (b) of the transferred attack attempts, it is vulnerable to a smaller portion.

For grafting without weight retraining, the results are shown in Figure 4.9 (right). For the more relevant direction of transferring adversarial examples from the grafted network to the baseline network, the robustness lies at 79.2%, far higher than the reported RA value for direct attacks on the baseline model. For only 5.3% of transferred inputs, the baseline model behaves equally, i.e. classifies the unperturbed image correctly but is vulnerable to the transferred attack. From the perspective of an attacker, this is a significantly worse performance than attacking the baseline network directly. Although intuitively a model grafted without weight retraining is less different from its baseline counterpart than with weight retraining, the transferability of adversarial examples is worse. This is in line with the interpretation of the model without weight retraining being even 'weaker' because it can not compensate for its structural simplification (caused by grafting) by adjusting weights. Intuitively, attacking this model requires less work and leads to a perturbation not large enough to fool the baseline model.

The results imply that using Linearity Grafting to create surrogate models for cheap robustness counterexample creation is an unpromising strategy. The focus of further work could be a deeper investigation of why such a large portion of adversarial examples created on the grafted model do not work on the baseline counterpart. Can the notion of a 'weak' and a 'strong' model be confirmed more formally? Is the attack success rate of these adversarial examples so low because they were created on a 'weaker' network and less 'work' was necessary to fool it? Is there a cheap way of manipulating the unsuccessful adversarial examples from the grafted network for them to be more transferable?

4.5. Grafting with and without Weight Retraining

As outlined at the end of Chapter 1, altering the grafting process to only optimize the slopes and intercepts of linear approximations and to leave the model weights untouched is motivated by the goal of being able to minimize and bound the difference between the original and the grafted model. This could be useful in a scenario where the original model has to be used in application, but we can use more easily obtainable verification results from the corresponding grafted model and transfer them back to the original model.
Compared to the grafting method presented in [2], models grafted without retraining weights produce consistently worse results. This holds both in terms of numeric evaluation metrics outlined in section 4.2 as well as desirable behaviors in verification status transfers and adversarial example transferability outlined in sections 4.3 and 4.4. This implies that using models grafted without weight retraining is not a promising strategy. The encouraging attributes of Linearity Grafting as presented in [2] are not conserved sufficiently in this variant of the technique. The results suggest that investing efforts in formally describing and bounding the relationship between a baseline model and its counterpart grafted without weight retraining is not worth it. Even if a way to transfer results within certain bounds back to the baseline model is found, the results obtained on the grafted models without weight retraining tend to not be much more helpful than results on the baseline model. Especially the often significant drops in Standard Accuracy heavily reduce the usefulness of models grafted without retraining.

5. Conclusion

The three main matters of interest to be discussed in this work were mentioned at the end of Chapter 1.

Regarding the reproducibility of results presented in [2], we can draw mixed conclusions. We were mostly able to reproduce the results obtained on baseline models. The incomplete description of the score creation algorithm and the lack of some crucial parts of the implementation in the grafting process however led to unresolvable differences in the grafting mask creation step. Trying out different versions of our implementation did not allow us to conclude that one of those alternatives matches the processes implemented in [2]. No parameter combination in the grafting mask creation step led to grafted models consistently producing similar results to the ones reported in [2]. We especially encountered difficulties reproducing the results reported on the MNIST ConvBig architecture.

The results on the influence of neuron selection heuristics on the performance of networks allow the following conclusions:
There seems to be a general tradeoff between preserving SA and boosting VA, which tends to correspond to the choice in weighting a neuron's insignificance score against its instability score. Due to the nature of the distributions of the raw indicator values, the choice of the ranking algorithm when calculating a neuron's score significantly influences this tradeoff. In fact, we see that varying the ranking algorithm can cause a greater difference in selected neurons than changing the significance heuristic. It also has a similar or greater effect on the difference in selected neurons when compared with the influence of varying gamma values.
We speculate that due to the very different distributions of indicator values depending on the specific trained model, determining fixed γ values that always produce the best results only makes sense when the ranking and normalization procedure leads to fairly evenly distributed scores. This is not the case in the visualization of neurons on the r_u - r_s plane shown in [2] and raises the question whether a simple ablation study on one model as performed in [2] to determine the optimal γ value is sufficient or meaningful.

Regarding the relationship between a baseline model and its grafted counterpart,

we come to the following conclusions:
Generally, the results reflect that a grafted model loses structural complexity and therefore expressiveness. When restricting the grafted model to only optimize for slope and intercept of the linear approximations in the grafted neurons, we further limit its ability to make up for this loss in structural complexity. Establishing a formally and sensibly bounded relationship between the baseline and the grafted model is probably only possible when not retraining weights in the grafting process, but we can see that this variant of the Linearity Grafting technique does not yield promising results.
The effect of simplifying the network's structure can also be seen in the transferability of adversarial examples: The notion of a strong (baseline) and weak (grafted) network is reflected by the fact that the grafted network has both a higher misclassification rate and a higher vulnerability to adversarial examples created on the baseline model than the other way around. This eliminates attacking a baseline network via its grafted counterpart as a sensible strategy.
When looking at the change in verification status for individual inputs under the baseline and grafted model, we see how Linearity Grafting removes uncertainty at the price of worsening some desirable properties of the network: The rate of inputs with a verification failure almost always drops, making a statement about robustness possible for a larger portion of the test set. The misclassification rate and attackability of the grafted network, however, increase. It is important to note that although in many cases Linearity Grafting facilitates a higher VA rate, this does not necessarily mean that the network is robust for a higher portion of inputs. There is some hidden ratio of attackable and non-attackable inputs in the set of verification failure inputs, which might make the baseline network more robust. Empirically, in most cases, this is implied by the results.
This raises an important question about the objectives when reasoning about robustness: Do we prefer a network for which we can guarantee a larger verified robustness, or do we prefer a network with a higher empirical robustness and better classification performance? It is important to note that the larger verified robustness comes from a simpler network structure enabling the verification procedure to finish within some arbitrary time and memory constraint more often, not necessarily from a higher inherent robustness of the network itself.

A limitation of Linearity Grafting is the fact that in the process of creating a grafting mask, information about unstable nodes is needed. As seen above, this information is crucial for creating useful grafting masks that eliminate unstable nodes and facilitate VA gains. The only way to obtain this information, however, is to run verification on the baseline model. It is therefore not possible to create and verify a grafted model without having run a verifier on the baseline model first. Doing this only on a small

subset of the test set to reduce the resources consumed is an option but will yield less representative data on the instability of nodes. Furthermore, verifying the baseline model is typically more resource-intensive per input, as more verification failures occur, meaning that more verification processes run until the end of the timeout length. Not having to run expensive and fruitless verification on the baseline model, however, is one of the central motivations for using Linearity Grafting.

