Universiteit Leiden
The Netherlands

Data Science and Artificial Intelligence

Transfer Learning of Robustness for Image Classification Neural Networks

David Wünsch, s3665534

Supervisors:
Annelot Bosman, M.Sc. & Dr. Jan van Rijn

BACHELOR THESIS

Leiden Institute of Advanced Computer Science (LIACS)
www.liacs.leidenuniv.nl 15/01/2026

Abstract

Neural networks have been applied to various safety-relevant applications. Although they perform very well under normal conditions, it has been shown that they are susceptible to adversarial input perturbations, leading them to misclassify images. Methods to adversarially train the network in order to improve its robustness have been developed, but they come at the cost of very high training times. This thesis investigates whether it is possible to transfer robustness from already existing adversarially trained source models and if it is computationally beneficial. We train models adversarially and conventionally from scratch on MNIST and CIFAR datasets and analyse their robustness. We perform transfer learning to adapt the source models to the target domain using conventional retraining. We then compare accuracy scores, robustness distributions and training times of the different models. We find that robustness could be successfully transferred, and approximately 66% of the robustness could be retained. We also find that transfer learning is roughly 10 to 250 times faster than adversarial training from scratch. We further tried to improve the transferred model's accuracy by experimenting with replacing the last layer with a multi-layer perceptron and adversarially retraining the network instead of conventionally retraining it. We conclude that this method has various applications. For example, if one's running time is restricted and one does not necessarily require maximum robustness and accuracy, one could apply this method. Also, if one wants to adapt an adversarially trained source model to multiple different target domains one could apply this method, because running time savings scale up.

Table of Contents

Chapter 1

Introduction

Neural networks are widely used for tasks such as modelling, pattern recognition, prediction, and classification tasks [1]. The subfield of image classification encompasses various applications such as medical image classifications [2, 3, 4], object recognition [5], satellite imagery analysis [6], and autonomous traffic sign detection for self-driving cars [7]. Safety is a major concern for some of those applications. It is imperative that a classified road sign or a medical image is correctly classified and not vulnerable to adversarial attacks.

Research by Szegedy et al. [8] has shown that neural networks are vulnerable to perturbations. Perturbations are small changes or disturbances to the image that are often imperceptible to the human eye [9]. This can lead the network to misclassify images that are easily recognisable by humans and, under normal conditions, also by the network. This results in potentially severe outcomes like car accidents or false medical decisions [10]. It is therefore crucial to evaluate and to improve the robustness of neural networks against those perturbations. This can be done by adversarial training, a method in which instances are perturbed before they are fed into the training loop [11].

To assess the robustness of neural networks, recent research has investigated how neural network robustness can be quantified and how it can be computed. Bosman et al. [12] have introduced the concept of robustness distributions to assess the robustness of a neural network dependent on the perturbation radius. This circumvents the limitations of the previously commonly used metric robust accuracy, which indicated the percentage of inputs that are classified correctly, regardless of the perturbation within a predefined bound ε.

Since training machine learning models from scratch requires significant data, resources, and running time, transfer learning has emerged as a new research field [13]. It is a method to speed up the training of networks for similar domain datasets by transferring the knowledge acquired from one task to another, exploiting the structure of a neural network to keep the feature extractors unchanged and only retrain the classification heads to adapt it to the new domain [14].

The work by Shafahi et al. [14] showed that robust feature extractors can be recycled from existing networks and that lifelong training methods can be utilised to prevent the network from forgetting robustness during retraining. They especially focused on transferring robustness from larger models trained on larger datasets to smaller datasets. This thesis is further based on the earlier work performed by Baumann [15], who retrained networks layer by layer from the end in order to investigate the influence of each layer on the robustness of the network. It has been shown that robustness can be transferred from MNIST to EMNIST models. The state of the art of research still lacks data on the transfer of robustness from smaller datasets to larger datasets, such as CIFAR-10 to CIFAR-100. Our work will close this gap and extend the research to new datasets and domains.

This thesis aims to investigate the transferability of robustness from adversarially trained neural networks trained on a smaller source domain dataset to a target model using conventional retraining on a similar but extended target domain dataset. We will study the following three main questions.

1. Does robustness transfer from an adversarially trained source neural network to a target domain dataset using conventional retraining?
2. Does transfer learning of robustness produce a model with similar robustness and accuracy as adversarial training of a target domain model from scratch?
3. Is transfer learning of robustness computationally more efficient than adversarial training of a target domain model from scratch?

We will explore those research questions by training the networks conventionally, adversarially, and by transferring robustness from the adversarially trained source model to the conventionally trained target model. We will compare the performance of the networks and investigate their robustness using robustness distributions using incomplete verification. We expect that we can transfer robustness and achieve similar or slightly worse robust accuracy scores than adversarially trained neural networks from scratch. We hypothesise this to be achievable with substantially less training time than adversarial training from scratch. During the experimentation phase, we will conduct additional side experiments, exploring adversarial retraining for transfer learning and replacing the last layer of networks with multi layer perceptrons.

The thesis is structured as follows. First, the background concepts are explained in Chapter 2. In Chapter 3, we discuss a paper and a master's thesis that are directly related to this work. Then we show the methods in Chapter 4 and experimental setup in Chapter 5. In Chapter 6, we analyse the experiments, present the outcomes and discuss the limitations of the work. Some conclusions and future work are discussed in Chapter 7.

Chapter 2

Background

This chapter introduces the fundamental concepts and techniques, laying the foundation for the later work in this thesis.

2.1 Neural Networks

Neural networks are computational models inspired by the structure and function of the brain [16]. They are commonly used for regression or classification tasks [1]. The following explanations about neural networks are based on the standard literature by Goodfellow et al. [17]. Neural networks consist of neurons organised in layers. The input layer is fed with the input data (features), and the output layer represents the prediction or classification of the neural network. The number of neurons in the input layer is equal to the number of input features, and the neurons in the output layer could, for example, represent a probability score (for classification) or a continuous value (for regression). The layers in between the input and the output layer are called hidden layers.

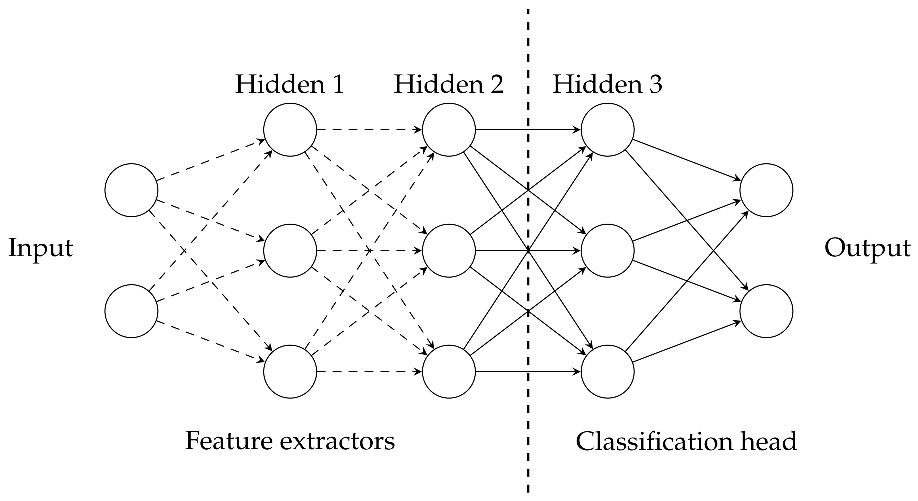


Figure 2.1: Schematic sketch of a neural network. The dashed lines represent possibly convolutional layers. The normal lines represent fully connected layers.

The neural network can be characterised by the width, depth, and size. The width is the maximum number of neurons in any layer. The depth represents the number of layers in the networks, and the size of the neural network is the total number of neurons across all layers. There are many different kinds of neural networks. Commonly researched models are fully-connected and convolutional neural networks. In fully-connected neural networks, all neurons of a layer are connected to all the neurons of the preceding and following layers. Convolutional neural networks are models in that not all neurons are necessarily connected among layers. The convolutional model can be thought of as consisting of a feature extractor part and a classification head. The feature extractor contains the convolutional layers, which act as a filter for the neural network, extracting scale- and location-invariant features. It can detect both simple features like edges and corners, and complex features like textures, shapes, and parts of objects. This representation is then passed to the classification head, which consists of fully connected layers. It maps the features to a discrete set of class labels or a probability distribution, resulting in the final prediction of the network. Convolutional models outperform traditional networks on image classification tasks and are widely used in research [18].

Mathematically, a model can be described as a classifier mapping $f : \mathbb{R}^m \to \{1, \ldots, N\}$ from image pixel values to a discrete label set. The training of the it consists of a forward and a backward pass. The following explanations about the training loop are based on the work by Hertz et al. [19]. In the forward pass, an example input x_i with labels y_i is chosen and applied to the input layer v_i^0:

$$v_i^0 = x_i. \tag{2.1}$$

Now, the applied activation is forward passed through the layer until the last layer is reached. The neuron inputs along this activation path are calculated using

$$h_i^k = \sum_j w_{ij}^k v_j^{k-1}, \tag{2.2}$$

where w_{ij}^k is the connection weight of the connection from neuron j in layer $k-1$ to neuron i in layer k and v_j^{k-1} is the output of neuron j in layer $k-1$. The neuron outputs are then calculated with

$$v_i^k = g(h_i^k), \tag{2.3}$$

using an activation function g. This can be a function like ReLU, sigmoid, or tanh. It introduces non-linearity into the network, which is crucial for enabling the model to learn complex patterns in non-linear data. ReLU has emerged as the state of the art in research. This way, the input is forward-passed until the last layer is reached. At the last layer the loss of the last layer is calculated:

$$\delta_i^K = g'(h_i^K)\left[y_i - v_i^K\right]. \tag{2.4}$$

y_i is the expected, labeled output for input x_i and v_i^K is the output of the neurons i of the last layer K and g' is the derivative of g. Now the backward pass of the training loop begins, where the adaptation and learning of the network happen. The loss is layer-by-layer back-propagated through the network, using the chain rule from calculus with

$$\delta_i^{k-1} = g'(h_i^{k-1}) \sum_j w_{ji}^k \delta_j^k. \tag{2.5}$$

Lastly, the weights of the neurons are updated with the learning rate η, depending on the loss with

$$w_i^k = w_i^k + \eta \delta_i^K v_j^{k-1}. \tag{2.6}$$

2.2 Local Robustness

The following formal concepts are mainly based on the work by Szegedy et al. [8]. In this and the following paragraphs, we let away the indices indicating the neurons and layers that we used in Section 2.1.

One can assess the performance of neural networks by calculating their accuracy. However, this reveals nothing about the security of the network. It is possible that models with very high accuracy scores of close to 100% can still be fooled by possibly maliciously perturbed images [20]. If a perturbed image leads the network to misclassify the data, the image is called *adversarial example* or *counterexample* [8].

Robustness describes the ability of a neural network to maintain the correct classification for perturbed images. A model is ε-robust for a given instance x_0, if its prediction $\arg\max f(x_0)$ stays the same despite a perturbed input with regards to a norm p, which Baumann [15] formally described by the condition

$$\forall x : \|x - x_0\|_p \leq \varepsilon \Rightarrow \arg\max f(x) = \arg\max f(x_0). \tag{2.7}$$

In this equation, D is a subset of a dataset, x is the perturbed input, x_0 is the clean input, and ε is the perturbation radius, i.e., the maximum magnitude of allowable input perturbations [21].

Local robustness implies robustness for a specific set of images, whereas global robustness implies robustness for all possible inputs. Also, local robustness refers specifically to small perturbations, and global robustness could be any faults in images. Global robustness would be a favourable metric, but it is impossible to compute. Therefore, local robustness is the only possible way to assess the robustness of a network [22].

One can quantify the robustness of neural networks through various metrics, such as robust accuracy or local robustness verification. Robust accuracy is the percentage of inputs that are provably classified correctly, regardless of the perturbation within the bound ε [23]. This metric has some limitations and cannot fully capture important aspects of robustness. It requires domain knowledge to pre-specify acceptable perturbation levels, and it does not indicate what level of perturbation is tolerated [12].

2.3 Verification and Robustness Distributions

Local robustness verification methods formally assess whether a neural network's prediction remains correct when a small, predefined perturbation ε is applied to the image. Two categories of methods can be employed for this task: incomplete and complete verification. Both methods are sound, which is a mathematical guarantee that the network is actually robust when the verifier states it. But only complete verification, given enough running time and resources, results in a definitive, either robust or not robust outcome. It provides a formal, mathematical proof that a network is robust for all possible inputs. While complete methods are exact and provide the highest level of assurance, the underlying problem is NP-hard. Hence, complete verification is computationally expensive and does not scale well to large networks or datasets. Incomplete verification sacrifices exactness for the sake of computational efficiency. It still guarantees that a model is robust for a predefined ε, if the verifier states it. However, its verification query can also result in an unknown outcome, for which the instance could either be verified as robust or not robust [24].

While verification methods can guarantee robustness for a predefined ε, they do not determine the extent to which a neural network is robust. To assess the exact magnitude of perturbation required to cause a misclassification, Bosman et al. [12] suggest the concept of critical ε^*. The critical ε^* value is defined as the value such that any perturbed input

$$x \in \{x : \|x - x_0\|_\infty \leq \varepsilon^*\} \tag{2.8}$$

cannot lead to misclassification, and some perturbations larger than ε^* provably lead to misclassification. This concept has also been described in other work as adversarial radius [25], adversarial result [26] and maximum perturbation bound [27]. An empirical lower-bound of the critical ε value is determined by searching a discretised set of ε-values, e.g. $S = \{0.001, 0.003, 0.005, ..., 0.399\}$, for the transition point from robust to non-robust. This transition point is then called the empirical lower bound $\tilde{\varepsilon}^*$. A k-binary search algorithm is employed to search for this transition point. In order to analyse a network for various inputs, Bosman et al. [12] introduce robustness distributions, which are a distribution of critical ε^* values over a set of test images. This robustness measure offers a holistic view of a network's vulnerability. The shape of this distribution reveals valuable information about the kind of robustness of the network and the generalisation to unseen instances. Both complete and incomplete verification methods can be employed to find critical ε^* values [25].

Furthermore, there are methods to estimate upper bounds for critical ε^* values, with running times two to four magnitudes faster than complete verification. Berger et al. [28] suggests using adversarial attacks, such as the later explained Projected Gradient Descent [29], for determining upper bounds for robustness distributions of a model f. A binary search algorithm is used to find the smallest perturbation radius that causes a misclassification, called the minimum adversarial perturbation p^*, producing near-optimal upper bounds for ε^*. We will employ this method in this thesis. Overall, one can summarise the gap between the various methods as follows:

$$\varepsilon^*_{incomplete} \leq \varepsilon^* \leq p^*. \tag{2.9}$$

For comparing robustness distributions of different models, we will determine the median of the p^* values. This will give a nuanced robustness metric for the networks, which makes it possible to compare the models with each other. The metric indicates that 50% of the instances have a higher p^* and 50% have a lower p^*. We choose the median over the mean because it is robust to outliers, which is common for robustness distributions.

