Radboud University

Faculty of Social Sciences

Approximating Black-Box Deep Neural Networks using Active Learning as a Proxy Measurement for Robustness

Master's Thesis in Artificial Intelligence

Author:

ing. Christoph Schmidl
s4226887

Internal supervisor:

Dr. Makiko Sadakata
Radboud University

Affiliated supervisor:

Dr. Nils H. Jansen
Radboud University

Second reader:

Dr. Giulio Mecacci
Radboud University

March 23, 2021

Contents

1 Introduction **4**
- 1.1 Scope and Research Questions . . . 5
- 1.2 Thesis Structure . . . 5

2 Background **7**
- 2.1 Information Theory . . . 7
- 2.2 Neural Networks . . . 8
  - 2.2.1 Convolutional Neural Networks . . . 11
  - 2.2.2 Residual Neural Networks . . . 15
- 2.3 Active Learning . . . 17
- 2.4 Adversarial Attacks . . . 21
- 2.5 Classifier Stealing . . . 23

3 Data **25**
- 3.1 MNIST . . . 25
  - 3.1.1 Exploratory Data Analysis . . . 25
- 3.2 CIFAR-10 . . . 28
  - 3.2.1 Exploratory Data Analysis . . . 28
- 3.3 GTSRB . . . 31
  - 3.3.1 Exploratory Data Analysis . . . 31

4 Methodology **36**
- 4.1 General Experimental Setup . . . 37
- 4.2 Experimental Setup I: Secret Model Training . . . 38
- 4.3 Experimental Setup II: Approximated Model Training . . . 38

5 Implementation **40**
- 5.1 Data Preprocessing . . . 41
- 5.2 Training and Approximation Phase . . . 41

6 Experiment I - Results **42**
- 6.1 VGG16 . . . 42
- 6.2 VGG19 . . . 43
- 6.3 AlexNet . . . 44
- 6.4 ResNet50 . . . 45
- 6.5 ResNet101 . . . 46

7 Experiment II - Results **47**
- 7.1 Agreement . . . 47
- 7.2 Transferability . . . 50

8 Discussion **51**

9 Conclusion and Future Work **52**

A Literature Review **60**
- A.1 Practical Black-Box Attacks against Machine Learning . . . 60
- A.2 How to Steal a Machine Learning Classifier with Deep Learning . . . 63
- A.3 Stealing Machine Learning Models via Predicition APIs . . . 64
- A.4 PRADA: Protecting Against DNN Model Stealing Attacks . . . 67
- A.5 Efficiently Stealing your Machine Learning Models . . . 70
- A.6 Knockkoff Nets: Stealing Functionality of Black-Box Models . . . 71

A.7 Extraction of Complex DNN Models 73
A.8 ActiveThief: Model Extraction using Active Learning and Unannotated Public Data 75

B Model Architectures **77**
B.1 AlexNet 77
B.2 VGG16 78
B.3 ResNet 79

C Secret Model Training Phase **80**
C.1 VGG16 80
C.2 VGG19 83
C.3 AlexNet 86
C.4 ResNet50 89
C.5 ResNet101 92

D GTSRB Labels **95**

Abstract

Machine learning models are part of our every-day lives. Let it be software that is responsible for handling self-driving cars or medical imaging techniques used for magnetic resonance imaging (MRI). These machine learning models can be divided into two main categories when it comes to visibility. The first one is called a "white-box" model since the access to this model's internal workings is accessible to the public. The second one is called a "black-box" model that is only accessible to the public to its limited interfaces. The overall consensus is that these models behave reliably in all situations regardless of the input that is given. In other words, these models should behave robustly. However, that is often not the case based on inputs that have been altered using adversarial crafting techniques. Furthermore, verifying the robustness of black-box models is an especially hard problem because the internal workings are not accessible.

This research concentrates on approximating black-box machine learning models by using five different active learning techniques. The approximated model may not share the same architecture as the original model but active learning assures that its behavior is similar to make assumptions about the robustness of the original black-box model. The experiments include five different deep neural networks of various complexities that have been trained on three different datasets, namely MNIST, CIFAR-10, and GTSRB. The VGG16 architecture has been used as the main architecture to approximate the other network architectures by using five different active learning strategies, namely random-, uncertainty-, K-center-, DFAL- and a combination of DFAL and K-center strategy. The approximation of the different network architectures has been evaluated using two different metrics. The first one is an overall agreement on a hold-out test set, while the second one is a transferability score that evaluates if adversarial crafting techniques based on the FGSM attack are also applicable to the original black-box model.