Following the optimistic conclusions regarding the potential of Linearity Grafting drawn in [2], we want to add a different interpretation of the results reported.
We assume that every problem (e.g. classifying MNIST images with 95% SA) has some degree of inherent difficulty and every NN architecture has an inherent capacity to solve problems up to a certain degree of difficulty determined by its structural complexity. These are strong and vague assumptions, furthermore, there is of course no total ordering of problems with respect to their difficulty neither a hierarchy of networks with respect to their structural complexity. Different NN architectures do well on different types of tasks. Still, if these basic notions are accepted, the results from [2] can largely be explained by the phenomenon of using overly complex models on simple tasks and Linearity Grafting only simplifying these unnecessarily complex models. Two Examples illustrate this point:
Firstly, there are models with less than 2000 trainable parameters achieving an accuracy of more than 99% on MNIST [67]. The ConvBig architecture on MNIST optimizes more than two million parameters. The fact that such a low VA value is reached on the ConvBig baseline network might be due to its unnecessary complexity. The grafted network still possesses sufficient expressiveness to solve the task but is now easier to verify because of its simplified structure. Instead of using Linearity Grafting, choosing a baseline model whose inherent maximum problem-solving capacity is closer to the actual problem difficulty might lead to a higher VA rate directly. This investigation of minimal models for certain problems is an area for possible future work and goes beyond the scope of this work. Linearity Grafting as a technique to create or find a minimal model for a problem, however, does not come with the computational or memory advantages found in better-studied distillation techniques like structured pruning. We do acknowledge that increasing model size tends to improve empirical robustness against attacks. There might be a tradeoff between finding the minimal, most easily verifiable network architecture and the most empirically robust architecture for a given problem.
Secondly, being able to reach a VA value of 28.30% on the ConvHuge model on CIFAR10 with 17 million trainable parameters is emphasized as an important result in [2], as verifying networks of this size was unfeasible so far. Upon closer inspection, the usefulness of this result should be seriously questioned. The grafted model experiences

a stark drop in SA from 90.68% to 62.62% and a drop in RA from 73.57% to 49.37%. A model for classifying CIFAR10 that outperforms the grafted ConvHuge model in all aspects can be found in the same results table: The baseline model using the CNN-B architecture outperforms the grafted ConvHuge model in all evaluation metrics: SA, RA and VA. The only reason for using a complex model such as the ConvHuge architecture is the improvement in SA and RA values compared to the CNN-B model. All these advantages, however, are lost in the grafted model. Linearity Grafting in this case seems to just create an unnecessarily overparameterized and still not very expressive architecture, which is still harder to verify than the simpler, not grafted CNN-B architecture.
We speculate that generally, the problem difficulty dictates the minimum complexity of a network, which in turn directly influences the maximally reachable verified robust accuracy. Linearity Grafting can minimize the gap between the minimum necessary complexity and the actual complexity of the network.

5.1. Limits

Due to the limited scope of this work, the amount of experiments and resources spent on optimizing training and verification parameters was naturally restricted. A careful search for enhanced parameters might lead to results painting a different picture of the potential of Linearity Grafting. Furthermore, although we did investigate the influence on the difference in neuron selection depending on the ranking algorithm, γ values and importance heuristics, the actual performance differences between networks grafted with the created masks were not compared excessively. Comparing the effect of varying γ values and importance heuristics on the performance of grafted networks might lead to the conclusion that these parameters cause changes in the neuron selection that are more significant than the ones caused by the ranking algorithm, although they numerically cause fewer changes in the graft index sets. This could be subject to further work. Additionally, incorporating different network architectures and grafting ratios, especially motivated by the above idea of a minimal model for a given problem could provide more insights into the influence of the complexity of models versus the influence of problem difficulty on robustness verifiability.

5.2. Outlook

The inherent complexity of the NN verification problem raises questions about its scalability and applicability, even if successful abstraction and simplification techniques

are discovered. The datasets and model architectures used in academic verification settings are toy examples, orders of magnitude less complex compared to currently deployed industry-standard data and models. The arithmetic distance measurements used to verify robustness within a certain perturbation radius (l_1, l_2, l_∞ norm) hardly align with human perception of the distinguishability of real-world data such as images or sound. Continuing to investigate these toy example versions under artificial conditions of course remains important, as it lays the foundation for tackling more complex and realistic scenarios.
Still, verification will probably not be the only answer to making technology using Neural Networks safer in the area of adversarial attackability. In the worst case, at some point, we might have to come to the conclusion that generally, we are not interested in the systems we can verify and that we can not verify the systems that we are interested in.

Abbreviations

BaB Branch and Bound

LG Linearity Grafting

ML Machine Learning

SA Standard Accuracy

VA Verified Accuracy

RA Robust Accuracy

UNR Unstable Neuron Ratio

FGSM Fast Gradient Sign Method

BIM Basic Iterative Method

PGD Projected Gradient Descent

FAT Fast Adversarial Training

ILCM Iterative Least-Likely Class Method

AI Artificial Intelligence

NLP Natural Language Processing

FNN Feedforward Neural Network

SGD Stochastic Gradient Descent

NN Neural Network

DNN Deep Neural Network

CNN Convolutional Neural Networks

RNN Recurrent Neural Networks

MILP Mixed Integer Linear Programming

SMT Satisfiability Modulo Theories

List of Figures

List of Tables

Bibliography

[1] W. Fedus, B. Zoph, and N. Shazeer, *Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity*, 2022. arXiv: 2101.03961 [cs.LG].

[2] T. Chen, H. Zhang, Z. Zhang, *et al.*, "Linearity grafting: Relaxed neuron pruning helps certifiable robustness," in *International Conference on Machine Learning*, PMLR, 2022, pp. 3760–3772.

[3] "Dnn illustration." (), [Online]. Available: https://tex.stackexchange.com/a/459879 (visited on 09/09/2023).

[4] C. Szegedy, W. Zaremba, I. Sutskever, *et al.*, *Intriguing properties of neural networks*, 2014. arXiv: 1312.6199 [cs.CV].

[5] I. J. Goodfellow, J. Shlens, and C. Szegedy, *Explaining and harnessing adversarial examples*, 2015. arXiv: 1412.6572 [stat.ML].