2.4 Adversarial Training

Conventional training involves performing training on the training set with the goal of maximising accuracy on the validation set [17]. Adversarial training tries to maximise accuracy and robustness at the cost of significantly longer training time [29]. Goodfellow et al. [11] suggested to improve robustness by training the model with adversarial examples. This can be done by applying an adversarial attack before the first training step, which is successful when it can break Equation 2.7. This example is added in the training loop of the neural network in Equation 2.1 and used for the training loop. The adversarial examples are created by using adversarial attack methods, which are further described in this section.

Adversarial attack methods can be categorised into white and black-box attacks. White-box attacks have full access to the model. They can access the architecture as well as parameters such as weights and gradients. Black-box attacks, on the other hand, have no access to the model architecture or its internal parameters. They solely function based on the output the model produces for a specific input [29].

The Fast Gradient Sign Method (FGSM) is a white-box adversarial attack that exploits the hypothesis that deep neural networks are too linear [11]. Due to this linearity, many imperceptible changes to the input can accumulate into a large, impactful change on the network's output. The attack tries to find a perturbation that maximises the loss-function in a single step, by taking the sign of the gradient ∇ of the loss-function δ, such as the one described in Equation 2.4, with model parameters θ and multiplying it by a predefined ε [11, 29]:

$$x = x_0 + \varepsilon \operatorname{sgn}\left(\nabla_{x_0} \delta(\theta, x_0, y)\right). \tag{2.10}$$

Madry et al. [29] observed that this single-step approach can cause overfitting on the adversarial examples and decrease the accuracy on natural examples. Therefore, they suggest an iterative multi-step process called Projected Gradient Descent (PGD). Instead of applying a single step that maximises the loss, it solves this maximisation problem more nuanced. Training a neural network can be seen as a minimisation problem of the loss function from Equation 2.4. Adversarial training is an inner maximisation problem of this loss function within this minimisation problem. The authors found that the loss landscape corresponding to this problem has a surprisingly tractable structure of local maxima [29]. To find those true local maxima, the gradient methods used in FGSM are iteratively applied in multiple small gradient steps:

$$x^{t+1} = \prod_{x+S} \left(x^t + \alpha \ \mathrm{sgn}(\nabla_{x_0} \delta(\theta, x_0, y)) \right). \tag{2.11}$$

In this equation, S is the set allowed perturbations, $\prod$ is the projection operator, ensuring that the adversarial example stays within the range $x + S$ and α is the size of each small step t. After each step, PGD projects the perturbed input back onto the ε-ball. PGD can find more challenging counterexamples and has emerged as one of the most popular and effective tools for adversarial training [9].

Recently Zou et al. [30] developed a new Dice Adversarial Robustness Distillation (DARD) framework. This framework is built upon the Dice Projected Gradient Descent training method, which relies on a Dice Loss and a dynamic weighting strategy. This approach mitigates overfitting on the specific attack patterns used by standard adversarial attacks, such as PGD, by shifting the attack's focus gradually from perturbing correctly classified samples to misclassified samples in the later iterations. DARD uses this to train a teacher model adversarially. The teacher model then provides soft labels, which are probability distributions over the classification labels, for both natural and adversarial examples. The student model is then adversarially trained on those soft labels. By using soft labels instead of hard labels, which only tell the true class, knowledge about both clean and adversarial instances can be transferred from the teacher model to the student model. With this method, the authors could notably improve both clean and robust accuracy at the same time compared to standard adversarial training.

2.5 Transfer Learning

Transfer learning has the primary goal to improve the learning of a target model by utilising knowledge extracted from a source domain and task [13]. It can be used to speed up training when training data is limited for the target domain or to avoid expensive and time-consuming data gathering [31]. This has broad applications in areas like fine-tuning image classification models to recognise different classes of objects or medical image classification networks, where models trained on extensive amounts of natural images are fine-tuned to medical images. However, transfer learning methods require that the source and target domains are similar. If they are not, performance could even be hurt, which is called negative transfer [32, 33].

Three central questions emerge about transfer learning: "What to transfer", "When to transfer", and "How to transfer". The latter refers to the algorithm that needs to be developed to transfer knowledge. The other two questions are used in the literature to further categorise the different transfer learning settings [13, 32].

The question "When to transfer" enables three categorisations: Inductive transfer learning, transductive transfer learning, and unsupervised transfer learning, as shown in Figure 2.2. In inductive transfer learning, the source and target tasks are different, the domains can be related or not. For this setting, some labelled data in the target domain is required. Transductive transfer learning has similar source and target tasks, but the domains are different. There is only labelled data available in the source, not in the target domain. When the tasks are different between the target and the source domain, and neither of the domains has labels available, the setting is unsupervised transfer learning [13, 32].

Transfer learning can further be split into four categories based on the question "What to transfer". In instance-based transfer learning, it is assumed that parts of the data in the source domain can be reused for learning in the target domain by employing techniques like reweighting or importance sampling. Feature representation transfer focuses on learning an optimal feature representation for the target domain, where the knowledge to be transferred is encoded into this feature representation. Another method is parameter transfer, which assumes that models for related tasks share

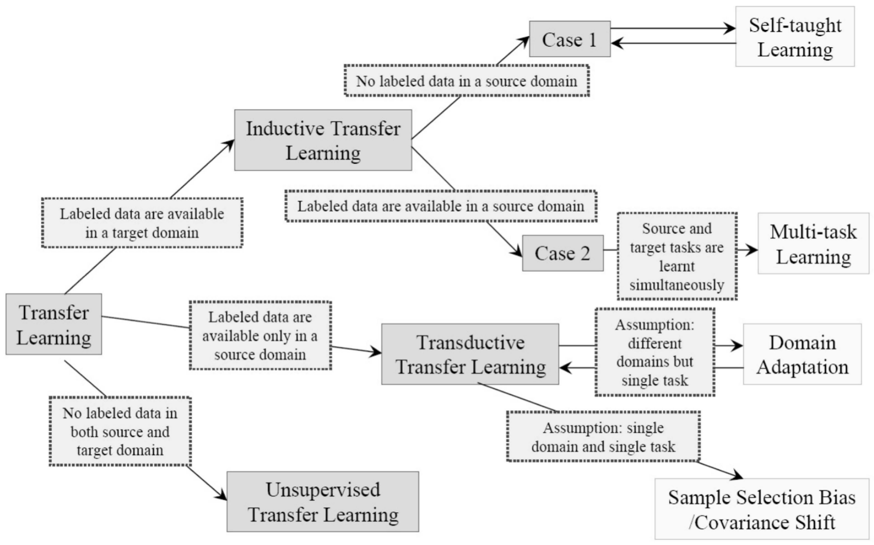

Case 1
Self-taught Learning
No labeled data in a source domain
Inductive Transfer Learning
Labeled data are available in a target domain
Labeled data are available in a source domain
Case 2
Source and target tasks are learnt simultaneously
Multi-task Learning
Transfer Learning
Labeled data are available only in a source domain
Transductive Transfer Learning
Assumption: different domains but single task
Domain Adaptation
No labeled data in both source and target domain
Assumption: single domain and single task
Unsupervised Transfer Learning
Sample Selection Bias /Covariance Shift

Figure 2.2: An overview of different transfer learning methods. Taken from [13].

some parameters or prior distributions of hyperparameters, encoding the transferred knowledge into these shared components. Lastly, relational knowledge transfer knowledge transferred is a similar relationship among data in the source and target domains [32].

Our methodology employs an inductive transfer learning approach, specifically a feature representation transfer. The approach was initially experimentally investigated by Yosinski et al. [34], and we will adhere to the technique presented in their work. First, one has to adapt the given source model, which is already trained on the source domain dataset, to the target domain. This can be done by replacing the final layer of the network with a layer which is suitable for the target domain. For example, if a network is trained on a dataset with 10 labels and one transfers its knowledge to a target domain with 100 labels, the final layer has to be replaced with a layer consisting of 100 neurons instead of 10. The other characteristics of the last layer, such as input features and bias, are taken over from the original classification head. Now, the network could technically be retrained on the target domain dataset. Before doing so, one has to decide which layers are kept and which are retrained. One can perform transfer learning with different configurations, from retraining only the classification head to retraining the whole network besides the very first layer. For retraining k layers of a neural network with depth K, one freezes the first $K - k$ layers and retrains the last k layers. Instead of freezing the first $K - k$ layers, one could also decide to fine-tune the first $K - k$ layers. For freezing the first $K - k$ layers, one sets the learning rate of the neurons in the frozen layers to zero, and for fine-tuning the first $K - k$ layers, the learning rate is reduced by a factor, e.g. 0.01. Usually, retraining is done with conventional training methods, but because of our focus on robustness, we will also investigate the effects of adversarial retraining, i.e. employing the methods discussed in Section 2.4 for retraining the layers.

Chapter 3

Related Work

After we have described the general knowledge about the topics local robustness, verification, robustness distributions, adversarial training and transfer learning, we discuss now some more work, which is directly related to this thesis. This specifically focuses on transfer learning of robustness.

3.1 Adversarially Robust Transfer Learning

Shafahi et al. [14] investigated the transferability of robustness among CIFAR-10, CIFAR-100 and ImageNet networks. They found that robustness in a neural network stems from robust feature extractors, which are resistant to adversarial perturbations. For that finding, they iteratively retrained the last block, then two, and so on until all layers of a Wide-ResNet 32-10 are re-initialised. With blocks, the authors refer to individual ResNet blocks and, for the last two blocks, to the batch norm and average pooling layer for the penultimate block and the fully connected layer for the last block.

Furthermore, they could successfully transfer robustness from a robust CIFAR-100 model to CIFAR-10. For that, they explored three methods: just retraining the last layer, replacing the last layer with a multi-layer perceptron and fine-tuning the whole network with lifelong learning and learning without forgetting methods. They could successfully transfer robustness with all three methods. They found that the more similar (in terms of distributions, number of classes, etc.) the source and target models are, the better the robustness transfers. The transfer of robustness by just retraining the last layer caused a drop in validation accuracy. To mitigate this, the authors studied the effect of replacing the last layer with a multi-layer perceptron instead of just adapting it to the target domain and retraining it. They found that adding just one additional hidden layer with 2048 neurons led the model to achieve 100% training accuracy. However, adding more layers only improved training accuracy but did not significantly improve validation accuracy. The authors assume overfitting to be the reason for this behaviour. Also, they found that adding more hidden layers improved the robustness to PGD attacks. This shows that adding additional layers does not hurt robustness [14].

They further examined another approach to solve the described generalisation issue. When simply retraining the entire network, the target model tends to forget the source model's robustness. Instead, they proposed using lifelong learning methods, especially learning without forgetting. They first retrained the fully connected parameters. Then the learning rate is cut, and both feature extractors and fully connected parameters are retrained. Also, they use a loss function that incorporates a distillation term from the robust source model, causing the feature representations of the source and target networks to remain similar. With this approach, they could improve validation accuracy while preserving the robust feature representations [14].

3.2 Evaluating the Transferability of Local Robustness in Neural Networks

Baumann [15] studied in his thesis whether robustness could be transferred from an adversarially trained MNIST network to EMNIST.

First, he explored which layers of a neural network have an impact on the network's robustness in order to determine which layers he could retrain for the domain adaptation. For that, he retrained the network layer by layer from the back on the same MNIST dataset until he completely retrained the model. He found that the network could be retrained until a specific drop point, at which the critical ε decreased drastically. This drop point was suggested to be the optimal cutoff for transfer learning, as it is a good trade-off between improving generalisability and remaining robustness. A polynomial function was fitted to the dependence of robustness on the number of layers and parameters, and it was shown that it could reliably predict the drop point. The author also found an unexpected increase of robustness in non-ReLU networks (specifically Tanh), when the networks were fully retrained [15].

The same experiment was repeated with retraining of an adversarially pre-trained MNIST model on the balanced EMNIST dataset for all possible cutoffs. It was shown that robustness could be transferred using this transfer learning approach. The robustness and accuracy metrics of the different cutoffs were compared with the predicted best cutoff using the drop point method from the previous experiment. It was validated that the optimal cutoff lays precisely on the calculated drop point. Therefore, the author concluded that using the drop point method is a good trade-off between accuracy and robustness. Also, they estimated that the approach using transfer learning led to a training speedup of 3.5 times compared to adversarial training from scratch [15].

The research in my thesis will build upon this work. We will reuse the algorithms used for transfer learning. The scope of my work will be primarily on the training time to confirm that the novel method actually leads to improved computational efficiency. Also, instead of working on the drop point, we will focus more on the comparison of accuracy and robustness between models trained using transfer learning and adversarial training from scratch.

Chapter 4

Methods

In order to investigate the transferability of robustness, we conduct an empirical study. Figure 4.1 shows a brief overview of our methodology.

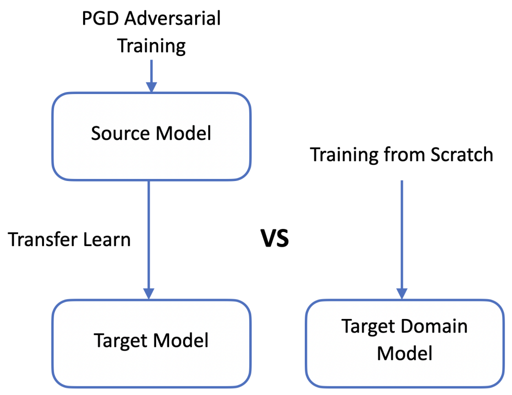


Figure 4.1: This is an overview of our methodology. Transfer learning of an adversarially trained source model to the target domain is performed using conventional retraining. The obtained target model is then compared to an adversarially trained target model regarding model accuracy scores, running times and robustness distributions.

We train neural networks conventionally and adversarially on both a source domain dataset and a target domain dataset, and measure their training time for later comparison. Adversarial training is performed using the PGD algorithm discussed in Section 2.4. We will explain specific settings in Section 5.2. To avoid ambiguity, we will call the neural network trained on the smaller and simpler source domain dataset 'source model' and the neural network trained on the dataset with a similar but extended target domain dataset 'target model'. The adversarially trained source model lays the foundation for further work in this thesis. It is used as a foundation for the following transfer learning step. The adversarially trained target models are later used as a benchmark for comparison of accuracy, robustness and running time. We train the model adversarially using the PGD method explained in Section 2.4.

When we have trained all source and target models, we can perform the transfer learning experiments. Transfer learning is performed using the inductive transfer learning approach with frozen layers as explained in Section 2.5. We perform transfer learning experiments with retraining k layers, $\forall k \in \{1, ..., K\}$, where K is the depth of the model. In later plots and tables, we use the notation TL #k for transfer learned models, which stands for transfer learning by retraining the last k layers. When we obtain the transfer learned models, we compare the models trained with this method to the adversarially trained target model regarding accuracy, running time and robustness distributions. The robustness distributions are created using the attack-based estimation method using PGD explained in Section 2.3. This reduces the running time for creating the distributions significantly, while producing near-optimal upper bounds for ε^*.