This research shows that it is possible to approximate black-box deep neural networks efficiently by using a combination of the DFAL- and K-center strategy but only if the original black-box model has been trained towards a high accuracy.

1 Introduction

Machine learning as a Service (MLaaS) has evolved into a profitable product for many companies like Amazon, Google, and others that offer cloud computing services. These services are often used through their APIs where users can make queries and feed data to the service to get a prediction in return and paying on a per-query basis. The underlying model is often hidden and is treated as a valuable business asset by the company due to expensive development costs. Therefore these models are perceived as black boxes by the users with the internal functionality hidden from the outside.
Users trust the internal validity of these models which means that the given predictions should be correct with certain, acceptable fault tolerance. Nevertheless, prior research as listed in section 2.4 has shown that it is possible to craft adversarial examples which get misclassified by these models using different approaches. The creation of such adversarial examples is also often referred to as an adversarial attack in the literature. A small perturbation to the original input which is almost unnoticeable to the human eye could result in a misclassification with high confidence described by [23]. Such small perturbations are sufficient to push an input over a decision boundary because inputs are often projected into higher dimensions throughout the model architecture where the dot product is calculated and the perturbation is several magnitudes larger than in the original input.

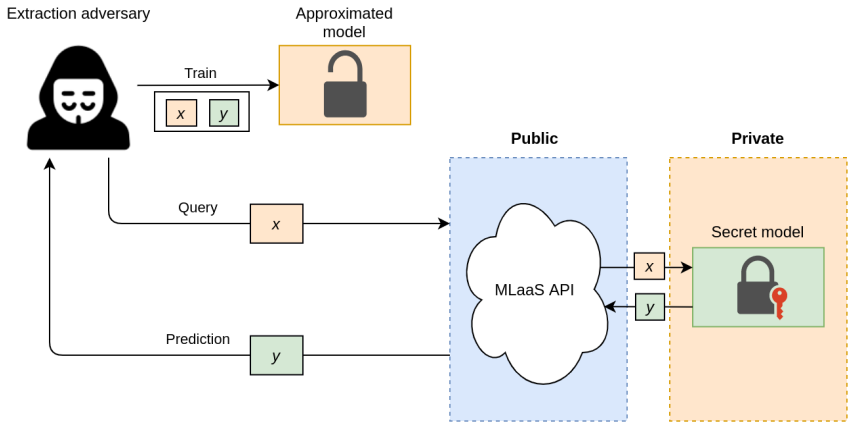


Figure 1: Approximating or extracting a secret model by using the public API of an oracle to train an approximated model.

Adversarial attacks have shown that most machine learning models are not as robust as they seem. Therefore research has been done to defend against those kinds of attacks described by [104]. Evaluating the robustness of Deep Neural Networks (DNNs) by using formal verification methods is especially difficult because they are large, non-linear, and non-convex, and verifying even simple properties about them is an NP-complete problem according to [39]. Linear programming (LP) solvers or satisfiability modulo theories (SMT) solvers come to their limits with regards to DNN verification and so far dedicated tools for this task were only able to verify small networks with a single hidden layer with only 10 to 20 hidden nodes according to [70] and [71].
One approach to tackle this problem would be to approximate the black-box model and transforming it into a white-box using a retraining approach as depicted in figure 1.

Classifier stealing describes the process of generating so-called surrogate models, approximated models, or knockoff nets as described by [62] and depicted in figure 2

taken from [62].

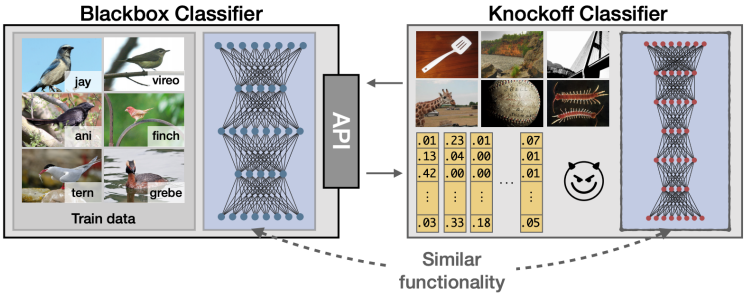


Figure 2: Knockoff Nets

While the complexity of the black-box classifier is often high, the approximated classifier does not necessarily have to share the same architecture and complexity. Most approximated classifiers often have a simpler architecture and lower complexity while sharing similar functionality. This reduction of complexity while still sharing similar functionality makes it attractive to formal verification methods to check the robustness of black-box models and is the focus of this research.