[6] C. Szegedy, W. Liu, Y. Jia, *et al.*, *Going deeper with convolutions*, 2014. arXiv: 1409.4842 [cs.CV].

[7] S. Qiu, Q. Liu, S. Zhou, and C. Wu, "Review of artificial intelligence adversarial attack and defense technologies," *Applied Sciences*, vol. 9, no. 5, 2019, ISSN: 2076-3417. DOI: 10.3390/app9050909.

[8] N. Carlini and D. Wagner, "Towards evaluating the robustness of neural networks," in *2017 ieee symposium on security and privacy (sp)*, Ieee, 2017, pp. 39–57.

[9] Y. Dong, F. Liao, T. Pang, *et al.*, "Boosting adversarial attacks with momentum," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 9185–9193.

[10] C. Xiao, R. Deng, B. Li, F. Yu, M. Liu, and D. Song, "Characterizing adversarial examples based on spatial consistency information for semantic segmentation," in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018, pp. 217–234.

[11] X. Wei, S. Liang, N. Chen, and X. Cao, "Transferable adversarial attacks for image and video object detection," *arXiv preprint arXiv:1811.12641*, 2018.

[12] Y. Belinkov and Y. Bisk, "Synthetic and natural noise both break neural machine translation," *arXiv preprint arXiv:1711.02173*, 2017.

[13] B. Liang, H. Li, M. Su, P. Bian, X. Li, and W. Shi, "Deep text classification can be fooled," *arXiv preprint arXiv:1704.08006*, 2017.

[14] K. Grosse, N. Papernot, P. Manoharan, M. Backes, and P. McDaniel, "Adversarial perturbations against deep neural networks for malware classification," *arXiv preprint arXiv:1606.04435*, 2016.

[15] A. Gnanasambandam, A. M. Sherman, and S. H. Chan, "Optical adversarial attack," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2021, pp. 92–101.

[16] K. Eykholt, I. Evtimov, E. Fernandes, *et al.*, "Robust physical-world attacks on deep learning visual classification," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 1625–1634.

[17] M. Sharif, S. Bhagavatula, L. Bauer, and M. K. Reiter, "Accessorize to a crime: Real and stealthy attacks on state-of-the-art face recognition," in *Proceedings of the 2016 acm sigsac conference on computer and communications security*, 2016, pp. 1528–1540.

[18] Z. Zhou, D. Tang, X. Wang, W. Han, X. Liu, and K. Zhang, "Invisible mask: Practical attacks on face recognition with infrared," *arXiv preprint arXiv:1803.04683*, 2018.

[19] E. R. Balda, A. Behboodi, and R. Mathar, "Adversarial examples in deep neural networks: An overview," *Deep learning: algorithms and applications*, pp. 31–65, 2020.

[20] N. Akhtar and A. Mian, "Threat of adversarial attacks on deep learning in computer vision: A survey," *Ieee Access*, vol. 6, pp. 14 410–14 430, 2018.

[21] X. Yuan, P. He, Q. Zhu, and X. Li, "Adversarial examples: Attacks and defenses for deep learning," *IEEE transactions on neural networks and learning systems*, vol. 30, no. 9, pp. 2805–2824, 2019.

[22] N. Papernot, P. McDaniel, X. Wu, S. Jha, and A. Swami, "Distillation as a defense to adversarial perturbations against deep neural networks," in *2016 IEEE symposium on security and privacy (SP)*, IEEE, 2016, pp. 582–597.

[23] N. Papernot and P. McDaniel, "Extending defensive distillation," *arXiv preprint arXiv:1705.05264*, 2017.

[24] W. Xu, D. Evans, and Y. Qi, "Feature squeezing: Detecting adversarial examples in deep neural networks," *arXiv preprint arXiv:1704.01155*, 2017.

[25] J. Gao, B. Wang, Z. Lin, W. Xu, and Y. Qi, "Deepcloak: Masking deep neural network models for robustness against adversarial samples," *arXiv preprint arXiv:1702.06763*, 2017.

[26] P. Samangouei, M. Kabkab, and R. Chellappa, "Defense-gan: Protecting classifiers against adversarial attacks using generative models," *arXiv preprint arXiv:1805.06605*, 2018.

[27] D. Meng and H. Chen, "Magnet: A two-pronged defense against adversarial examples," in *Proceedings of the 2017 ACM SIGSAC conference on computer and communications security*, 2017, pp. 135–147.

[28] F. Liao, M. Liang, Y. Dong, T. Pang, X. Hu, and J. Zhu, "Defense against adversarial attacks using high-level representation guided denoiser," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 1778–1787.

[29] G. K. Dziugaite, Z. Ghahramani, and D. M. Roy, "A study of the effect of jpg compression on adversarial images," *arXiv preprint arXiv:1608.00853*, 2016.

[30] C. Xie, J. Wang, Z. Zhang, Y. Zhou, L. Xie, and A. Yuille, "Adversarial examples for semantic segmentation and object detection," in *Proceedings of the IEEE international conference on computer vision*, 2017, pp. 1369–1378.

[31] L. Schmidt, S. Santurkar, D. Tsipras, K. Talwar, and A. Madry, "Adversarially robust generalization requires more data," *Advances in neural information processing systems*, vol. 31, 2018.

[32] D. Tsipras, S. Santurkar, L. Engstrom, A. Turner, and A. Madry, "Robustness may be at odds with accuracy," *arXiv preprint arXiv:1805.12152*, 2018.

[33] N. Carlini and D. Wagner, *Adversarial examples are not easily detected: Bypassing ten detection methods*, 2017. arXiv: 1705.07263 [cs.LG].

[34] N. Carlini and D. Wagner, *Magnet and "efficient defenses against adversarial attacks" are not robust to adversarial examples*, 2017. arXiv: 1711.08478 [cs.LG].

[35] A. Athalye, N. Carlini, and D. Wagner, *Obfuscated gradients give a false sense of security: Circumventing defenses to adversarial examples*, 2018. arXiv: 1802.00420 [cs.LG].

[36] N. Carlini and D. Wagner, "Defensive distillation is not robust to adversarial examples," *arXiv preprint arXiv:1607.04311*, 2016.

[37] N. Carlini, A. Athalye, N. Papernot, *et al.*, *On evaluating adversarial robustness*, 2019. arXiv: 1902.06705 [cs.LG].

[38] C.-H. Cheng, G. Nührenberg, and H. Ruess, "Maximum resilience of artificial neural networks," in *Automated Technology for Verification and Analysis: 15th International Symposium, ATVA 2017, Pune, India, October 3–6, 2017, Proceedings 15*, Springer, 2017, pp. 251–268.