We use the tools as represented in the overview in Figure 4.2. For conventional and adversarial training and transfer learning we use the adversarial training box. It saves the training time and directly evaluates the model and stores the clean accuracy of the model. The adversarial training box is accessible on GitHub via this link `https://github.com/Aaron99B/adversarial-training-box`. We improved and adapted it to work better for our specific use case. Also, we included transfer learning scripts that make use of the adversarial training box. The source for all scripts used to perform the experiments and create the plots is provided in this repository: `https://github.com/Skylake143/bachelorthesis_davidwunsch`. Robustness distributions and corresponding plots are created using ADA-VERONA [12].

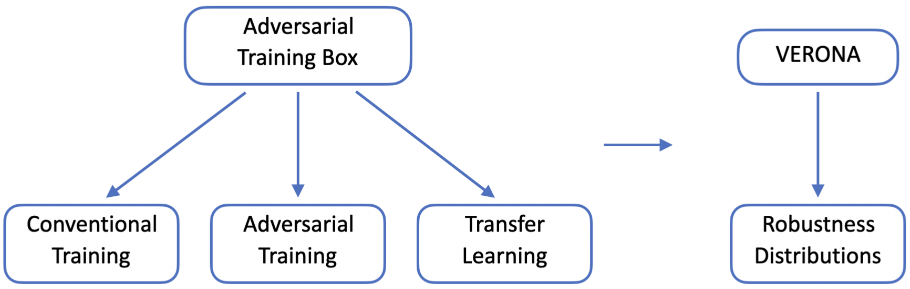


Figure 4.2: An overview of the used tools.

Chapter 5

Experimental Setup

We have explained the methodology for our empirical study in the previous chapter. Now, we show the specific implementation of our methodology. We show which dataset and which architectures we use for the models. Furthermore, we discuss how we adversarially train the networks and how we perform transfer learning. Also, we explain how we obtain the robustness distributions.

5.1 Data and Model Architectures

We perform transfer learning experiments on MNIST to EMNIST and on CIFAR-10 to CIFAR-100. We discuss the architectures used for those datasets and the datasets themselves in the following. An overview of the performed experiments and which network architectures and which datasets are used is presented in Table 5.1.

Table 5.1: Overview of performed experiments, which shows which models are trained on which datasets and transferred to which target domain dataset. All models are adversarially trained using PGD with $k = 40$ iterations on the source dataset. The further details of the training and transfer learning configurations is explained in this chapter.

Source Dataset	**Target Dataset**	**Models**	**Transfer Learning mode**
MNIST	EMNIST	RELU_4_1024, CNN-3, CNN-4, CNN-7	Conventional Retraining
CIFAR-10	CIFAR-100	ResNet-18, ResNet-34, ResNet-50, WideResnet-28-10, WideResnet-32-10	Conventional Retraining
MNIST	EMNIST	CNN-7	Adversarial Retraining
CIFAR-10	CIFAR-100	ResNet-34, WideResNet-34-10	Adversarial Retraining

5.1.1 MNIST and EMNIST

We start our experimentation on the MNIST and EMNIST datasets [35, 36]. These are one of the most researched datasets, and they are small enough that adversarial training times are comparable among multiple models. We choose four model architectures for investigation, one of which is the fully connected mnist_relu_4_1024, as this is large enough to be able to handle the 47 classes of the EMNIST dataset. Furthermore, we explore four different-sized CNN networks. We investigate an arbitrary small CNN network (CNN-3), which has one convolutional and one fully-connected layer. We also investigate the CNN from Madry et al. [29], calling it CNN-4, because of its exceptionally high robustness and because it was used in the paper which established the PGD attack. The network has two convolutional layers and two fully-connected layers. We also use the CNN model suggested by Yang et al. [37], referring to it as CNN-7, since they investigated generalisation of adversarially trained neural networks. Their CNN model consists of four convolutional layers and three fully connected layers. Their CNN model with dropout layers consists of two convolutional layers, two dropout layers and two fully connected layers.

5.1.2 CIFAR-10 and CIFAR-100

We also perform experiments on the CIFAR-10 and CIFAR-100 datasets [38] to see if the results found for MNIST and EMNIST also generalise to other, more complex datasets. We choose five model architectures to explore this. We study three residual networks: ResNet-18, ResNet-34 and ResNet-50 [39]. Residual networks are especially deep due to their large number of filters and have been shown to work well for the CIFAR dataset. Also, its network and layer depth are crucial for successful transfer learning, as more blocks can be adapted when retraining the network to adapt to the target domain dataset.

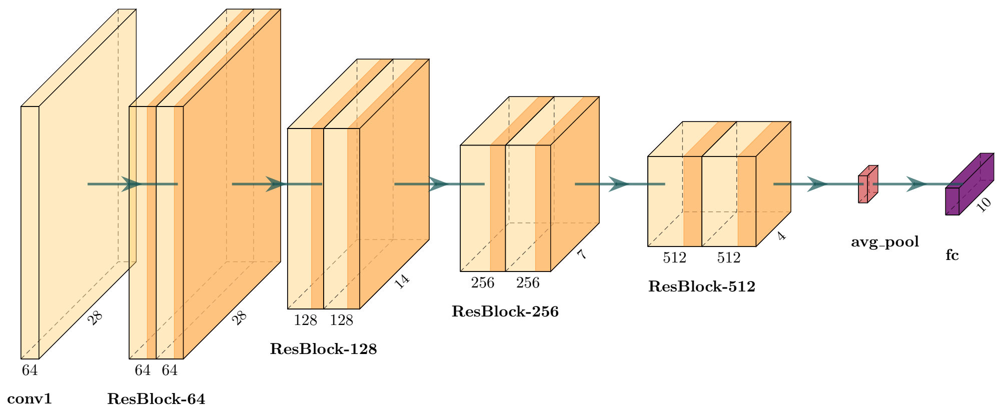


Figure 5.1: Overview of the ResNet-18 network architecture adapted for CIFAR-10. For transfer learning, we retrain those blocks, block by block, starting from the back (fully connected) to the beginning (conv1).

Residual networks consist of seven stages, consisting of an initial convolutional layer, four main residual blocks and a final output head, consisting of an average pooling layer and a fully connected layer such as depicted in Figure 5.1. Residual networks generalise well, which is a requirement for successful transfer learning.

Furthermore, we investigate two Wide Residual Networks (WRN), WRN-28-10 and WRN-34-10, because they have demonstrated state-of-the-art performance on CIFAR-10 and CIFAR-100 [40]. As shown in Figure 5.2, wide residual networks consist of seven stages with one initial convolutional stage and a final output head stage, consisting of a batch normalisation, average pooling and a fully connected layer. The three main stages contain the wide residual blocks.

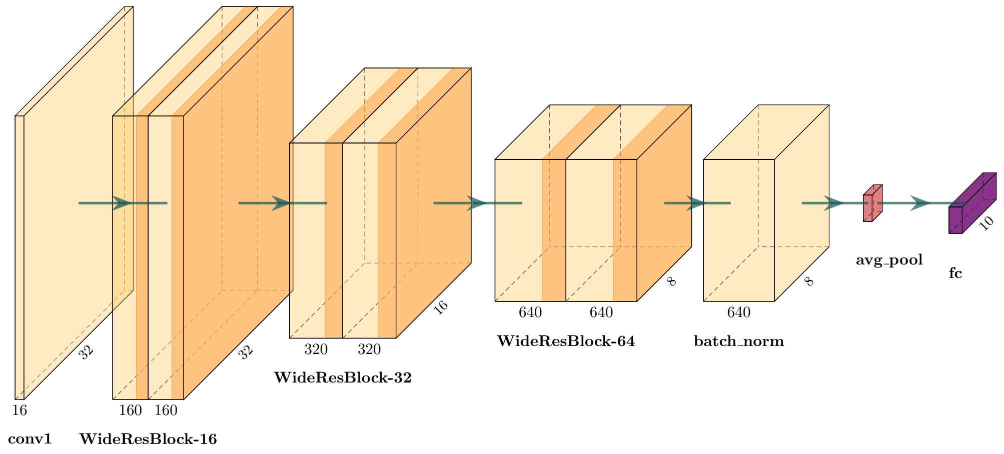


Figure 5.2: Overview of the WideResNet-28-10 network architecture adapted for CIFAR-10. For transfer learning, we retrain those blocks, block by block, starting from the back (fully connected) to the beginning (conv1). A specificity of wide residual networks is that the batch normalisation and average pooling cannot be retrained, because they do not contain trainable parameters, or they would destabilise the retraining of the other layers. They are therefore kept frozen, and the robustness distribution plots for transfer learning in Chapter 6 therefore only consist of five robustness distributions.

5.2 Adversarial Training using PGD

We perform adversarial training by perturbing the training data before the forward pass. The perturbation of the training data is performed using a PGD adversarial attack with $\varepsilon = 0.3$ and 40 iterations as proposed by Madry et al. [29] for MNIST and EMNIST. For CIFAR-10 and CIFAR-100, we use a PGD adversarial attack with $\varepsilon = 8/256$ and 7 iterations. We use an 80/20 split of the training dataset for training and validation sets and use the dataset's test set for final evaluation. This is a slight difference from the training configuration by Madry et al. [29]. They use the full training set for training and use the test set for both validation and final evaluation. This causes our final results to be slightly worse than theirs, although we use the same hyperparameters besides this. It stems from the fact that our models are evaluated on a completely unseen test set while theirs is not. For training of all the MNIST and EMNIST networks, we choose the optimiser Adam and StepLR scheduler, and for the networks trained on CIFAR-10 and CIFAR-100, we use the SGD optimiser and MultiStepLR scheduler. The configuration and the hyperparameters are shown in Table A.2 in the Appendix. The choices are based on the configurations based on the training configurations Madry et al. [29] chose in their initial work. Adversarial training is performed on the ALICE HPC cluster using nodes equipped with Nvidia L4 GPUs with 24GB of VRAM. The jobs are configured to use the GPUs exclusively in order to guarantee honest training time comparisons. The training time is measured GPU-aware using CUDA events and synchronisation. Final testing is performed using an additional test set with multiple adversarial attack configurations.

To guarantee comparability of training time among the networks and to compare it later on with the training time using transfer learning methods, we use early stopping for the MNIST and EMNIST networks. Usually, early stopping is based on the metrics validation accuracy or loss. In adversarial training, we observed that validation robust accuracy still improves when validation accuracy has already converged to a stable value. To ensure that the training is not stopped when robust accuracy could still improve, we set validation robust accuracy as the early stopping metric. Because of the noisy nature of this metric, we also apply a moving average with a window of 10 and set the patience to 6. This configuration turned out to be the most effective way to stop closest when validation robust accuracy converges. For CIFAR-10 and CIFAR-100, we adversarially train the networks for 200 epochs according to the method used by Madry et al. [29]. For ResNet-34 and ResNet-50, we ran into issues in the later phase of the training, and the hyperparameters do not seem to work as well for those networks. Because training time limitations made hyperparameter optimisation unfeasible, we decided to use an early stopper for those two networks, which stopped before the training issues occurred.

5.3 Transfer Learning

For transfer learning, we adhere to the methodology explained in Section 2.5. We first load the source model and create a copy of it. The layers of the copied model are automatically extracted. The last layer is identified and adapted to the target domain, keeping the input features the same and changing the output classes to match the number of classes of the target domain dataset. The network gradients are frozen, and the last k layers or blocks of the network are unfrozen, and their parameters are reset. For residual and wide residual networks, the blocks that we retrain correspond to the blocks in the graphic in Figure 5.1 and Figure 5.2.

The networks are then conventionally retrained with the aforementioned configuration. The rest of the transfer learning script is therefore equivalent to conventional training using an 80/20 split for training and validation split using early stopping, and the dataset's test set for final evaluation. We use the hyperparameter configuration shown in Appendix A. Transfer Learning is performed with the exact same hardware and early stopping configuration as described in Section 5.2. The experiment is performed with all possible numbers of retraining layers $k \in \{1, 2, \ldots, K\}$, where K is the depth of the neural network. This is later used to create accuracy and robustness plots dependent on the number of retrained layers. For CIFAR networks, we changed the batch size for transfer learning from 128 to 256, as this produced better results and accuracy scores.

We furthermore perform additional side experiments to improve the generalisation of the model. Shafahi et al. [14] suggested replacing the last layer of the model with a multi-layer perceptron. We do this by taking over the input features of the last layer of the source model and adapting the output classes to match the number of classes of the target domain. We then try to add between one and three additional layers with 2048 neurons before the classification head.

Additionally, we perform another side experiment by performing transfer learning with adversarial retraining instead of conventional retraining. Layer extraction, adaptation and freezing are performed exactly as with transfer learning using conventional retraining. The retraining of the unfrozen layers is then just done using adversarial training as performed in Section 5.2.

5.4 Robustness Distributions using ADA-VERONA

We create robustness distributions using ADA-VERONA [12]. To determine each critical ε^*, we use the estimation method discussed in Section 2.3 to determine upper bounds for critical ε^* using PGD attacks with $k = 40$ iterations. The minimum adversarial perturbations p^* are searched in a discretised search space of perturbation values $\varepsilon \in \{0.000, 0.005, 0.010, \ldots, 0.800\}$ for MNIST and EMNIST models and $\varepsilon \in \{0.000, 0.001, 0.002, \ldots, 0.110\}$ for CIFAR-10 and CIFAR-100 networks. We made the decision to use the critical ε^* estimation method because complete verification methods would require two to four magnitudes more running time, which would be infeasible for the scope of this thesis. Robustness distributions are created on the test set of the datasets, which contains 10000 instances for MNIST, CIFAR-10 and CIFAR-100 and a random subset of 10000 instances of the 18800 instances big EMNIST test set. The distributions are only created from the instances that are classified correctly if they are unperturbed. The reason for this is that the median p^* would be artificially diminished for models with lower clean accuracy, because misclassified instances would be attributed a p^* of 0.0. This would make a comparison of robustness distributions among multiple models with varying clean accuracy scores impossible.

Chapter 6

Results

In this chapter, we present the results of our empirical study. First, we analyse whether robustness transfers at all. Then we weigh off robustness and accuracy and analyse their dependency. Additionally, we compare the training times for the different approaches. Lastly, we demonstrate the results for the side experiments.

6.1 Robustness transfer

Our first research objective was to investigate the extent to which robustness is transferable. We investigated this by training models conventionally and adversarially from scratch and then performing transfer learning from an adversarially trained source model to the target domain by retraining the network layer by layer conventionally.

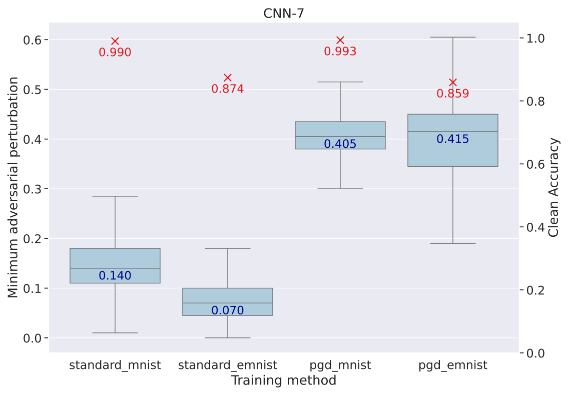
The figures in this chapter show robustness distributions and accuracy scores of models trained from scratch on the left side, and transfer learned models on the right side. The description of the models trained from scratch indicates whether the model was conventionally (standard) or adversarially (pgd) trained, and on which dataset the models were trained and evaluated. The plot on the right side shows the transfer learned models. The number on the x-axis indicates how many layers were retrained. Thus, k indicates that the last k layers are retrained and the first $K - k$ layers are frozen.