1.1 Scope and Research Questions

The intersection of adversarial attacks and classifier stealing approaches that query a prediction API to get familiar with the inner workings of a model could be used to build an efficient pre-processing step to formally verify the robustness of black-box machine learning models. Special interest in this endeavor should be directed towards the approximation of a black-box model as described by [13]. The requirements for SMT-solvers as formal verification methods for DNNs have already been investigated and motivates this research [39].

This research tries to answer the following questions:

1. Is it possible to approximate black-box machine learning models by using active learning and adversarial attacks to evaluate the robustness?
2. Do approximated machine learning models share the same decision boundaries as their black-box counterpart models?
3. Do the transferability and the agreement between the black-box and approximated model vary when the architectures differ?

1.2 Thesis Structure

Chapter 1 gives a general introduction to the reader and introduces the research questions that this project tries to answer. Chapter 2 explains the background material that may be helpful to non-experts to understand the presented research. Topics like information theory, neural networks, active learning, adversarial attacks, and classifier stealing attacks are part of this chapter. Chapter 3 explores the three different datasets that have been used for the experiments in this research, namely MNIST, CIFAR-10, and GTSRB. Different dimensionality reduction techniques were used to make assumptions about the datasets before the actual experiments were conducted. Chapter 4 covers the

methodology and the different experimental setups. Chapter 5 presents the implementations that are important to understand the experiments in more detail. Chapter 6 gives an overview of the results concerning the first experiment where all secret models have been trained on before mentioned datasets. Chapter 7 presents the results of the approximation and in how far VGG16 was able to approximate the secret models in terms of agreement and transferability. Chapter 8 discusses the results. Finally, chapter 9 ends with a conclusion about the research and how far this project was able to answer the research questions. Possible future work is also mentioned in this chapter.

2 Background

2.1 Information Theory

Information theory represents the scientific foundation for the study of quantification, storage, and communication of information. The american mathematician Claude Shannon is often described as "the father of information theory" and contributed massively to the field [82]. He is most famous for his concept of information entropy that is also called "Shannon entropy".

Entropy can be seen as the average level or expected level of "surprise" of a certain probability distribution, i.e., a variable's possible outcomes. For continuous probability distributions, entropy is defined according to equation 2.1.1, where b is the base of the logarithm. Most applications of the logarithm use a base of 2 or Euler's number e for the natural logarithm. The corresponding units of entropy are bits for $b = 2$ and nats for $b = e$.

$$H[x] = -\int p(x) \log_b p(x)\ dx \tag{2.1.1}$$

The continuous case can easily be translated to the case for discrete probability distributions as shown in equation 2.1.2.

$$H[x] = -\sum_x p(x) \log_b p(x) \tag{2.1.2}$$

However, the following equations use the continuous case for a more unified explanation.

Whereas entropy describes the "average" or expected level of surprise, the information content (also called "surprisal") is one of the entropy's building blocks and is defined in equation 2.1.3. It describes the information content associated with an event.

$$I(x) = h(x) = -\log_b p(x) \tag{2.1.3}$$

The information content is important to calculate the storage needs for encoding problems. Consider a series of coin tosses like "HTHH", where "H" stands for heads and "T" stands for tails. Given that we tossed the coin four times and each coin toss has two possible outcomes, the total amount of possible outcomes would be 16 since $2^4 = 16$. This series could also be represented in binary notation as 0100. To answer the question of how many bits we need to represent all 16 states, we can take the reciprocal of the probability of the particular event "HTHH". Given that the coin is fair and the event is one out of 16 possible outcomes, we can use $\frac{1}{16}$ as the argument for the information content with a logarithm base of 2. The total amount of bits we need to encode all 16 states is then answered in equation 2.1.4.

$$h(\frac{1}{16}) = 4 \tag{2.1.4}$$

To adjust the previous example to the entropy case, assume that we have a biased coin where the probability of heads is $\frac{3}{4}$ and tails is $\frac{1}{4}$. Encoding these probabilities into a vector of the form $v = [0.75, 0.25]$ and computing the entropy of this simple probability distribution is shown in equation 2.1.5.

$$H(v) = 0.8113 \tag{2.1.5}$$

However, the highest entropy is always given when the probability distribution is uniform, i.e., every event has the same probability to occur. A fair coin would have

the probability distribution of $\frac{1}{2}$ for heads and $\frac{1}{2}$ for tails. Encoding these probabilities again into a vector of the form $v = [0.5, 0.5]$ and computing the entropy would result in a value of 1.