[39] V. Tjeng, K. Xiao, and R. Tedrake, "Evaluating robustness of neural networks with mixed integer programming," *arXiv preprint arXiv:1711.07356*, 2017.

[40] X. Huang, M. Kwiatkowska, S. Wang, and M. Wu, "Safety verification of deep neural networks," in *Computer Aided Verification: 29th International Conference, CAV 2017, Heidelberg, Germany, July 24-28, 2017, Proceedings, Part I 30*, Springer, 2017, pp. 3–29.

[41] R. Ehlers, "Formal verification of piece-wise linear feed-forward neural networks," in *Automated Technology for Verification and Analysis: 15th International Symposium, ATVA 2017, Pune, India, October 3–6, 2017, Proceedings 15*, Springer, 2017, pp. 269–286.

[42] G. Katz, C. Barrett, D. L. Dill, K. Julian, and M. J. Kochenderfer, "Reluplex: An efficient smt solver for verifying deep neural networks," in *Computer Aided Verification: 29th International Conference, CAV 2017, Heidelberg, Germany, July 24-28, 2017, Proceedings, Part I 30*, Springer, 2017, pp. 97–117.

[43] A. Sinha, H. Namkoong, R. Volpi, and J. Duchi, "Certifying some distributional robustness with principled adversarial training," *arXiv preprint arXiv:1710.10571*, 2017.

[44] H. Zhang, T.-W. Weng, P.-Y. Chen, C.-J. Hsieh, and L. Daniel, "Efficient neural network robustness certification with general activation functions," *Advances in neural information processing systems*, vol. 31, 2018.

[45] G. Singh, T. Gehr, M. Püschel, and M. Vechev, "An abstract domain for certifying neural networks," *Proceedings of the ACM on Programming Languages*, vol. 3, no. POPL, pp. 1–30, 2019.

[46] K. Xu, H. Zhang, S. Wang, *et al.*, "Fast and complete: Enabling complete neural network verification with rapid and massively parallel incomplete verifiers," *arXiv preprint arXiv:2011.13824*, 2020.

[47] S. Wang, H. Zhang, K. Xu, *et al.*, "Beta-crown: Efficient bound propagation with per-neuron split constraints for neural network robustness verification," *Advances in Neural Information Processing Systems*, vol. 34, pp. 29 909–29 921, 2021.

[48] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding," *arXiv preprint arXiv:1510.00149*, 2015.

[49] Y. LeCun, J. Denker, and S. Solla, "Optimal brain damage," *Advances in neural information processing systems*, vol. 2, 1989.

[50] S. Wang, X. Wang, S. Ye, P. Zhao, and X. Lin, "Defending dnn adversarial attacks with pruning and logits augmentation," in *2018 IEEE Global Conference on Signal and Information Processing (GlobalSIP)*, IEEE, 2018, pp. 1144–1148.

[51] G. S. Dhillon, K. Azizzadenesheli, Z. C. Lipton, *et al.*, "Stochastic activation pruning for robust adversarial defense," *arXiv preprint arXiv:1803.01442*, 2018.

[52] V. Sehwag, S. Wang, P. Mittal, and S. Jana, "Hydra: Pruning adversarially robust neural networks," *Advances in Neural Information Processing Systems*, vol. 33, pp. 19 655–19 666, 2020.

[53] S. Gui, H. Wang, H. Yang, C. Yu, Z. Wang, and J. Liu, "Model compression with adversarial robustness: A unified optimization framework," *Advances in Neural Information Processing Systems*, vol. 32, 2019.

[54] S. Ye, K. Xu, S. Liu, *et al.*, "Adversarial robustness vs. model compression, or both?" In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2019, pp. 111–120.

[55] V. Sehwag, S. Wang, P. Mittal, and S. Jana, "Towards compact and robust deep neural networks," *arXiv preprint arXiv:1906.06110*, 2019.

[56] A. Jordao and H. Pedrini, "On the effect of pruning on adversarial robustness," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2021, pp. 1–11.

[57] Y. Fu, Q. Yu, Y. Zhang, *et al.*, "Drawing robust scratch tickets: Subnetworks with inborn robustness are found within randomly initialized networks," *Advances in Neural Information Processing Systems*, vol. 34, pp. 13 059–13 072, 2021.

[58] E. Wong, L. Rice, and J. Z. Kolter, "Fast is better than free: Revisiting adversarial training," *arXiv preprint arXiv:2001.03994*, 2020.

[59] M. Andriushchenko and N. Flammarion, *Understanding and improving fast adversarial training*, 2020. arXiv: `2007.02617 [cs.LG]`.

[60] L. Deng, "The mnist database of handwritten digit images for machine learning research," *IEEE Signal Processing Magazine*, vol. 29, no. 6, pp. 141–142, 2012.

[61] A. Krizhevsky, G. Hinton, *et al.*, "Learning multiple layers of features from tiny images," 2009.

[62] S. Dathathri, K. Dvijotham, A. Kurakin, *et al.*, "Enabling certification of verification-agnostic networks via memory-efficient semidefinite programming," *Advances in Neural Information Processing Systems*, vol. 33, pp. 5318–5331, 2020.

[63] M. Mirman, T. Gehr, and M. Vechev, "Differentiable abstract interpretation for provably robust neural networks," in *International Conference on Machine Learning*, PMLR, 2018, pp. 3578–3586.

[64] A. Madry, A. Makelov, L. Schmidt, D. Tsipras, and A. Vladu, "Towards deep learning models resistant to adversarial attacks," *arXiv preprint arXiv:1706.06083*, 2017.

[65] H. Zhang, S. Wang, K. Xu, *et al.*, "General cutting planes for bound-propagation-based neural network verification," *Advances in Neural Information Processing Systems*, vol. 35, pp. 1656–1670, 2022.

[66] X. Ye, P. Dai, J. Luo, *et al.*, "Accelerating cnn training by pruning activation gradients," in *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XXV 16*, Springer, 2020, pp. 322–338.

[67] R. Grimov. "Minimal mnist models." (), [Online]. Available: `https://github.com/ruslangrimov/mnist-minimal-model` (visited on 09/10/2023).

[68] A. Kurakin, I. Goodfellow, and S. Bengio, "Adversarial machine learning at scale," *arXiv preprint arXiv:1611.01236*, 2016.

A. Appendix

A.1. Code and additional Material

All code used to run the presented experiments and to create the visualizations can be found under github.com/brandisf/linearity-grafting-materials. The code was adapted from the resources provided in [2] and uses the α-β-crown verifier ([44], [46], [47], [65]).