All of the figures show similar trends, which indicates that robustness could successfully be transferred for all networks, as detailed in the following.

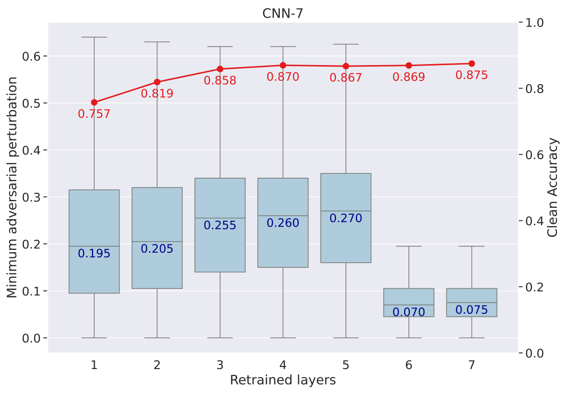
6.1.1 Transfer Learning from MNIST to EMNIST

Figure 6.1 shows the results for the CNN-7 model. We adversarially trained a source model with a median p^* of 0.405. When retraining the models from the back of the network layer by layer, the median p^* increases until we retrained five layers, afterwards it drops completely to the same median p^* as the conventionally trained target model. We could achieve, at best, a median p^* of 0.270 for retraining five layers. This corresponds to two-thirds of the median p^* of the source model, slightly less than four times the median p^* of a conventionally trained target model. The same behaviour can be observed for the CNN-4 network. The results for this are shown in the appendix in Figure B.2. One network architecture was too small to be able to transfer robustness to the same extent, which is also shown in the appendix in Figure B.3.

Although two thirds of the robustness could be transferred with regard to the median of the distribution, it is also important to point out that the distributions are differently shaped. In the box plots (where the whiskers indicate 75% of the datapoints) it becomes apparent that the model is



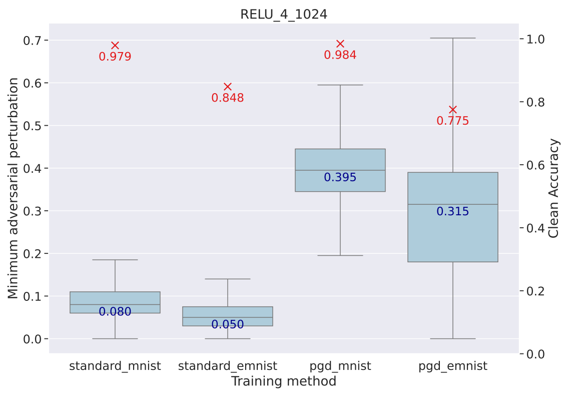
(a) Conventionally and adversarially trained models from scratch.



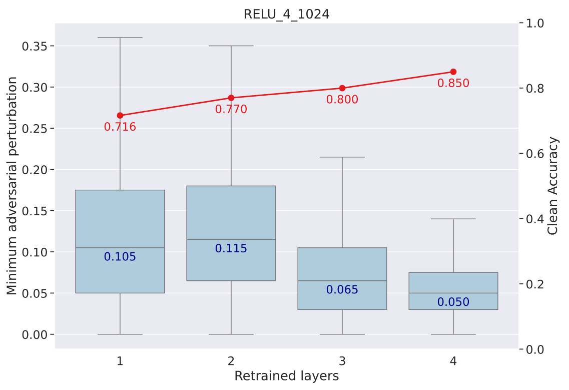
(b) Transfer learned models from adversarially trained source model to target domain with k retrained layers.

Figure 6.1: Accuracy scores and robustness distributions for CNN-7 networks trained on MNIST and EMNIST. One notices that for retraining the models layer by layer from the back, the median p^* increases until five layers, followed by a sharp drop of robustness. At best a median p^* of 0.270 could be achieved for retraining five layers, which corresponds to two thirds of the median p^* of the source model adversarially trained from scratch and is roughly four times the median p^* of a conventionally trained target model.

mostly distributed between 0.2 and 0.6 for the model adversarially trained on EMNIST. This distribution is different for the transfer learned model, which is distributed between 0.0 and 0.65. This behaviour becomes apparent when you create a CDF plot of the different models, which is shown in the appendix in Figure B.1. The models adversarially trained from scratch have a very S-shaped distribution, while the transfer learned models have a more linear distribution. This suggests that there are very few instances (only 20%) with a low p^* of less than 0.3, while there are significantly more instances for the transfer learned models due to their linear nature.



(a) Conventionally and adversarially trained models from scratch.



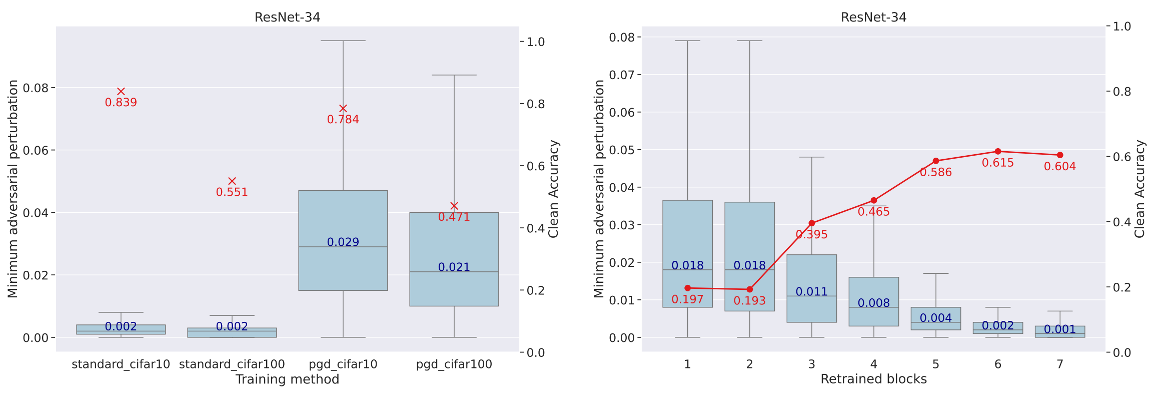
(b) Transfer learned models from adversarially trained source model to target domain with k retrained layers.

Figure 6.2: Accuracy scores and robustness distributions for RELU_4_1024 networks trained on MNIST and EMNIST. The adversarially trained source model has a median p^* of 0.395. At best, 0.115 median p^* could be reached with retraining the last two layers, which is roughly one third of the median p^* of the source model and target model adversarially trained from scratch. It is still more than twice the median p^* of a conventionally trained target model.

One notices that robustness transfer works better for deeper, more complex models. For the two less complex models, robustness could still be transferred, but to a smaller degree. For the RELU_4_1024 network analysed in Figure 6.2, the source model adversarially trained from scratch resulted in a median p^* of 0.395. At best, 0.115 median p^* could be reached by retraining the last two layers. This is slightly less than one-third of the median p^* of the source model and slightly more than one-third of the median p^* of a target model trained from scratch. While these are significantly worse results than for the two more complex models, it is still more than twice the median p^* of a conventionally trained target model.

6.1.2 Transfer Learning from CIFAR-10 to CIFAR-100

For all five of the analysed CIFAR network architectures results were similar. All of the conventionally trained CIFAR-10 and CIFAR-100 models have a median p^* close to zero. The median p^* of most CIFAR architectures shows a gradual decline, in accordance with the work from [14]. This is different compared to transfer learning from MNIST to EMNIST. The CIFAR models seem to require all convolutional blocks to retain maximum robustness. Only retraining the last two blocks seems to be the best way to adapt the model to the target domain. We will analyse two networks in this section; the rest is shown in Appendix B. For the ResNet-34 model adversarially trained from scratch on CIFAR-10, we achieve a median p^* of 0.029 and can retain 0.018 when retraining the last or the last two blocks of the network. This corresponds to approximately 62% of the median p^* of the source model. Also, the median p^* is approximately 85% as high as the median p^* obtained by training a target model from scratch.



(a) Conventionally and adversarially trained models from scratch.

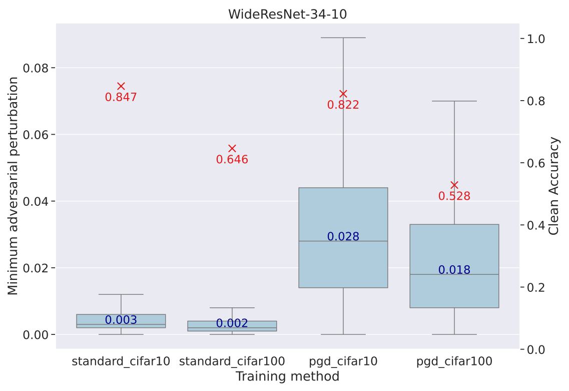
(b) Transfer learned models from adversarially trained source model to target domain with k retrained layers.

Figure 6.3: Accuracy scores and robustness distributions for ResNet-34 networks trained on CIFAR-10 and CIFAR-100. Adversarially training the source model from scratch results in a median p^* of 0.029. When retraining the last or the last two blocks, a median p^* of 0.018 can be retained, which corresponds to approximately 62% of the median p^* of the source model and approximately 85% of training a target model from scratch.

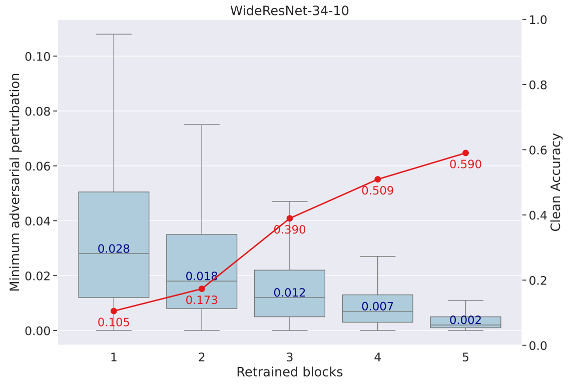
The robustness distributions are very similar for the models trained from scratch and the transfer learned models. Both span from 0.0 to approximately 0.8. Also, the CDF plots depicted in Figure B.5 show a very similar distribution in terms of the shape of the distribution. This would also be logical, because we try to retain the robust feature extractors of the network and only try to adapt the parts of the network that are responsible for adapting to the target domain.

For the WideResNet-34 architecture, we obtained a median p^* of 0.028 for adversarial training from scratch on CIFAR-10. Noticeably, retraining the last block on CIFAR-100 also gives us a median p^* of 0.028, which is even higher than the median p^* of 0.018 for an adversarially trained CIFAR-100 network from scratch.

One can conclude that robustness transfers and that approximately two-thirds of the median p^* of the source model is retained. Next, we want to analyse the dependence and impact of transfer learning on clean accuracy and its relation to robustness.



(a) Conventionally and adversarially trained models from scratch.



(b) Transfer learned models from adversarially trained source model to target domain with k retrained layers.

Figure 6.4: Accuracy scores and robustness distributions for WideResNet-34-10 networks trained on CIFAR-10 and CIFAR-100. We obtain a median p^* of 0.028 for adversarial training the source model from scratch. Retraining the last block also gives us a median p^* of 0.028, which is even higher than the median p^* of 0.018 for adversarially training the target model from scratch.

6.2 Robustness Accuracy Tradeoff

In the previous section, we analysed the transferability of robustness. Next, we want to investigate the tradeoff between robustness and accuracy scores. For that purpose, we create scatter plots which show the median p^* in relation to the clean accuracy for retraining the last k layers of the networks. One notices a fundamentally different behaviour for transfer learning from MNIST to EMNIST compared to transfer learning from CIFAR-10 to CIFAR-100. For the investigated MNIST networks, robustness and accuracy improve until a certain point, followed by a sharp drop in median p^*. The accuracy keeps improving with the number of retrained layers. For CIFAR networks, on the other hand, one notices a linear relationship between median p^* and clean accuracy. For increased clean accuracy, the median p^* decreases. Therefore, one has to weigh off the importance of robustness and accuracy score for CIFAR networks, while one has a specific amount of retrained layers which work best for an MNIST network.

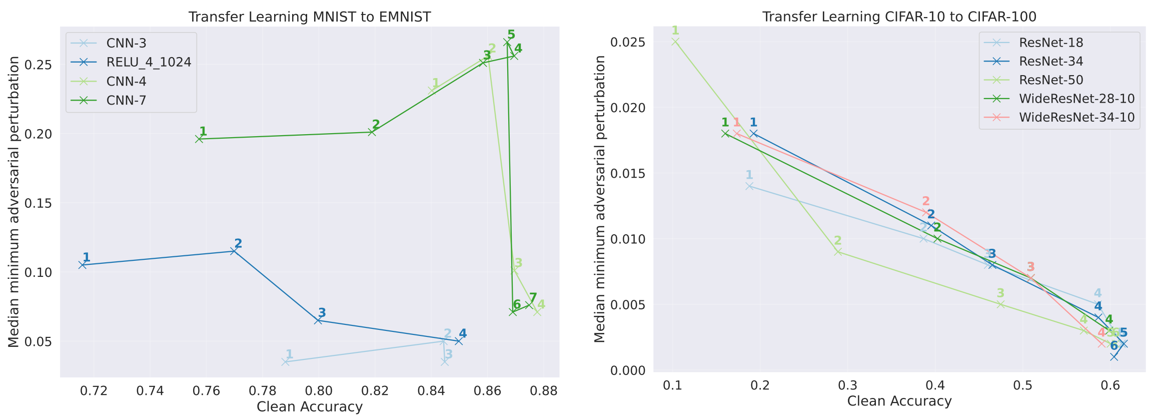


Figure 6.5: Scatter plots of median p^* over clean accuracy of transfer learned models. The numbers on the x-axis indicate the number of retrained layers k from the back of the model. The plots show a different behaviour for transfer learning among MNIST datasets and among CIFAR datasets. Transfer learning among MNIST datasets shows an increased robustness and accuracy score until a certain drop point, while transfer learning among CIFAR datasets shows a linear negative relationship between median p^* and clean accuracy.

All in all, transfer learning using conventional retraining comes with the caveat of mitigated clean accuracy for CIFAR datasets, forcing one to choose a tradeoff between acceptable clean accuracy and robustness. For MNIST datasets, one can reach similar clean accuracy scores, as the relation between median p^* and clean accuracy is different, and one can retrain more layers to adapt better to the target domain. We did some further experiments on adapting the last layer with a multi layer perceptron of either one, two or three additional hidden layers. This method could notably improve accuracy, while the median p^* was only slightly lower. The experiment is shown in Figure B.9.