The information content and entropy are useful to calculate how "surprising" a probability distribution is. However, these tools are restricted to only one distribution. If we wanted to compare two probability distributions to each other, we had to rely on a measurement called "Kullback-Leibler divergence".

The Kullback-Leibler divergence is also known as the KL divergence or relative entropy [44] between distribution $p(x)$ and $q(x)$. It is not a symmetrical quantity, i.e., $KL(p||q) \not\equiv KL(q||p)$. The KL divergence can be constructed from the entropy defintion from equation 2.1.1 as follows in equation 2.1.7.

$$KL(p||q) = -\int p(x) \log_b q(x)\ dx - \left(-\int p(x) \log_b p(x)\ dx \right) \tag{2.1.6}$$

$$= -\int p(x) \log_b \left\{ \frac{q(x)}{p(x)} \right\} dx \tag{2.1.7}$$

KL divergence is often used indirectly by supervised machine learning models as a so-called "loss function" or "cost function", where it measures the performance of a model. The loss function measures in how far the model's predicted probability diverges from the actual label probability.

A common loss function is the so-called "cross-entropy" and can be constructed by using the KL divergence as shown in equation 2.1.8.

$$H(p, q) = H(P) + KL(p||q) \tag{2.1.8}$$

As equation 2.1.8 shows, cross-entropy is the sum of the entropy and the KL-divergence but can further be simplified as follows in equation 2.1.9.

$$H(p, q) = -\int p(x) \log_b q(x)\ dx \tag{2.1.9}$$

Neural networks heavily rely on loss functions like cross-entropy and are explained in section 2.2.

2.2 Neural Networks

Frank Rosenblatt invented the first artificial neural network (ANN) in 1958 [75] that was loosely inspired by its biological counterpart. As shown in figure 3, the single biological neuron[1] takes its signals from the dendrites which are then further processed by the nucleus. If the signal's action potential is high enough to surpass a certain threshold then the signal gets forwarded to the next neuron over the axon and the axon terminal points. The axon terminal points can then be connected to the dendrites of other neurons to build a neural network.

[1] `https://en.wikipedia.org/wiki/Axon`

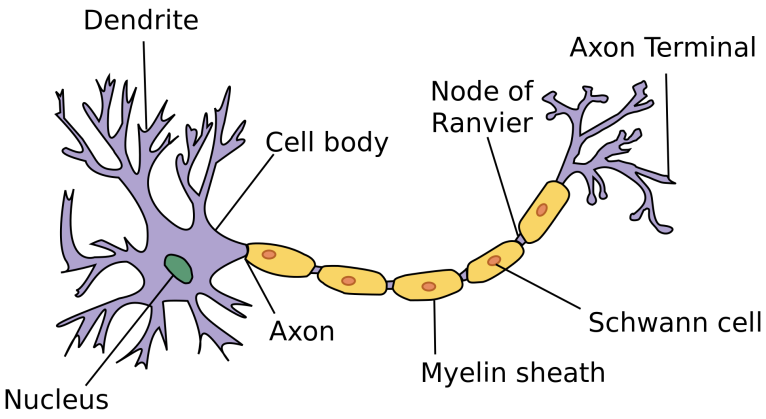


Figure 3: A single biological neuron

Although the first ANN invented by Rosenblatt took inspiration from a network of many biological neurons, his invention was named "perceptron". A perceptron is a simple network with one input and one output layer that can contain a variable amount of artificial neurons. Given that this architecture was rather limited in its classification capabilities, it was later on extended with so-called "hidden layers" between the input and output layer. This extended architecture was named "multilayer perceptron" and is a special kind of feedforward neural network (FFNN) as shown in figure 4.
As the name suggests, the input layer takes the inputs of a certain data sample that has been preprocessed into numerical values. These numerical values are then forwarded through the hidden layers and finally aggregate towards the end in the output layer. The output layer can be used to make predictions or classifications.