A.2. Selection of Adversarial Attack Strategies

Most attacking strategies are gradient-based and aim to find the most sensitive direction of the gradient of the loss function with respect to the input. This is reminiscent of the idea of backpropagation, but instead of minimizing the loss with respect to the model weights, the model weights are considered fixed and the goal is to maximize the loss with respect to changes in the input.
The **Fast Gradient Sign Method (FGSM)** proposed in [5] constructs an adversarial example in the following manner with $\boldsymbol{\theta}$ representing the model parameters and y representing the class label, where $\nabla_x J$ is the gradient of the loss function with respect to the input $\boldsymbol{x}$:

$$\boldsymbol{x'} = \boldsymbol{x} + \epsilon \cdot sign(\nabla_x J(\boldsymbol{\theta}, \boldsymbol{x}, y)) \tag{A.1}$$

One step target class methods pursue an alternative approach by minimizing the loss on an incorrect target label instead of maximizing the loss on the true label:

$$\boldsymbol{x'} = \boldsymbol{x} - \epsilon \cdot sign(\nabla_x J(\boldsymbol{\theta}, \boldsymbol{x}, y_{target})) \tag{A.2}$$

The **Basic Iterative Method (BIM)** presented in [68] applies FGSM repeatedly with a small step size:

$$\boldsymbol{x_0} = \boldsymbol{x}, \boldsymbol{x_{n+1}} = Clip_{x,\epsilon}\{\boldsymbol{x_n} + \alpha \cdot sign(\nabla_x J(\boldsymbol{\theta}, \boldsymbol{x_n}, y))\} \tag{A.3}$$

Other iterative methods include the **Projected Gradient Descent (PGD) Method**, which

operates similarly to BIM but incorporates random restarts and initialization within the ϵ - ball and the **Iterative Least-Likely Class Method (ILCM)**, which iteratively minimizes the loss on the least likely class as the target class [68].

A.3. Additional Result Visualizations

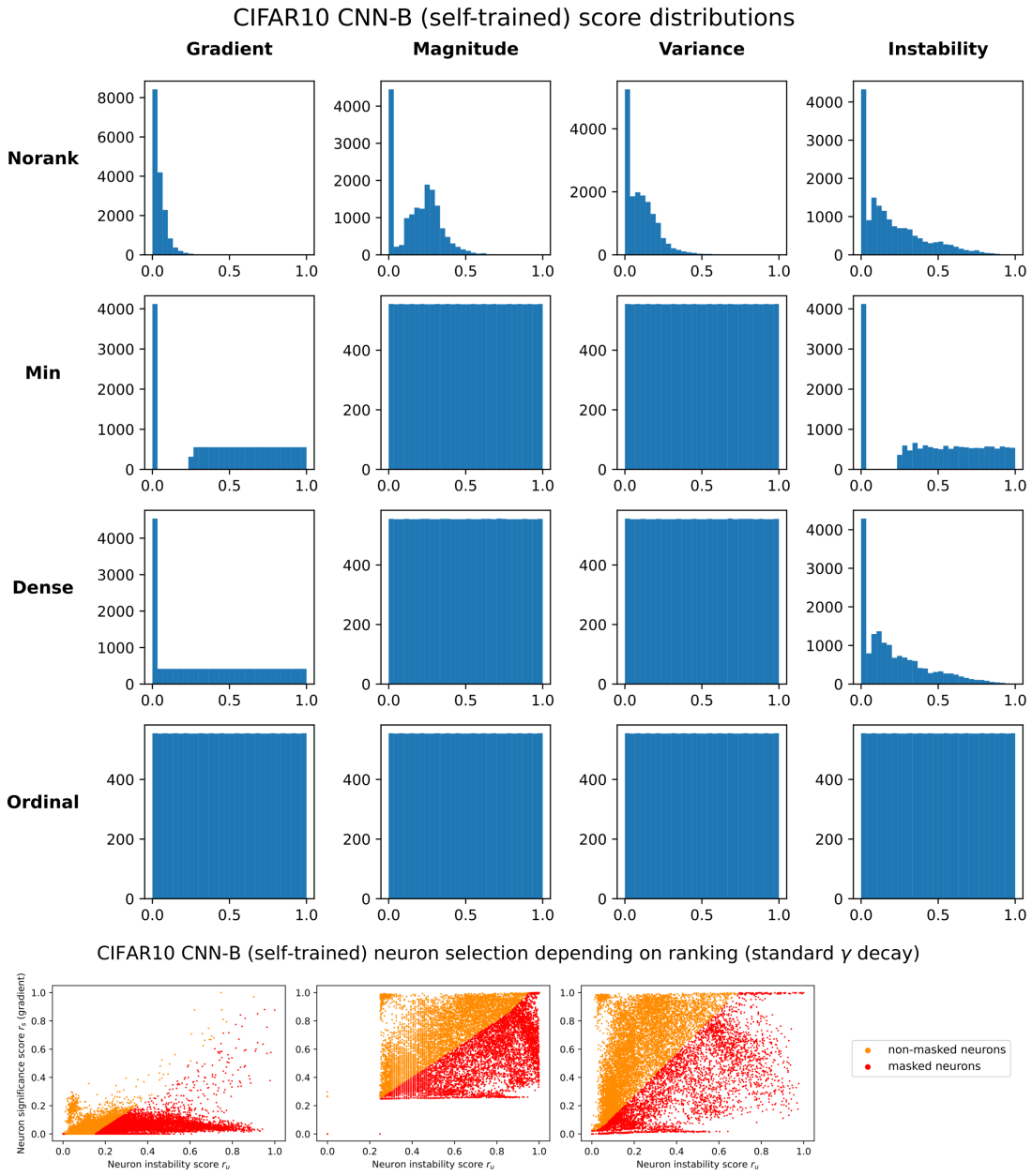

Figure A.1.: Score distribution and neuron selection depending on ranking algorithm for self-trained CIFAR10 **CNN-B** model