6.3 Computational Efficiency of Transfer Learning

We analysed the robustness and accuracy scores of the networks. Now we want to analyse whether transfer learning is computationally beneficial compared to adversarial training from scratch. For that, we measure the training time of all experiments we performed and compare them in this section. We show graphic representations of the training times of the previously analysed models in this section and show all other data in Appendix B. In these figures, TL #k indicates transfer learning with conventional retraining of the last k layers.

Adversarially training a CNN-7 network on MNIST took 2197 seconds. Adapting it to EMNIST took between 104 and 211 seconds. Compared to adversarial training of the EMNIST model from scratch, which took 6819 seconds, this is between 32 and 65 times faster.

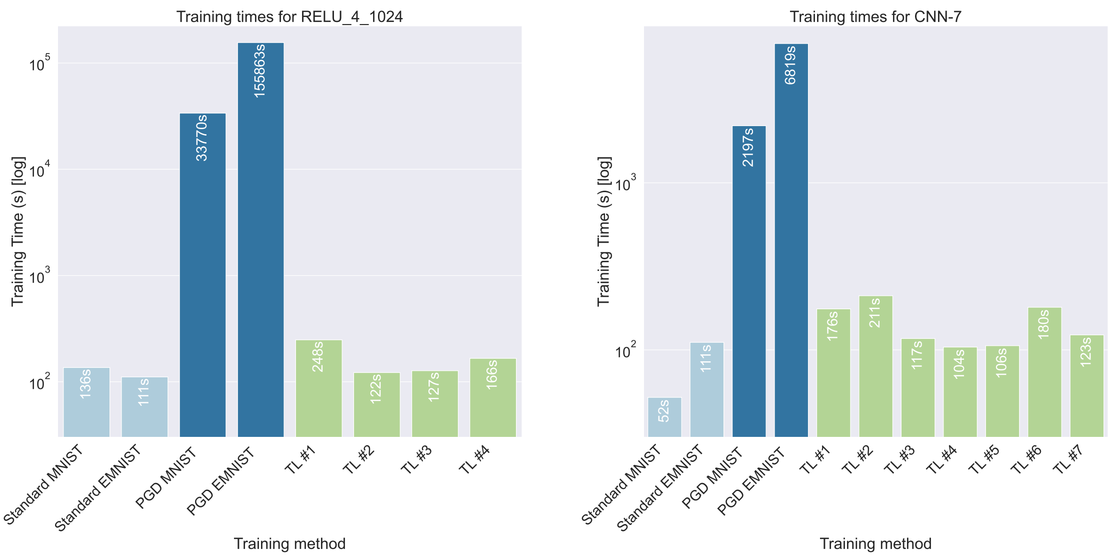


Figure 6.6: Training times for MNIST and EMNIST networks. TL #k indicates transfer learning with conventional retraining of k layers from the back. One notices that adversarially training from scratch takes two to three magnitudes longer than conventional training and transfer learning.

Also, for CIFAR networks, transfer learning is significantly faster than training from scratch. For a ResNet-34 adversarial training of a CIFAR-10 network took 32480 seconds. Transfer learning using conventional retraining took between 746 and 2339 seconds, which is roughly 14 to 44 times faster than adversarial training of a CIFAR-100 network from scratch.

Remarkably, the CIFAR-100 networks are slightly faster to train compared to CIFAR-10. Especially compared to MNIST and EMNIST, where the MNIST models are significantly faster to train. However, this stems from the fact that we use an early stopper, which often stops a bit earlier for MNIST compared to EMNIST, while we use a fixed number of training epochs for CIFAR-10 and CIFAR-100 training.

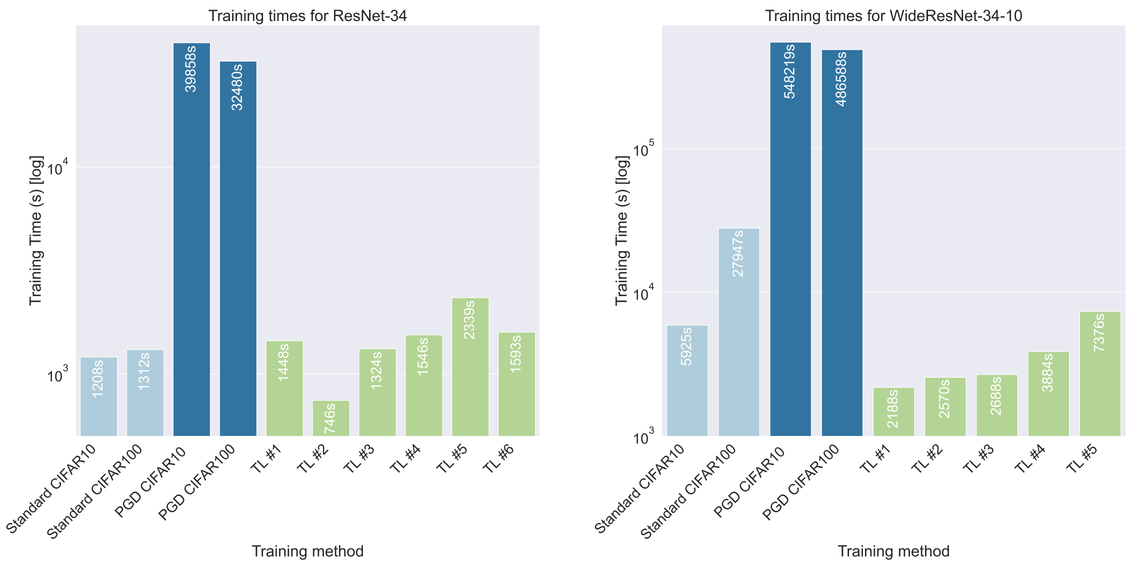
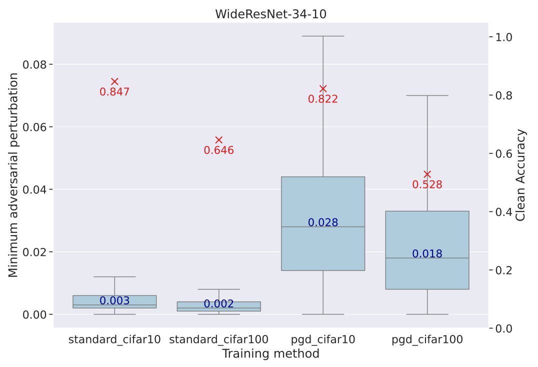


Figure 6.7: Training times for CIFAR-10 and CIFAR-100 networks. TL #*k* indicates transfer learning with conventional retraining of *k* layers from the back. One notices that adversarially training from scratch takes roughly two magnitudes longer than conventional training and transfer learning.

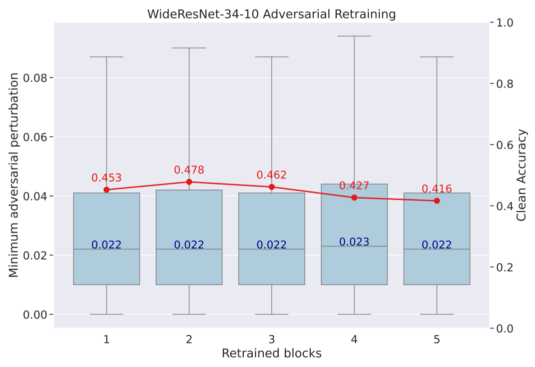
We conclude that transfer learning using conventional retraining is up to three magnitudes faster compared to adversarial training from scratch and therefore provides great training time benefits.

6.4 Transfer Learning using Adversarial Retraining

For transferring robustness from a CIFAR-10 source model to CIFAR-100, we observed that one has to trade off clean accuracy and robustness, as they are in a negative linear relation to each other. Therefore, we were interested in whether this could be solved by adversarially retraining the last blocks instead of conventionally retraining them. We investigate this approach in this section.



(a) Conventionally and adversarially trained models from scratch.



(b) Transfer learned models from adversarially trained source model to target domain with *k* retrained layers using adversarial retraining.

Figure 6.8: Accuracy scores and robustness distributions for WideResNet-34-10 networks trained on CIFAR-10 and CIFAR-100. We obtain a median p^* of 0.028 for adversarial training the source model from scratch. Adversarial retraining resulted in a median p^* of 0.022 for retraining all layers, which is higher than the median p^* of 0.018 for adversarially training the target model from scratch.

We performed transfer learning using adversarial retraining with a fixed number of 30 epochs of a WideResNet-34-10 model. This is the same model as we already analysed in Figure 6.4. With conventional retraining, we achieved a median p^* of 0.028 at 10.5% accuracy. The results for adversarial retraining are shown in Figure 6.8. Using adversarial retraining, the amount of retrained layers did not seem to affect the achieved median p^*, which was approximately 0.022 for all retrained layers. It is interesting that even for five retrained blocks (which is equivalent to training from scratch), the median p^* is higher than the fully trained model from scratch. The robustness distributions of the transfer learned models even span a wider range of 0.0 to 0.09 compared to 0.0 to 0.07 for the CIFAR-100 model adversarially trained from scratch. CDF plots for the models are shown in the appendix in Figure B.8. The accuracy was slightly dependent on the number of retrained blocks and decreased after retraining two blocks. For retraining two blocks, it peaked at 47.8%, which is almost as high as adversarial training from scratch, which resulted in a model with 52.8% accuracy. Therefore, this approach seems to be superior to conventional retraining.

Next, we analyse at what training time cost this superior method comes. In Figure 6.8, we compare the training times for the different WideResNet-34-10 training methods. While transfer learning using adversarial retraining takes more than ten times longer than conventional retraining, it is still ten times faster than adversarial training of a target model from scratch, but delivers a model comparable in accuracy and robustness. This approach may therefore offer an optimal tradeoff.

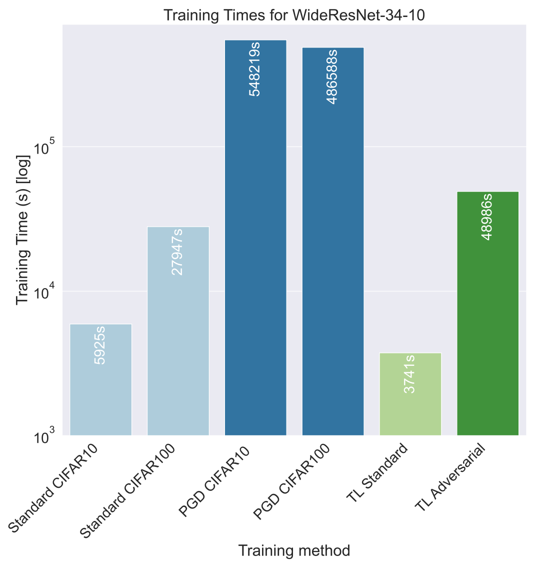


Figure 6.9: Training times for CIFAR-10 and CIFAR-100 networks. TL Standard and TL Adversarial are the average training times for transfer learning using conventional and adversarial retraining. One notices that adversarially training from scratch takes roughly two magnitudes longer than conventional training and transfer learning using conventional retraining, and one magnitude longer than transfer learning using adversarial retraining. Therefore, adversarial retraining is still ten times faster than adversarial training from scratch.

This method seems to be especially beneficial for CIFAR-10 to CIFAR-100 transfer learning. We repeated the experiment for ResNet-34 and could achieve very similar results, which are presented in Figure B.12. We also experimented with transfer learning from MNIST to EMNIST, which is shown in the Figure B.11. We also achieve improvements for median p^*, which are not as big as for transfer learning among CIFAR models.

6.5 Limitations

One limiting factor was the prohibitively expensive nature of this empirical study, as adversarial training and verification for robustness distributions are computationally very heavy operations.

We therefore could not perform hyperparameter optimisation and were forced to use the hyperparameter configurations from the literature. This caused issues for some networks, as the parameters might not be ideal for them. Transfer learning could have been done ideally with hyperparameters which are more suitable for fine-tuning the model. Only the batch size has been adapted slightly, and the rest of the parameters were kept the same to ensure comparability and for training time reasons.

Another issue we encountered was that the performance of the models on clean instances might have an influence on the robustness distributions, as the verifier only creates robustness distributions on instances that are correctly classified if unperturbed. When a model has a low clean accuracy, fewer instances are available for creating the distributions, and therefore, the distributions might not include instances that are susceptible to perturbations. This might lead to artificial improvements of the p^* found for those models.

Chapter 7

Conclusion and Outlook

In this thesis, we performed the following experiments:

- We conventionally and adversarially trained source and target models from scratch
- We performed transfer learning from the adversarially trained source models to the target domain layer by layer
- We analysed the obtained networks with regard to accuracy scores, robustness distributions, and training times

We found that robustness could be partially transferred for most models as long as they are complex enough. Roughly two-thirds of the robustness could be retained for two of the four MNIST neural networks. Similarly, for transfer learning from CIFAR-10 to CIFAR-100, roughly two-thirds of the robustness could be transferred for most networks. For MNIST networks, transfer learning could be performed with achieving the same or slightly worse accuracy as a target model trained from scratch. For CIFAR network the correlation between robustness and accuracy was negative linear, one therefore had to weigh off between accuracy and robustness. With adversarial retraining, we achieved similar or even slightly better robustness distributions than adversarial training of a target model from scratch with slightly worse clean accuracy at considerably less training time.

We further found that transfer learning from an existing source model is computationally significantly more efficient than adversarially training of a target domain model from scratch. For MNIST networks, transfer learning using early stopping was between 35 and 45 times faster than adversarial training from scratch. CIFAR transfer learning was between 25 and 250 times faster than adversarially training from scratch. It is also remarkable that with adversarial retraining, robustness could be fully transferred, and training times were still ten times faster than adversarial training of a target model from scratch.

These conclusions suggest some applications for transfer learning of robustness. Transfer learning using conventional retraining is not useful when one simply wants to acquire the best possible target domain model, and training time does not play a role, as robustness could only be transferred partly. However, if maximum robustness is not crucial and training time is limited, transfer learning is very useful. For example, if one already has access to a robust source model and wants to adapt it to a similar target domain, one could use transfer learning and obtain a robust target domain model with considerably less training time compared to adversarial training from scratch. The approach might moreover be particularly useful if one wants to obtain multiple target domain models and only wants to adversarially train one source model from scratch, as the training time benefits scale up. If the best possible robustness is crucial, transfer learning using adversarial retraining is very useful, as it is still up to ten times faster than adversarial training from scratch.

Some interesting future work would be to further explore transfer learning with adversarial retraining and investigate other transfer learning approaches, such as learning without forgetting, as performed by Shafahi et al. [14]. The authors investigated learning without forgetting for transfer learning of a more complex source model to a simpler target domain and found that it could improve accuracy and generalisation. However, they did not investigate learning without forgetting for transfer learning from a simpler source model to a more complex target domain. It would be interesting to see if this also improves accuracy and generalisation for our use cases. Also, combining this method with replacing the classification head with a multi layer perceptron, such as shown in Figure B.9 would be worth investigating. Both methods improve generalisation and accuracy on the target domain. Therefore, it would be interesting to see if the effects scale up when combined. Furthermore, incorporating the DARD framework recently developed by Zou et al. [30] for transfer learning would be an interesting task. Moreover, extending this research from MNIST and CIFAR datasets to ImageNet and OpenImages would provide even more insights about the generalisation and scaling of this method to larger datasets.