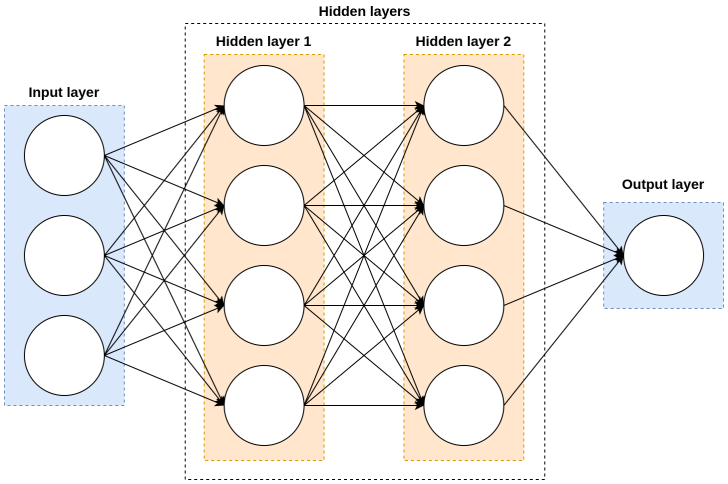


Figure 4: Feedforward neural network (FFNN) architecture

This simple idea is extended by various components that are important to make the artificial neural network work. As depicted in figure 5, the artificial neuron takes the incoming connections from other neurons $x_1 - x_n$ that are associated with certain weights $w_1 - w_n$. The weights determine how strong the connections are. The incoming weighted connections are used to compute a weighted sum added to a bias term b. Just as its biological counterpart, the weighted sum gets only forwarded when a certain

threshold is met. To mimic the biological counterpart of an electrical threshold, a so-called "activation function" is used that takes the bias term and the weighted sum as its input. The activation function then determines the value that should get propagated to the next neurons.

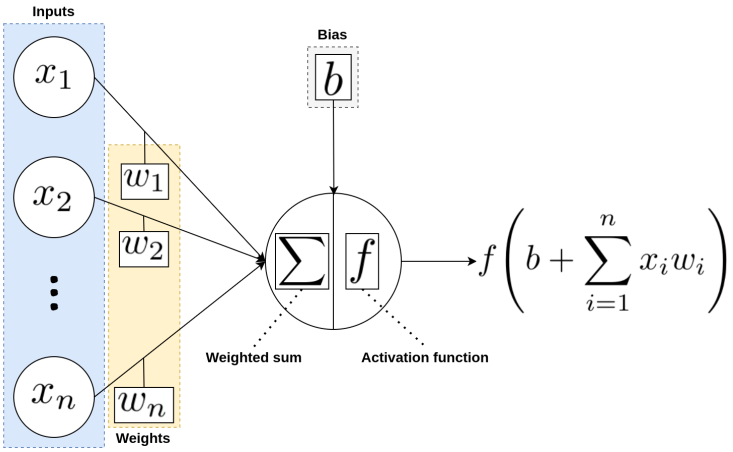


Figure 5: An example of an artificial neuron with inputs x_1 to x_n and their associated weights w_1 to w_n. A bias b and an activation function f is applied to the weighted sum of inputs

Activation functions like Sigmoid or rectified linear unit (ReLU) [101] can be used as the concrete function to also introduce non-linearity into the network. It has also been shown that newer activation functions like ReLU help to prevent problems like the vanishing gradient problem [30].

The actual learning phase of an ANN involves a technique called backpropagation [100] and an optimization function such as gradient descent [76].

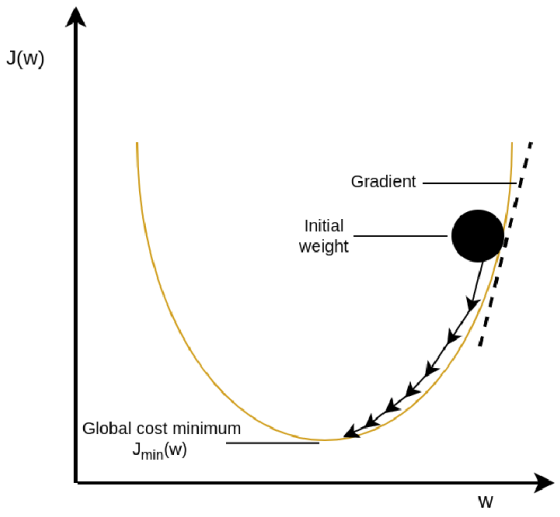


Figure 6: Simplified version of gradient descent

As soon as the signals reach the output neurons, the predicted value gets compared

to the actual label of the sample by using a loss function like cross-entropy which is further explained in section 2.1. The derivative of the loss function is then used by an optimization technique like gradient descent to iteratively find the global cost or loss minimum as depicted in figure 6. The value of the weights is iteratively adjusted concerning the gradient to find the global loss minimum. The computed error between the prediction and the label is then backpropagated through the network by applying the chain rule to adjust the previously mentioned weights concerning the error. Several iterations of this procedure lead to a network architecture that can learn a function mapping between inputs and outputs.