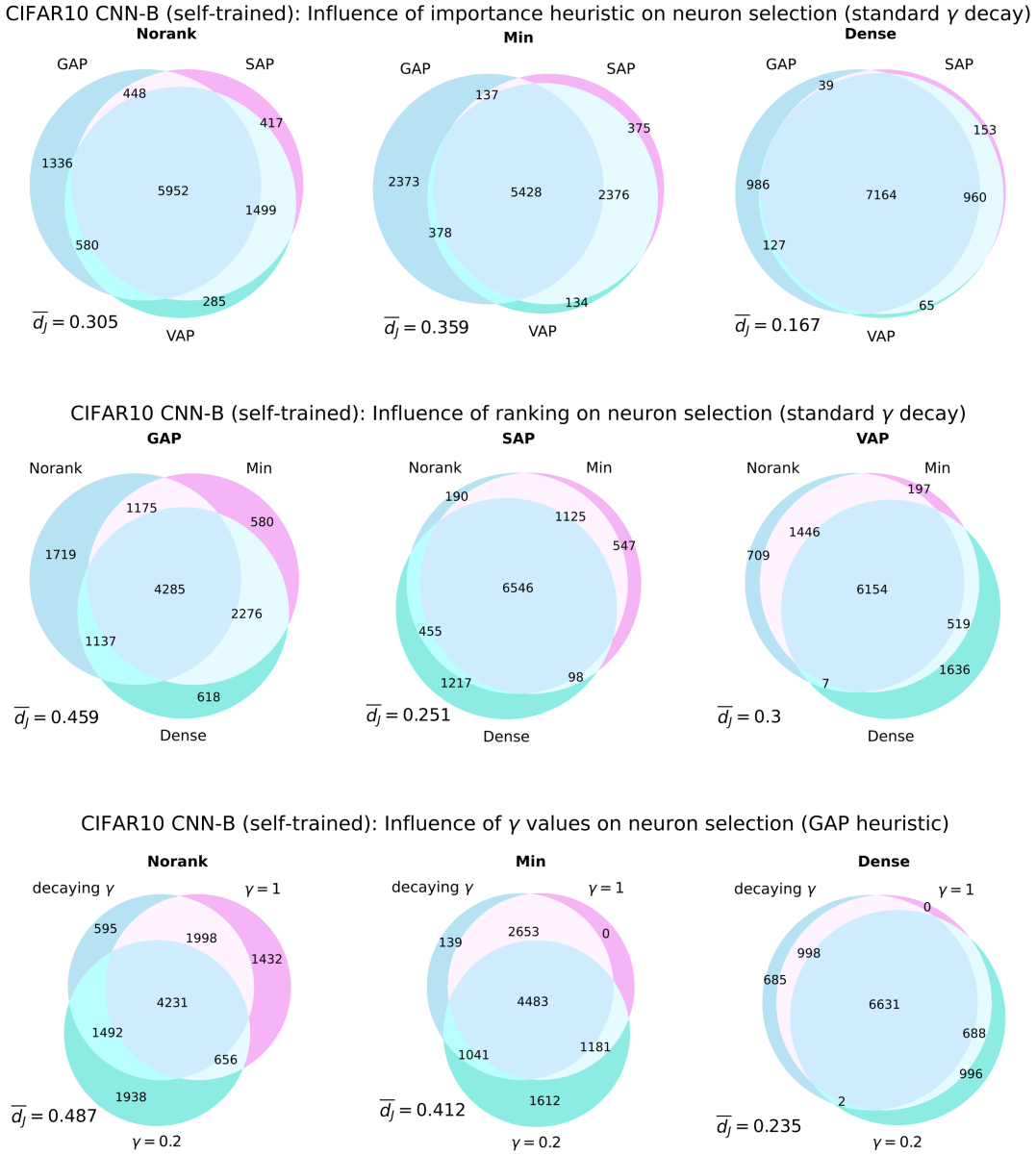

Figure A.2.: Grafting mask creation parameter influence on neuron selection, self-trained CIFAR10 **CNN-B** model

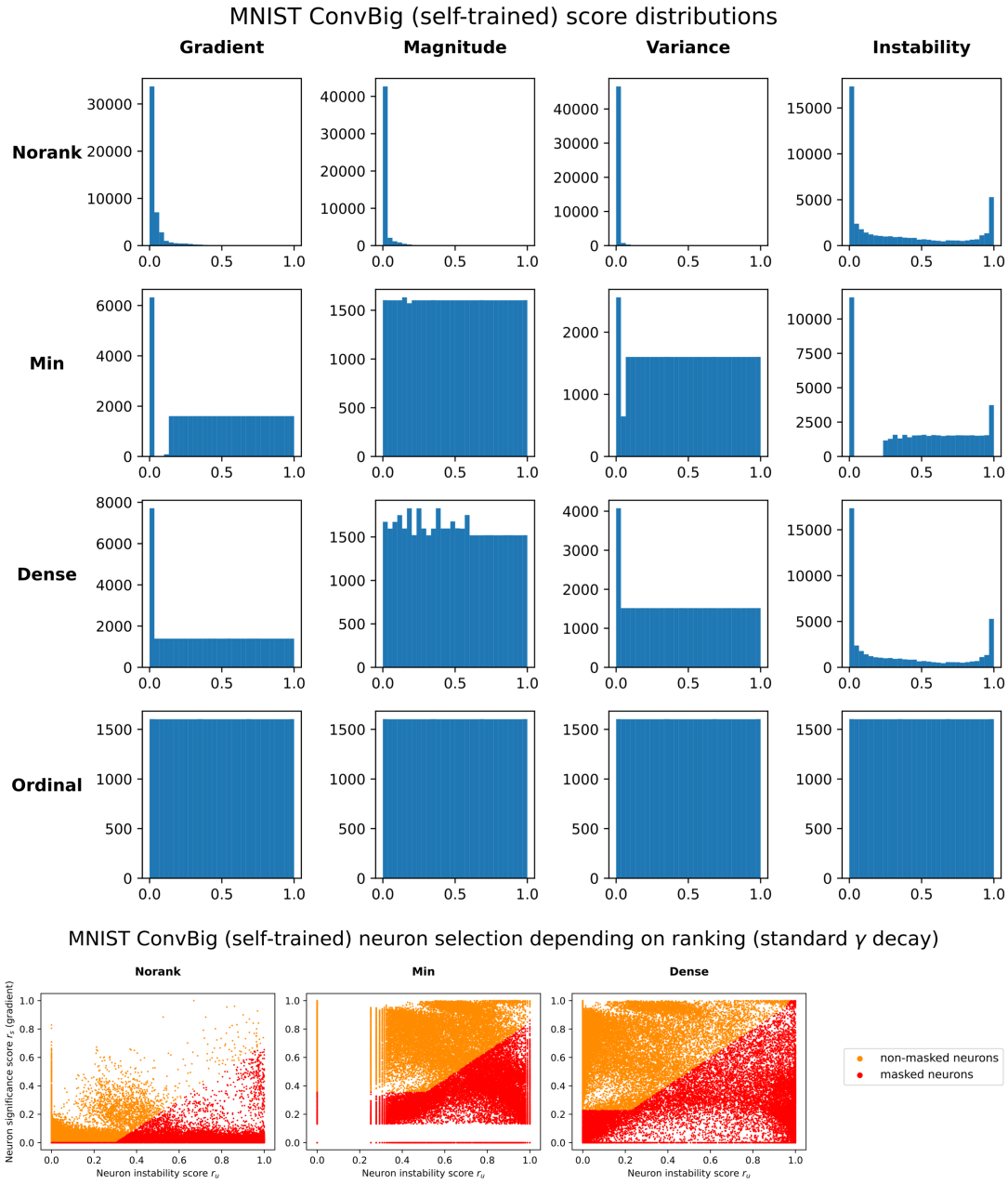

Figure A.3.: Score distribution and neuron selection depending on ranking algorithm for self-trained MNIST **ConvBig** model