Acknowledgements

This work was performed using the compute resources from the Academic Leiden Interdisciplinary Cluster Environment (ALICE) provided by Leiden University.

I want to thank Holger Hoos for the valuable additional input he gave.

The LLMs Claude Sonnet and Gemini were used to aid the experimental and writing process. All outputs were thoroughly reviewed and corrected. The LLMs were used in the following ways:

- Discuss ideas and ask for explanations of complex concepts. Answers were verified with the literature and available papers.
- Help improving, bug fixing and writing parts of the code. Produced code was always double checked.
- Improve my the writing style and find spelling and grammar errors in my thesis text.

References

[1] O. I. Abiodun, A. B. Jantan, A. E. Omolara, K. V. Dada, N. Mohamed, and H. Arshad. "State-of-the-art in artificial neural network applications: A survey". In: *Heliyon*, 4.11 (2018), e00938.

[2] M. D. V. Thurston, D. H. Kim, and H. K. Wit. "Neural Network Detection of Pacemakers for MRI Safety". In: *Journal of Digital Imaging*, 35.6 (2022), pp. 1673–1680.

[3] L. Cai, J. Gao, and D. Zhao. "A review of the application of deep learning in medical image classification and segmentation." In: *Annals of Translational Medicine*, 8.11 (2020), p. 713.

[4] M. Z. Joel, S. Umrao, E. Chang, R. Choi, D. X. Yang, J. S. Duncan, A. Omuro, R. Herbst, H. M. Krumholz, and S. Aneja. "Using Adversarial Images to Assess the Robustness of Deep Learning Models Trained on Diagnostic Images in Oncology." In: *JCO Clinical Cancer Informatics*, 6 (2022), e2100170.

[5] S. Sharma and K. Guleria. "Deep learning models for image classification: comparison and applications". In: *2nd International Conference on Advance Computing and Innovative Technologies in Engineering, ICACITE*. 2022, pp. 1733–1738.

[6] A. Ivanda, L. Seric, and M. Braovic. "Exploring Applications of Convolutional Neural Networks in Analyzing Multispectral Satellite Imagery: A Systematic Review". In: *Big Data Mining and Analytics*, 8.2 (2025), pp. 407–429.

[7] Z. Yu and T. Ye. "Autonomous traffic sign detection for self-driving car system using convolutional neural network algorithm". In: *Journal of Intelligent & Fuzzy Systems*, 46.3 (2024), pp. 5975–5984.

[8] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. J. Goodfellow, and R. Fergus. "Intriguing properties of neural networks". In: *2nd International Conference on Learning Representations, ICLR*. OpenReview.net, 2014.

[9] T. Bai, J. Luo, J. Zhao, B. Wen, and Q. Wang. "Recent Advances in Adversarial Training for Adversarial Robustness". In: *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence*. ijcai.org, 2021, pp. 4312–4321.

[10] L. Sun, M. Tan, and Z. Zhou. "A survey of practical adversarial example attacks". In: *Cybersecurity*, 1.1 (2018), p. 9.

[11] I. J. Goodfellow, J. Shlens, and C. Szegedy. "Explaining and Harnessing Adversarial Examples". In: *3rd International Conference on Learning Representations, ICLR 2015*. OpenReview.net, 2015.

[12] A. W. Bosman, A. Berger, H. H. Hoos, and J. N. van Rijn. "Robustness Distributions in Neural Network Verification". In: *Journal of Artificial Intelligence Research*, 83 (2025), pp. 1–15.

[13] S. J. Pan and Q. Yang. “A Survey on Transfer Learning”. In: *IEEE Transactions on Knowledge and Data Engineering*, 22.10 (2010), pp. 1345–1359.

[14] A. Shafahi, P. Saadatpanah, C. Zhu, A. Ghiasi, C. Studer, D. W. Jacobs, and T. Goldstein. “Adversarially robust transfer learning”. In: *8th International Conference on Learning Representations, ICLR*. OpenReview.net, 2020.

[15] M. Baumann. “Evaluating the Transferability of Local Robustness in Neural Networks”. MA thesis. Chair for AI Methodology, RWTH Aachen University, 2023.

[16] F. Rosenblatt. “The perceptron: A probabilistic model for information storage and organization in the brain.” In: *Psychological Review*, 65.6 (1958), pp. 386–408.

[17] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, 2016.

[18] M. Dovbnych and M. Plechawska–Wójcik. “A comparison of conventional and deep learning methods of image classification”. In: *Journal of Computer Sciences Institute*, 21 (2021), pp. 303–308.

[19] J. A. Hertz, A. Krogh, and R. G. Palmer. *Introduction to the theory of neural computation*. Vol. 1. Addison-Wesley, 1991, pp. I–XXII, 1–327.

[20] P. Xu, W. Ruan, and X. Huang. “Quantifying safety risks of deep neural networks”. In: *Complex & Intelligent Systems*, 9.4 (2023), pp. 3801–3818.

[21] G. Katz, C. W. 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*. Vol. 10426. Springer, 2017, pp. 97–117.

[22] W. Ruan, M. Wu, Y. Sun, X. Huang, D. Kroening, and M. Kwiatkowska. “Global Robustness Evaluation of Deep Neural Networks with Provable Guarantees for the Hamming Distance”. In: *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI*. ijcai.org, 2019, pp. 5944–5952.

[23] H. Zhang, Y. Yu, J. Jiao, E. P. Xing, L. E. Ghaoui, and M. I. Jordan. “Theoretically Principled Trade-off between Robustness and Accuracy”. In: *Proceedings of the 36th International Conference on Machine Learning, ICML*. Vol. 97. PMLR, 2019, pp. 7472–7482.

[24] L. Li, T. Xie, and B. Li. “SoK: Certified Robustness for Deep Neural Networks”. In: *44th IEEE Symposium on Security and Privacy, SP*. IEEE, 2023, pp. 1289–1310.

[25] J. Liu, L. Chen, A. Miné, H. Yu, and J. Wang. “Input Validation for Neural Networks via Local Robustness Verification”. In: *23rd IEEE International Conference on Software Quality, Reliability, and Security, QRS*. IEEE, 2023, pp. 237–246.

[26] C. Liu, T. Arnon, C. Lazarus, C. A. Strong, C. W. Barrett, and M. J. Kochenderfer. “Algorithms for Verifying Deep Neural Networks”. In: *Foundations and Trends in Optimization*, 4.3-4 (2021), pp. 244–404.

[27] T. Weng, H. Zhang, P. Chen, J. Yi, D. Su, Y. Gao, C. Hsieh, and L. Daniel. “Evaluating the Robustness of Neural Networks: An Extreme Value Theory Approach”. In: *6th International Conference on Learning Representations, ICLR*. OpenReview.net, 2018.

[28] A. Berger, N. Eberhardt, A. W. Bosman, H. Duwe, J. N. van Rijn, and H. Hoos. “Empirical Analysis of Upper Bounds of Robustness Distributions using Adversarial Attacks”. In: *The 19th Learning and Intelligent Optimization Conference*. 2025.

[29] A. Madry, A. Makelov, L. Schmidt, D. Tsipras, and A. Vladu. “Towards Deep Learning Models Resistant to Adversarial Attacks”. In: *6th International Conference on Learning Representations, ICLR*. OpenReview.net, 2018.

[30] J. Zou, S. Zhang, M. Qiu, and C. Li. “DARD: Dice Adversarial Robustness Distillation against Adversarial Attacks”. In: *CoRR*, abs/2509.11525 (2025).

[31] M. Ranaweera and Q. H. Mahmoud. “Virtual to Real-World Transfer Learning: A Systematic Review”. In: *Electronics*, 10.12 (2021).

[32] R. Ribani and M. Marengoni. “A Survey of Transfer Learning for Convolutional Neural Networks”. In: *32nd SIBGRAPI Conference on Graphics, Patterns and Images Tutorials, SIBGRAPI-T*. IEEE, 2019, pp. 47–57.

[33] X. Chen and T. Luo. “Adversarial-Robust Transfer Learning for Medical Imaging via Domain Assimilation”. In: *Advances in Knowledge Discovery and Data Mining - 28th Pacific-Asia Conference on Knowledge Discovery and Data Mining, PAKDD 2024*. Vol. 14648. Springer, 2024, pp. 335–349.

[34] J. Yosinski, J. Clune, Y. Bengio, and H. Lipson. “How transferable are features in deep neural networks?” In: *Advances in Neural Information Processing Systems 27: Annual Conference on Neural Information Processing Systems*. 2014, pp. 3320–3328.

[35] L. Deng. “The MNIST Database of Handwritten Digit Images for Machine Learning Research”. In: *IEEE Signal Processing Magazine*, 29.6 (2012), pp. 141–142.

[36] G. Cohen, S. Afshar, J. Tapson, and A. van Schaik. “EMNIST: Extending MNIST to handwritten letters”. In: *2017 International Joint Conference on Neural Networks (IJCNN)*. 2017, pp. 2921–2926.

[37] Y. Yang, C. Rashtchian, H. Zhang, R. Salakhutdinov, and K. Chaudhuri. “A Closer Look at Accuracy vs. Robustness”. In: *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems, NeurIPS*. Curran Associates, Inc., 2020.

[38] A. Krizhevsky. “Learning Multiple Layers of Features from Tiny Images”. In: *Technical Report, University of Toronto* (2009), pp. 32–33.

[39] K. He, X. Zhang, S. Ren, and J. Sun. “Deep Residual Learning for Image Recognition”. In: *IEEE Conference on Computer Vision and Pattern Recognition, CVPR*. IEEE Computer Society, 2016, pp. 770–778.

[40] S. Zagoruyko and N. Komodakis. “Wide Residual Networks”. In: *Proceedings of the British Machine Vision Conference, BMVC*. BMVA Press, 2016.

Appendix A

Hyperparameters and Verification Parameters

Table A.1: Hyperparameters used for MNIST and EMNIST conventional training and transfer learning.

Parameter	Value
Learning rate	0.001
Weight decay	0.0001
Batch size	256
Scheduler step size	10
Scheduler gamma	0.98
Patience	6
Max epoch	300

Table A.2: Hyperparameters used for MNIST and EMNIST adversarial training.

Parameter	Value
Learning rate	0.001
Weight decay	0.0001
Batch size	256
Scheduler step size	10
Scheduler gamma	0.98
Attack epsilon	0.3
Patience	6
Max epoch	300

Table A.3: Hyperparameters used for MNIST and EMNIST transfer learning using adversarial retraining.

Parameter	Value
Learning rate	0.001
Weight decay	0.0001
Batch size	256
Scheduler step size	10
Scheduler gamma	0.98
Patience	6
Max epoch	30

Table A.4: Hyperparameters used for CIFAR-10 and CIFAR-100 conventional training. For ResNet-34, ResNet-50, WideResNet-28-10 and WideResNet-34-10 we used early stopping with patience 6, because the training crashed for the later smaller learning rates. All other models were fully trained until epoch 200.

Parameter	Value
Learning rate	0.1
Weight decay	0.0005
Batch size	128
Scheduler milestones	[60,120,160]
Scheduler gamma	0.2
Max epoch	200

Table A.5: Hyperparameters used for CIFAR-10 and CIFAR-100 adversarial training. For ResNet-34 and ResNet-50 we used early stopping with patience 6, because the training crashed for the later smaller learning rates. All other models were fully trained until epoch 200.

Parameter	Value
Learning rate	0.1
Weight decay	0.0005
Batch size	128
Scheduler milestones	[60,120,160]
Scheduler gamma	0.2
Attack epsilon	8/255
Max epoch	200

Table A.6: Hyperparameters used for CIFAR-10 and CIFAR-100 transfer learning using conventional retraining. Early stopping was activated with patience 6.

Parameter	Value
Learning rate	0.1
Weight decay	0.0005
Batch size	256
Scheduler milestones	[60,120,160]
Scheduler gamma	0.2
Max epoch	200

Table A.7: Hyperparameters used for CIFAR-10 and CIFAR-100 transfer learning using adversarial retraining.

Parameter	Value
Learning rate	0.1
Weight decay	0.0005
Batch size	256
Max epoch	30

Table A.8: Verification parameters for MNIST and EMNIST networks.

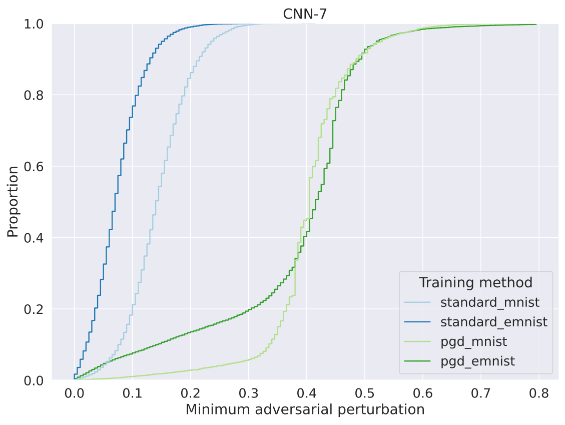
Parameter	Value
Epsilon search space	[0.0, 0.8]
Epsilon step size	0.005
Attack	PGD-40

Table A.9: Verification parameters for CIFAR-10 and CIFAR-100 networks.

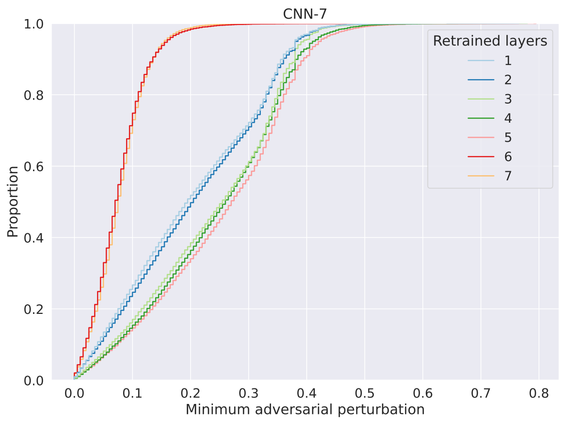
Parameter	**Value**
Epsilon search space	[0.0, 28/256]
Epsilon step size	0.001
Attack	PGD-40

Appendix B

Results

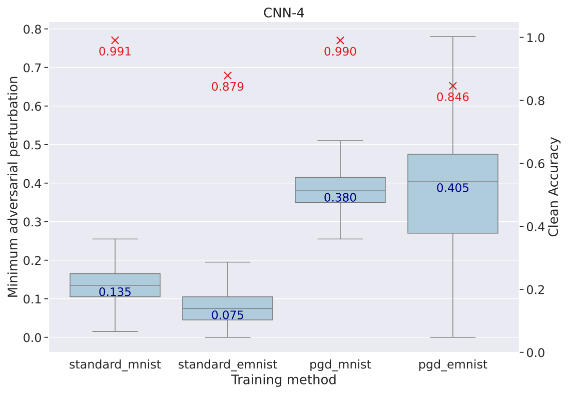


(a) Conventionally and adversarially trained models from scratch.