It has been shown that such FFNN architectures are universal function approximators even when only one hidden layer is used [33]. Although most applications of FFNN involve tasks like the recognition of handwritten characters [20] or speech recognition [27], their performance seems less than ideal when it comes to more sophisticated image recognition tasks.

For this purpose, convolutional neural networks (CNNs) seem to achieve higher performance and are introduced in section 2.2.1.

2.2.1 Convolutional Neural Networks

Convolutional neural networks (CNNs) are a special type of ANN which have been popularized by Yann LeCun in 1989 with his LeNet architecture [48]. LeNet or lenet-5 is a CNN and therefore also a feedforward neural network (FFNN) which has achieved outstanding results in various computer vision tasks such as the recognition of handwritten characters [49]. CNNs differ from simple FFNN architectures by their capability to learn spatial hierarchies of features by using different building blocks such as convolutional, pooling, and fully connected layers.

A convolutional layer as depicted in figure 7 makes use of various so-called "filters" or sometimes called "kernels". The output of the filter operation applied to an input image is forwarded to an activation function to obtain feature maps.

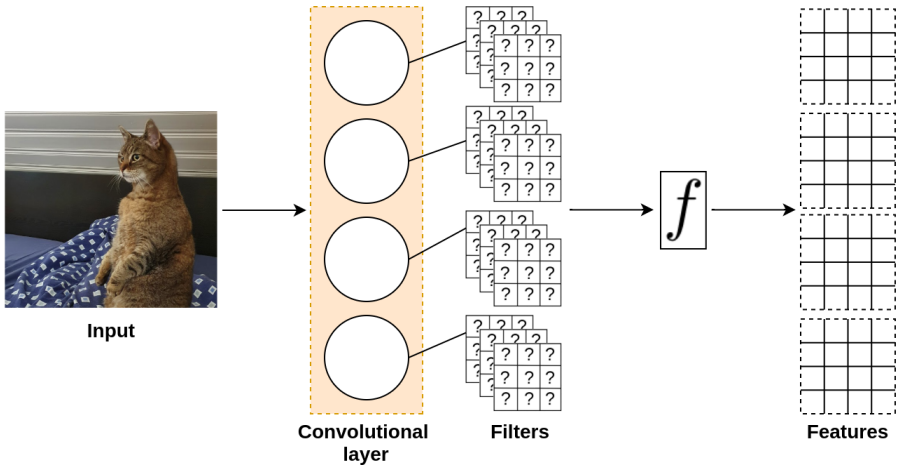


Figure 7: Convolutional layer consisting of different filters

This convolution operation is applied by dividing the raw image input into several pixel value matrices as shown in figure 8. A filter is then applied to a subset of the matrix

to obtain the dot product which results in a convolved feature matrix. The filter slides over the image in a sliding window fashion with a certain step or off-set size that is also referred to as "stride". The idea of applying filters or kernels to images is based on an experiment by [35] in 1959 where a microelectrode was inserted into the primary visual cortex of an anesthetized cat. They found out that some neurons only fired when the visible, projected pattern aligned along a certain angle while other neurons responded only to another angle. Filters are also often used in photo editing software for blurring purposes or edge detection.

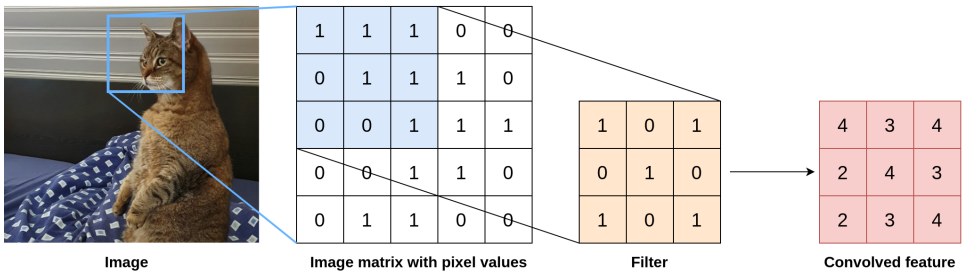


Figure 8: Convolution

One limitation of the resulting feature maps is that they record precise feature positions which makes them vulnerable to image changes like rotation or shifting. A so-called "pooling" operation after the convolutional layer addresses this issue by downsampling. Different pooling operations exist such as average-pooling or max-pooling. Max-pooling is depicted in figure 9 and shows the downsampling operation by only taking the highest value in each quadrant of the matrix.