Figure A.4.: Grafting mask creation parameter influence on neuron selection, self-trained MNIST **ConvBig** model

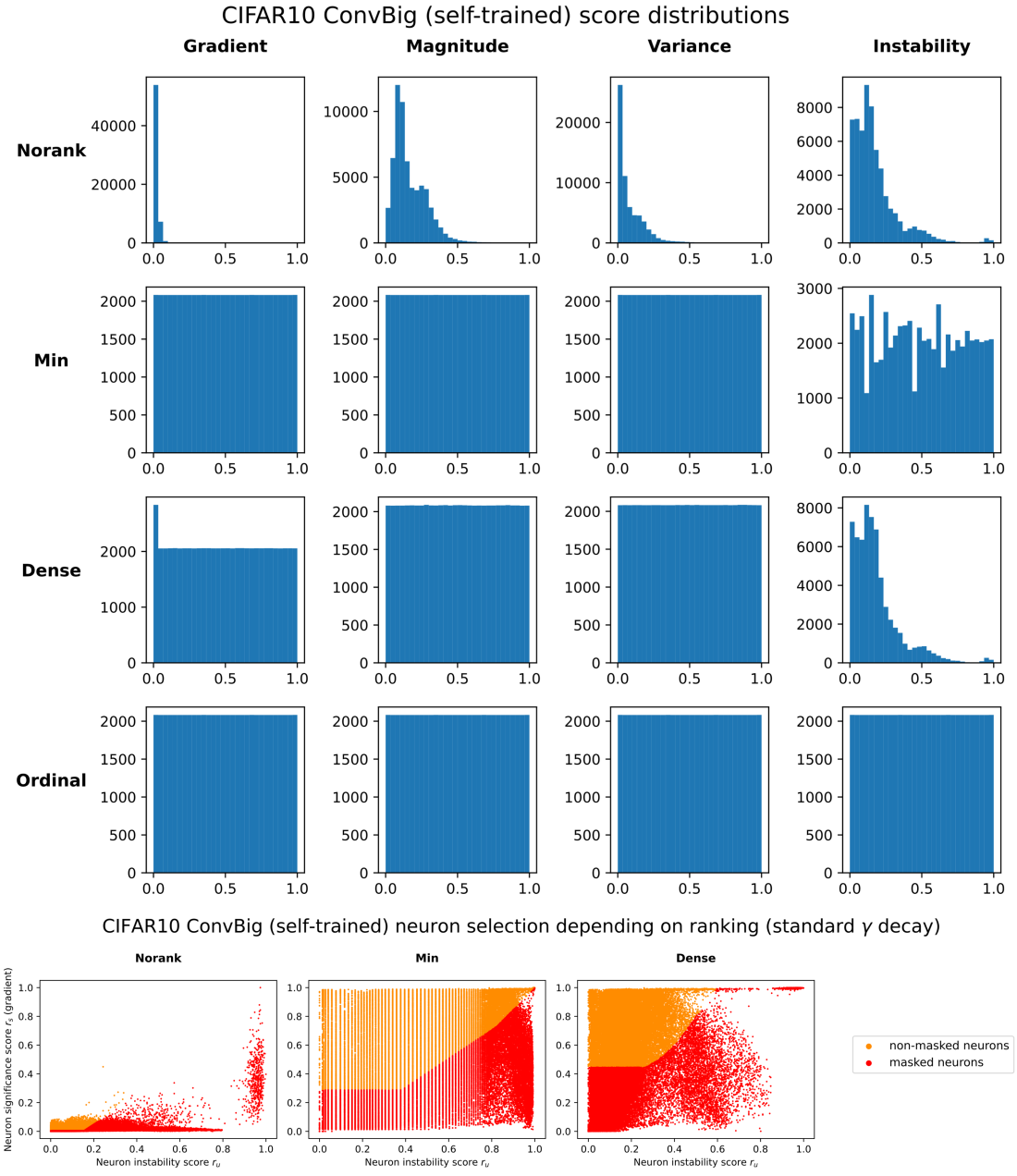

Figure A.5.: Score distribution and neuron selection depending on ranking algorithm for self-trained CIFAR10 **ConvBig** model