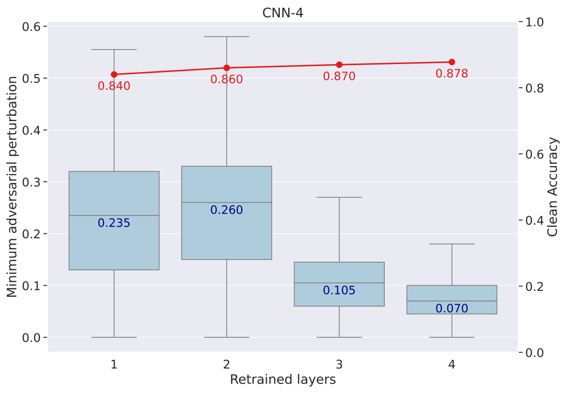


(b) Transfer learned models from adversarially trained source model to target domain with k retrained layers.

Figure B.1: CDF plots for CNN-7 networks trained on MNIST and EMNIST. The models adversarially trained from scratch show a more S-shaped robustness distribution while the transfer learned models show a more linear distribution.

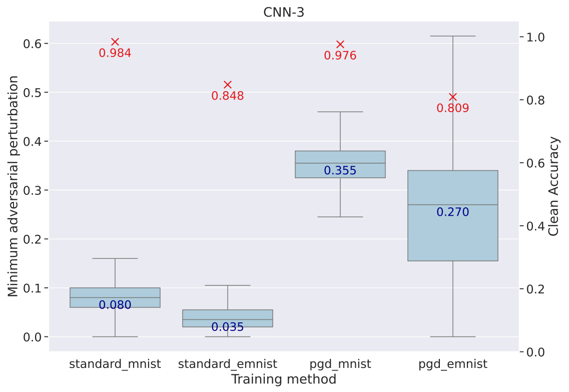


(a) Conventionally and adversarially trained models from scratch.

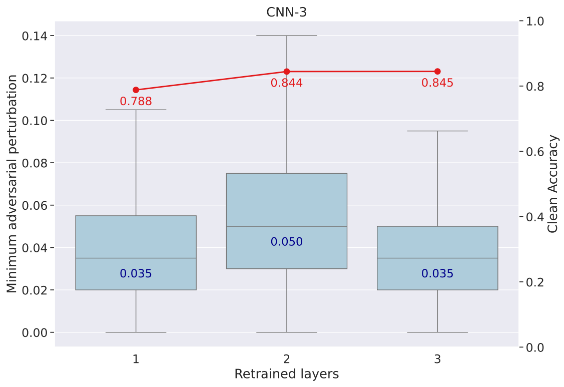


(b) Transfer learned models from adversarially trained source model to target domain with k retrained layers.

Figure B.2: Accuracy scores and robustness distributions for CNN-4 networks trained on MNIST and EMNIST. An adversarially trained EMNIST model from scratch has a median p^* of 0.405, while transfer learning resulted in a median p^* of 0.260 for retraining two layers, which is roughly two thirds of the median p^* of the model trained from scratch.



(a) Conventionally and adversarially trained models from scratch.

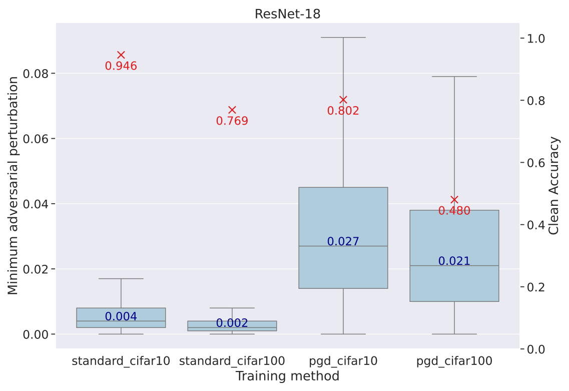


(b) Transfer learned models from adversarially trained source model to target domain with k retrained layers.

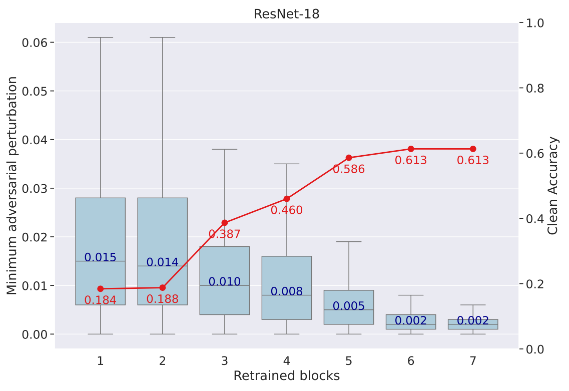
Figure B.3: Accuracy scores and robustness distributions for CNN-3 networks trained on MNIST and EMNIST. One notices that transfer learning was not as successful for this architecture. At best a median p^* of 0.050 could be achieved for retraining the last two layers, which corresponds to less than a fifth of the median p^* of the source model adversarially trained from scratch and is only slightly better than the median p^* of a conventionally trained target model.

Table B.1: Detailed results for RELU_4_1024 architecture.

Model	Accuracy	Median p*	Training Time	Early Stopping Epoch
Standard MNIST	97.89%	0.08	136s	16
Standard EMNIST	84.80%	0.05	111s	12
PGD MNIST	98.42%	0.395	33770s	166
PGD EMNIST	77.51%	0.315	155863s	217
TL #1	71.59%	0.105	248s	28
TL #2	76.99%	0.115	122s	15
TL #3	79.97%	0.065	127s	15
TL #4	84.98%	0.05	166s	19

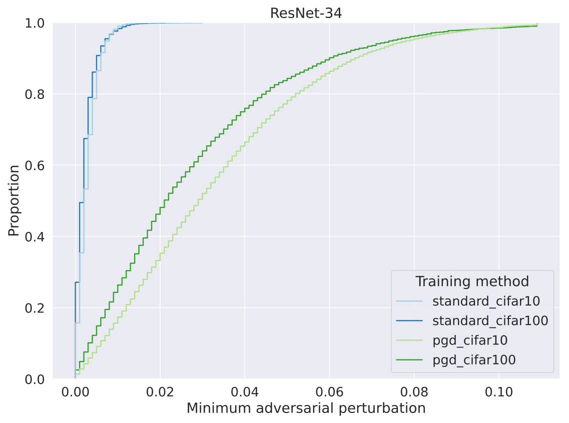


(a) Conventionally and adversarially trained models from scratch.

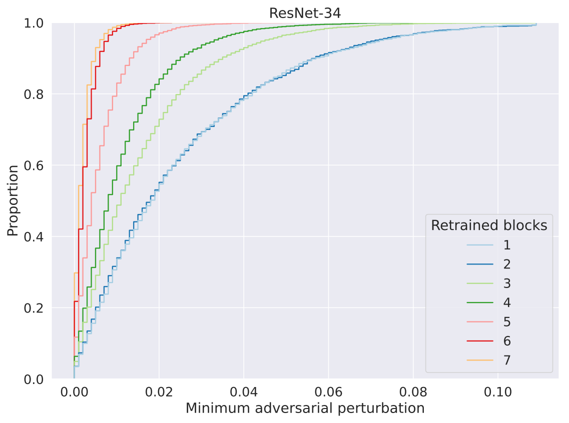


(b) Transfer learned models from adversarially trained source model to target domain with k retrained layers.

Figure B.4: Accuracy scores and robustness distributions for ResNet-18 networks trained on CIFAR-10 and CIFAR-100. At best a median p^* of 0.015 could be achieved for retraining one block, which corresponds to slightly more than two thirds of the median p^* of the source model adversarially trained from scratch and is roughly eight times the median p^* of a conventionally trained target model.



(a) Conventionally and adversarially trained models from scratch.

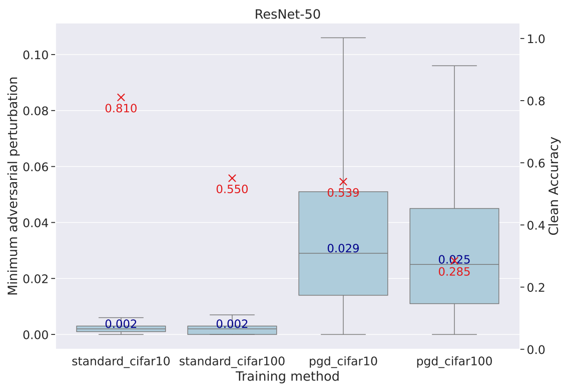


(b) Transfer learned models from adversarially trained source model to target domain with k retrained layers.

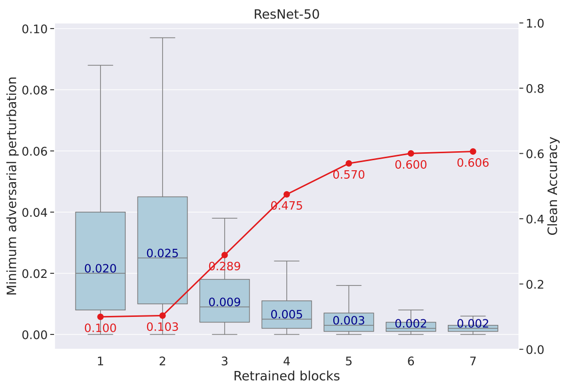
Figure B.5: CDF plots for ResNet-34 networks trained on CIFAR-10 and CIFAR-100. Here the distributions look very similar.

Table B.2: Detailed results for CNN-3 architecture.

Model	Accuracy	Median p*	Training Time	Early Stopping Epoch
Standard MNIST	98.37%	0.08	156s	18
Standard EMNIST	84.79%	0.035	141s	14
PGD MNIST	97.63%	0.355	11549s	74
PGD EMNIST	80.90%	0.27	24537s	70
TL #1	78.81%	0.035	488s	48
TL #2	84.44%	0.05	104s	11
TL #3	84.48%	0.035	200s	20

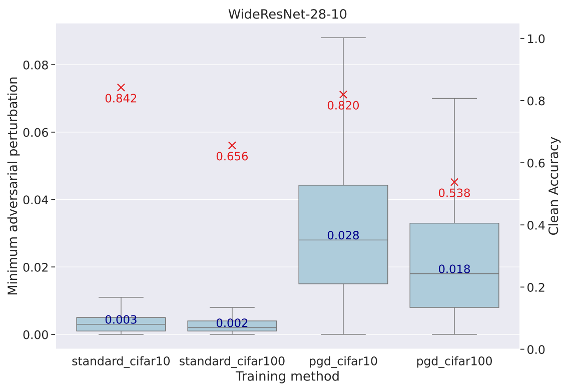


(a) Conventionally and adversarially trained models from scratch.

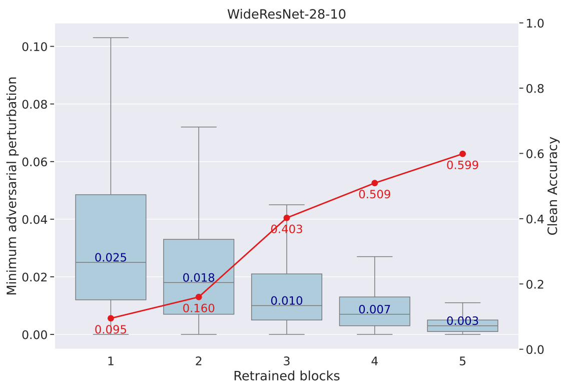


(b) Transfer learned models from adversarially trained source model to target domain with k retrained layers.

Figure B.6: Accuracy scores and robustness distributions for ResNet-50 networks trained on CIFAR-10 and CIFAR-100. At best a median p^* of 0.025 could be achieved for retraining two blocks, which is equal to the median p^* of the source model adversarially trained from scratch and is roughly 13 times the median p^* of a conventionally trained target model.



(a) Conventionally and adversarially trained models from scratch.

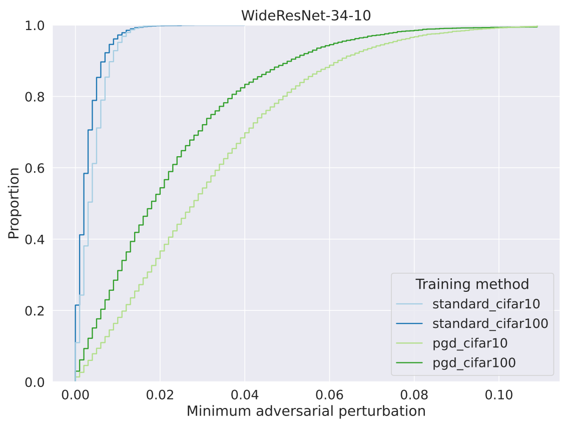


(b) Transfer learned models from adversarially trained source model to target domain with k retrained layers.

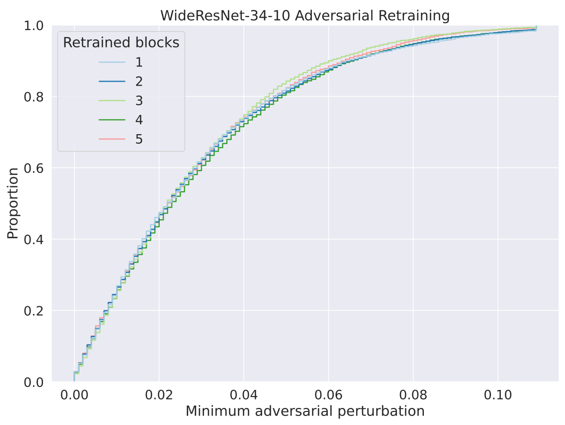
Figure B.7: Accuracy scores and robustness distributions for WideResNet-28-10 networks trained on CIFAR-10 and CIFAR-100. At best a median p^* of 0.025 could be achieved for retraining one block, which is even more than the median p^* of the source model adversarially trained from scratch and is roughly 13 times the median p^* of a conventionally trained target model.

Table B.3: Detailed results for CNN-4 architecture.

Model	Accuracy	Median p*	Training Time	Early Stopping Epoch
Standard MNIST	99.06%	0.136	122s	20
Standard EMNIST	87.88%	0.076	146s	14
PGD MNIST	99.03%	0.381	7052s	42
PGD EMNIST	84.59%	0.406	8858s	41
TL #1	84.02%	0.231	190s	21
TL #2	86.02%	0.256	166s	18
TL #3	86.96%	0.101	177s	18
TL #4	87.77%	0.071	161s	15



(a) Conventionally and adversarially trained models from scratch.



(b) Transfer learned models from adversarially trained source model to target domain with k retrained layers using adversarial retraining.

Figure B.8: CDF plots for WideResNet-34-10 networks trained on CIFAR-10 and CIFAR-100. All of the transfer learned models resemble each other very closely.

Table B.4: Detailed results for CNN-7 architecture.