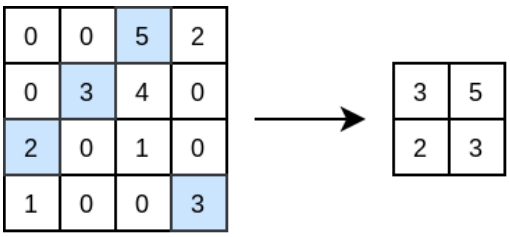


Figure 9: Max-pooling layer with maximum values in each quadrant marked by a blue color

Average-pooling operates similarly but takes the average of each quadrant.
One problem of applying several filters inside a CNN network is because filters reduce the original size of the input. The deeper the network the higher the downsampling rate of the convolutional layer. This may result in an image size that is smaller than the filter size. One solution to this problem is the application of a padding operation such as zero-padding depicted in figure 10.

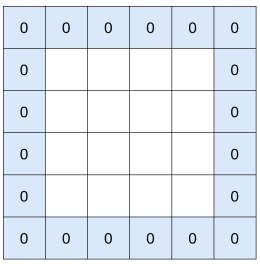


Figure 10: Zero-padding: The original pixel matrix is expanded by a border of one pixel with the value of zero

Zero-padding ensures that the original input size of the image is preserved so that all filter operations can be applied inside the network.

One of the last layers of a CNN architecture is the fully connected layer that takes the features of the convolution and pooling layers to classify the input. This fully connected layer can be seen as an FFNN that goes through the same backpropagation process as the neural network described in section 2.2.

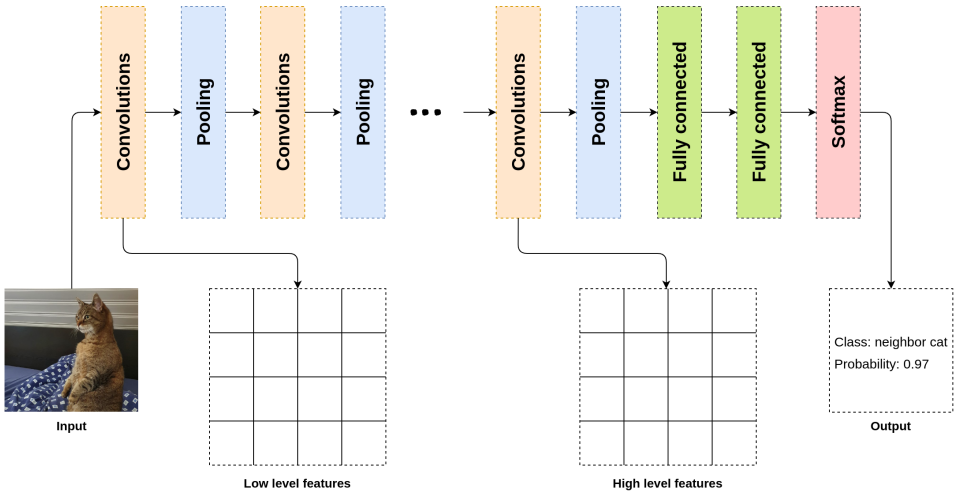


Figure 11: High level overview of an CNN architecture

Common CNN architectures often follow the design depicted in figure 11. Several convolutions and pooling layers are stacked after each other followed by fully connected layers and a softmax function at the end. The softmax function [60] is another type of activation function that ensures that all outputs of the last fully connected layer are transferred into a proper probability distribution. The output of the softmax function is a probability distribution over all class labels between 0 and 1 where the sum of the probabilities is equal to 1.

One special CNN architecture was designed by Alex Krizhevsky with Ilya Sutskever and Geoffrey Hinton [43] in 2012 and was named "AlexNet". AlexNet won the ImageNet Large Scale Visual Recognition Challenge in 2012 with a top-5 error of 15.2%. The reason for its high performance was due to its architectural depth as shown in figure 12. Another technique that also supported this high performance is the introduction of a

batch normalization layer [36].