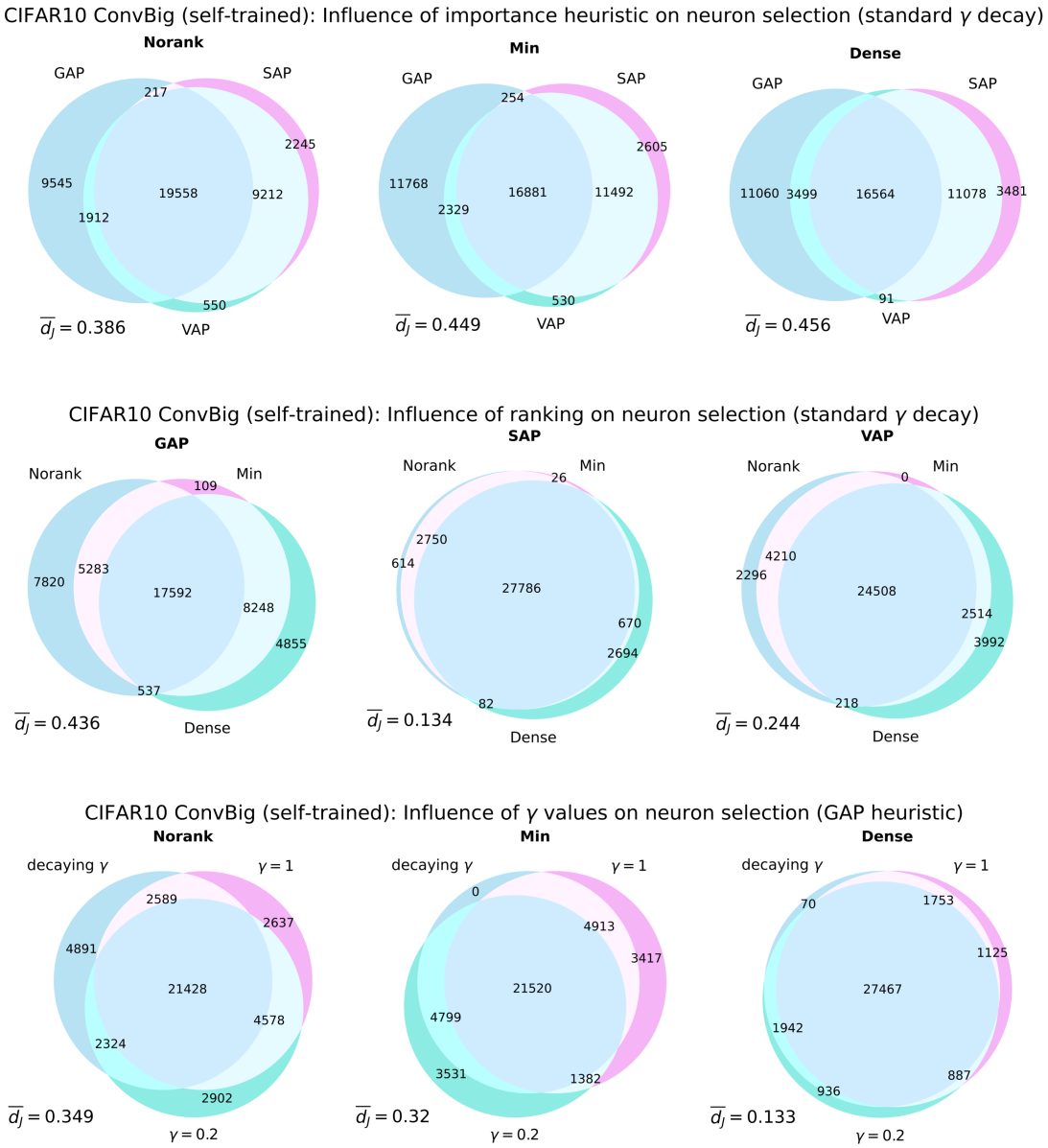

Figure A.6.: Grafting mask creation parameter influence on neuron selection, self-trained CIFAR10 **ConvBig** model

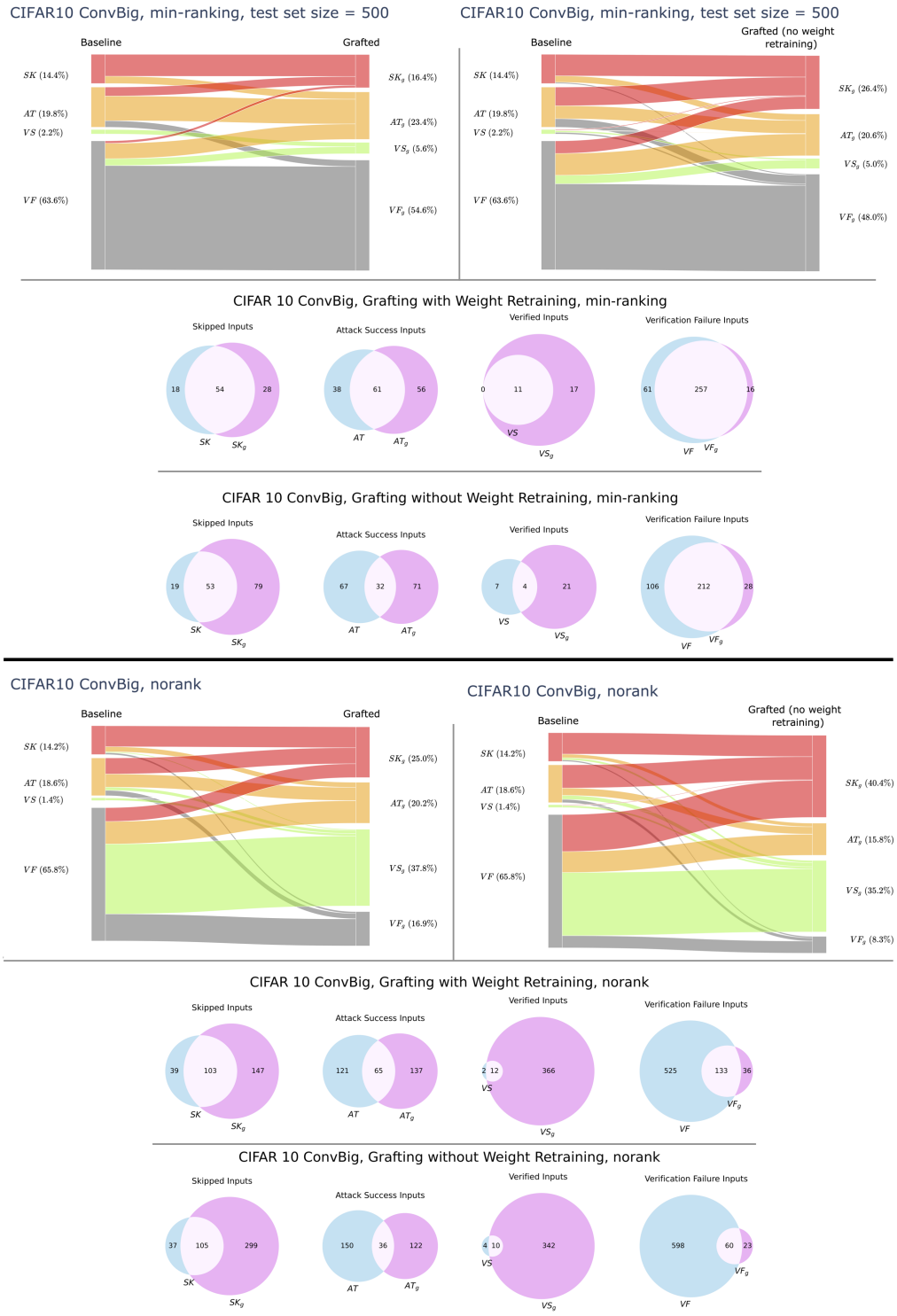

Figure A.7.: Additional verification results for self-trained CIFAR10 **ConvBig** model

Figure A.8.: Additional verification results for self-trained MNIST **ConvBig** model