Model	Accuracy	Median p*	Training Time	Epochs
Standard MNIST	98.98%	0.140	52s	12
Standard EMNIST	87.38%	0.070	111s	14
PGD MNIST	99.29%	0.405	2197s	36
PGD EMNIST	85.89%	0.415	6819s	58
TL #1	75.74%	0.195	176s	27
TL #2	81.89%	0.205	211s	31
TL #3	85.84%	0.255	117s	17
TL #4	86.95%	0.260	104s	15
TL #5	86.70%	0.270	106s	15
TL #6	86.90%	0.070	180s	23
TL #7	87.48%	0.075	123s	16
TL #1 Adversarial	73.75%	0.265	1357s	20
TL #2 Adversarial	81.37%	0.315	1446s	20
TL #3 Adversarial	85.62%	0.36	1778s	20
TL #4 Adversarial	86.25%	0.38	1760s	20
TL #5 Adversarial	86.72%	0.38	1510s	20
TL #6 Adversarial	84.56%	0.37	1757s	20
TL #7 Adversarial	84.80%	0.39	1745s	20

Table B.5: Detailed results for ResNet-18 architecture.

Model	Accuracy	Median p*	Training Time	Epochs
Standard CIFAR10	94.60%	0.004	25855s	200
Standard CIFAR100	76.91%	0.002	18034s	200
PGD CIFAR10	80.18%	0.027	88915s	200
PGD CIFAR100	48.03%	0.021	95024s	200
TL #1	18.43%	0.015	451s	25
TL #2	18.77%	0.014	660s	34
TL #3	38.66%	0.01	1052s	50
TL #4	46%	0.008	1192s	51
TL #5	58.58%	0.005	1453s	56
TL #6	61.3%	0.002	1168s	39
TL #7	61.29%	0.002	919s	30

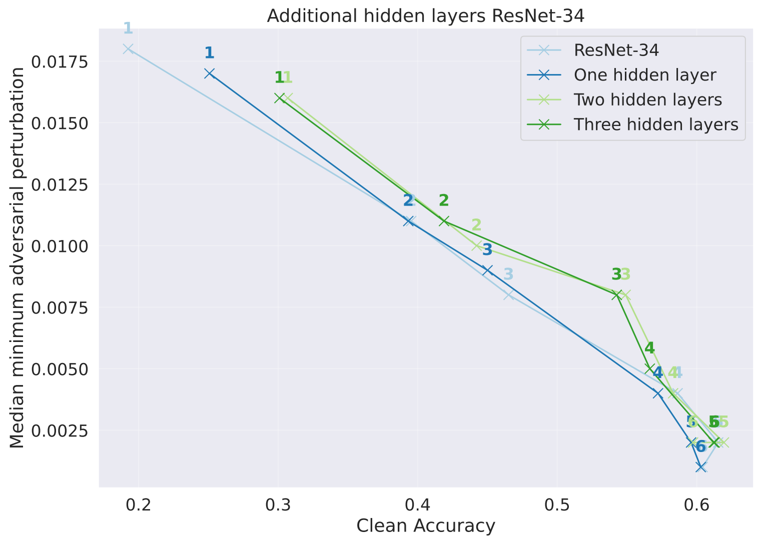


Figure B.9: Scatter plot of median p^* over clean accuracy for transfer learned ResNet-34 models. The light blue line shows the model obtained using conventional retraining; the other lines are models with additional hidden layers of 2048 neurons each. Adding hidden layers was performed by replacing the last layer with a multi layer perceptron and adapting the last layer to the target domain. One notices that adding especially two or three hidden layers could improve the performance of the models.

Table B.6: Detailed results for ResNet-34 architecture.

Model	Accuracy	Median p*	Training Time	Epochs
Standard CIFAR10	83.92%	0.002	1208s	22
Standard CIFAR100	55.06%	0.002	1312s	24
PGD CIFAR10	78.44%	0.029	39858s	84
PGD CIFAR100	47.08%	0.021	32480s	84
TL #1	19.67%	0.018	1448s	46
TL #2	19.25%	0.018	746s	26
TL #3	39.54%	0.011	1324s	44
TL #4	46.52%	0.008	1546s	45
TL #5	58.63%	0.004	2339s	56
TL #6	61.54%	0.002	1593s	34
TL #7	60.42%	0.001	2032s	42
TL #1 Adversarial	44.73%	0.023	16116s	40
TL #2 Adversarial	44.22%	0.023	17038s	40
TL #3 Adversarial	45.28%	0.022	14617s	40
TL #4 Adversarial	42.29%	0.025	15068s	40
TL #5 Adversarial	38.06%	0.023	13826s	40
TL #6 Adversarial	37.12%	0.026	14265s	40
TL #7 Adversarial	35.48%	0.025	15063s	40
TL #1 MLP1	25.79%	0.017	1002s	34
TL #2 MLP1	25.08%	0.017	1291s	43
TL #3 MLP1	39.33%	0.011	1714s	54
TL #4 MLP1	45.01%	0.009	1129s	32
TL #5 MLP1	57.23%	0.004	1933s	46
TL #6 MLP1	59.6%	0.002	1574s	34
TL #7 MLP1	60.84%	0.001	1923s	40
TL #1 MLP2	29.95%	0.016	764s	27
TL #2 MLP2	30.69%	0.016	1651s	54
TL #3 MLP2	44.23%	0.01	2702s	84
TL #4 MLP2	54.91%	0.008	3160s	80
TL #5 MLP2	58.31%	0.004	1926s	44
TL #6 MLP2	61.95%	0.002	1870s	37
TL #7 MLP2	59.74%	0.002	1189s	24
TL #1 MLP3	30.96%	0.015	1279s	39
TL #2 MLP3	30.1%	0.016	717s	24
TL #3 MLP3	41.89%	0.011	1510s	46
TL #4 MLP3	54.27%	0.008	4379s	98
TL #5 MLP3	56.64%	0.005	2003s	43
TL #6 MLP3	61.21%	0.002	1812s	36
TL #7 MLP3	61.28%	0.002	1572s	31

Table B.7: Detailed results for ResNet-50 architecture.

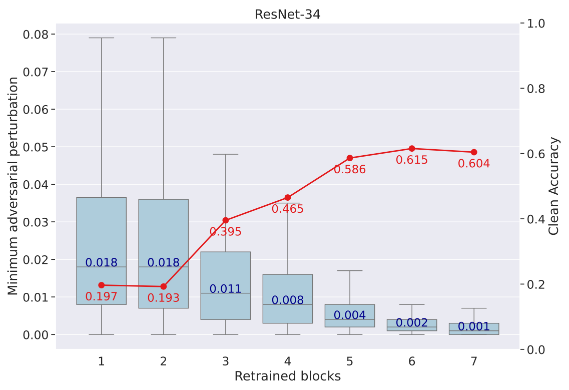
Model	Accuracy	Median p*	Training Time	Epochs
Standard CIFAR10	81.04%	0.002	3408s	33
Standard CIFAR100	55.04%	0.002	3331s	32
PGD CIFAR10	53.91%	0.029	41226s	47
PGD CIFAR100	28.49%	0.025	38343s	55
TL #1	9.95%	0.02	921s	21
TL #2	10.32%	0.025	1406s	30
TL #3	28.88%	0.009	1606s	31
TL #4	47.47%	0.005	1820s	32
TL #5	56.96%	0.003	3490s	47
TL #6	60.02%	0.002	2785s	33
TL #7	60.61%	0.002	2871s	33

Table B.8: Detailed results for WideResNet-28-10 architecture.

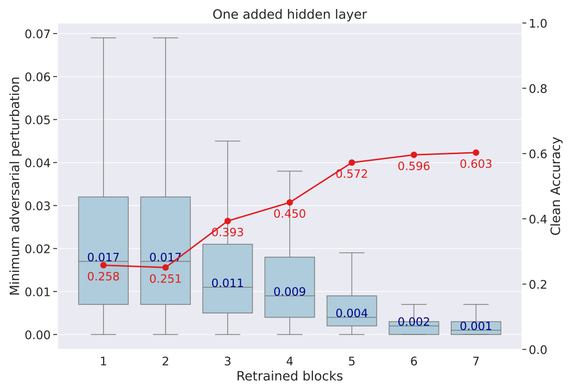
Model	Accuracy	Median p*	Training Time	Epochs
Standard CIFAR10	84.22%	0.003	4502s	26
Standard CIFAR100	65.58%	0.002	20437s	84
PGD CIFAR10	81.95%	0.028	492196s	200
PGD CIFAR100	53.78%	0.018	417888s	200
TL #1	9.48%	0.025	2007s	29
TL #2	16.01%	0.018	1376s	20
TL #3	40.25%	0.01	3911s	44
TL #4	50.92%	0.007	4253s	36
TL #5	59.87%	0.003	5615s	37

Table B.9: Detailed results for WideResNet-34-10 architecture.

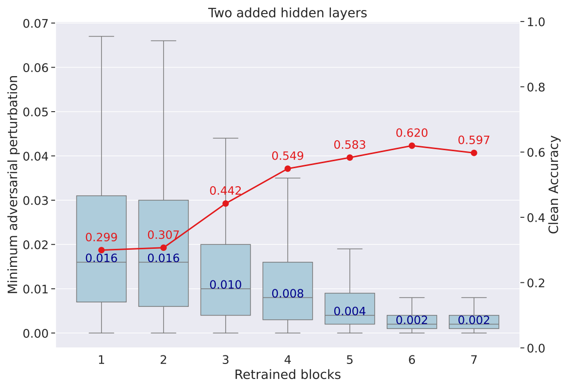
Model	Accuracy	Median p*	Training Time	Epochs
Standard CIFAR10	84.65%	0.003	5925s	28
Standard CIFAR100	64.61%	0.002	27947s	84
PGD CIFAR10	82.19%	0.028	548219s	200
PGD CIFAR100	52.83%	0.018	486588s	200
TL #1	10.52%	0.028	2188s	26
TL #2	17.33%	0.018	2570s	30
TL #3	38.95%	0.012	2688s	25
TL #4	50.93%	0.007	3884s	28
TL #5	59.01%	0.002	7376s	38
TL #1 Adversarial	45.26%	0.022	51060s	30
TL #2 Adversarial	47.82%	0.022	50025s	30
TL #3 Adversarial	46.17%	0.022	48511s	30
TL #4 Adversarial	42.69%	0.023	47176s	30
TL #5 Adversarial	41.64%	0.022	48161s	30



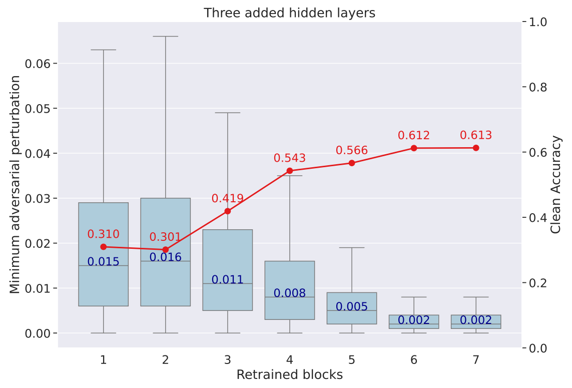
(a) Transfer learned ResNet-34 model with k retrained layers.



(b) Transfer learned ResNet-34 model with one additional hidden layer with 2048 neurons.

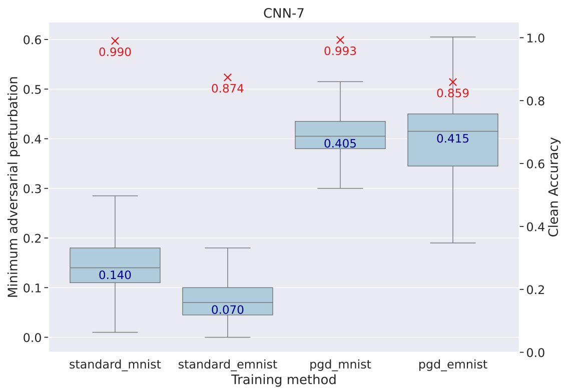


(c) Transfer learned ResNet-34 model with two additional hidden layers of 2048 neurons each.

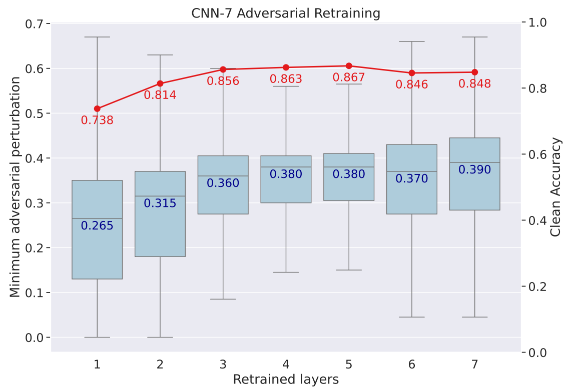


(d) Transfer learned ResNet-34 model with three additional hidden layers of 2048 neurons each.

Figure B.10: Accuracy scores and robustness distributions for transfer learned ResNet-34 models with added hidden layers of 2048 neurons each. Adding hidden layers was performed by replacing the last layer with a multi layer perceptron and adapting the last layer to the target domain. One notices that adding hidden layers could improve the accuracy of the models while slightly decreasing median p^*.

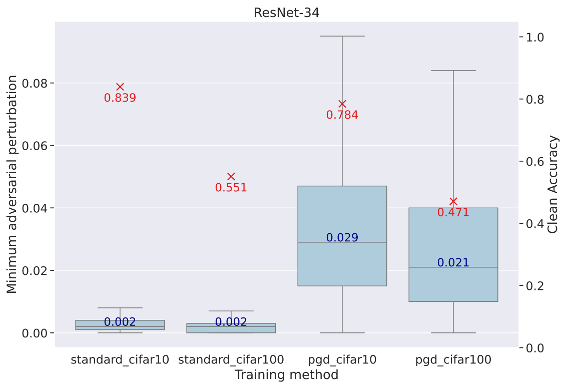


(a) Conventionally and adversarially trained models from scratch.

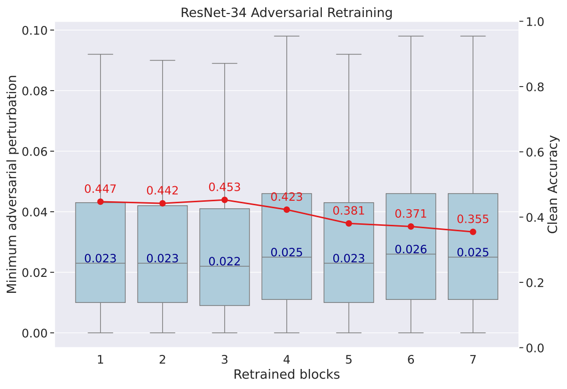


(b) Transfer learned models from adversarially trained source model to target domain with k retrained layers using adversarial retraining.

Figure B.11: Accuracy scores and robustness distributions for CNN-7 networks trained on MNIST and EMNIST. One notices that for retraining the models from the back layer by layer, the median p^* increases until five layers, followed by a sharp drop in robustness. At best, a median p^* of 0.266 could be achieved for retraining five layers, which corresponds to two-thirds of the median p^* of the source model adversarially trained from scratch and is roughly four times the median p^* of a conventionally trained target model.



(a) Conventionally and adversarially trained models from scratch.



(b) Transfer learned models from adversarially trained source model to target domain with k retrained layers using adversarial retraining.

Figure B.12: Accuracy scores and robustness distributions for ResNet-34 networks trained on CIFAR-10 and CIFAR-100.