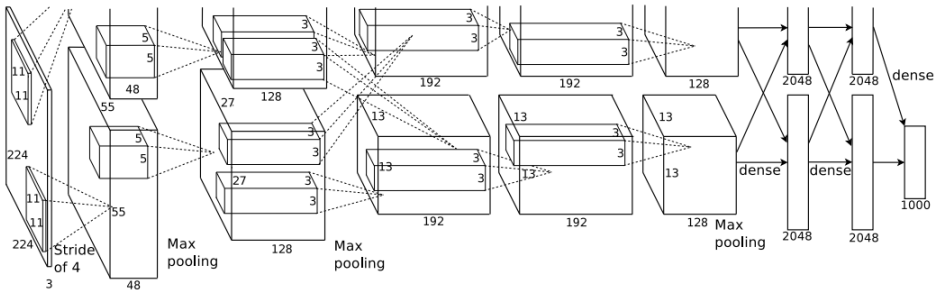


Figure 12: AlexNet architecture diagram that was taken from [43]

The architecture in figure 12 shows that it includes the same CNN components that were explained before where the word "dense" describes the fully connected layer.

The successor of AlexNet is called VGG16 and is a CNN architecture proposed by K. Simonyan and A. Zisserman [86] in 2014. VGG is an abbreviation for visual geometry group. The model achieved a 92.7% top-5 test accuracy on ImageNet [16].

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224×224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

Figure 13: VGG CNN configuration where the depth increases from the left (A) to the right (E). The table was taken from [86]

It improved the AlexNet architecture by replacing large kernel-sized filters in the first and second convolutional layers with multiple 3x3 kernel-sized filters. The overall architecture of VGG is depicted in figure 13 that was taken from [86]. The digits behind "VGG" specify the number of weight layers inside the CNN architecture. While VGG16 has 16 weight layers, VGG19 includes 19 weight layers.

The main problem with CNN architectures is the increased training time the more complex or deeper the architecture becomes. This problem has been addressed by a new CNN architecture called residual neural networks and is further explained in section 2.2.2.

2.2.2 Residual Neural Networks

Residual neural networks (ResNets) solve some of the issues of CNN architectures, e.g., the degradation problem. The degradation problem states that CNN architectures do not necessarily improve their performance by adding layers but that the addition of layers may lead to worse performance than a shallow network as depicted in figure 14. It is shown that a 56-layer deep network has a higher training error than a 20-layer network and that this performance difference also holds when inspecting the test error.

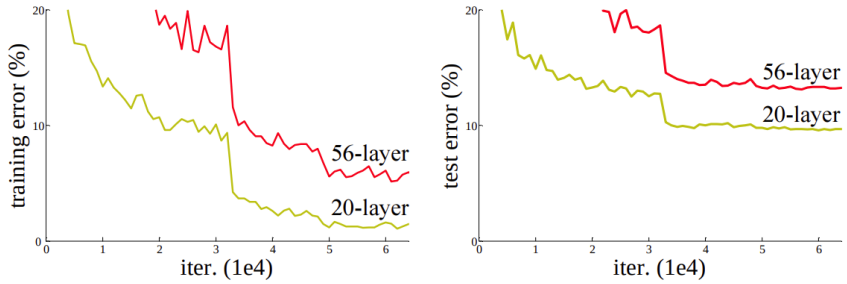


Figure 14: Training error (left) and test error (right) on CIFAR-10. Depicting neural networks with a layer depth of 20 and 56. The shallower network has a lower training error than the deeper network. The diagram was taken from [25]

A possible explanation for this problem lies in the very nature of the CNN architecture itself by trying to build more high-level hierarchical features the deeper the network becomes. The addition of layers may lead to a situation where no more additional features can be learned and the feature maps do not change anymore. At that point, the CNN architecture tries to learn a so-called identity mapping of feature maps as depicted in figure 15. The output of a pooling- and convolutional-layer is a set of feature maps that do not change after a certain point.

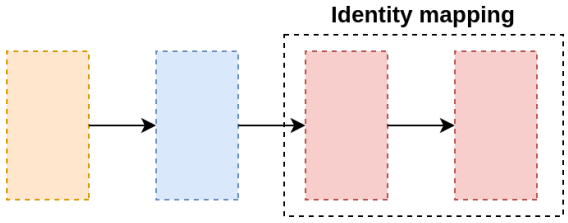


Figure 15: The deeper the network becomes, the higher the probability that the last layers are repeating their feature maps.

The overall assumption would be that this identity mapping is a simple operation that only has to learn weights that ensure that the input is equal to the output as depicted in figure 16. This simple mapping is highlighted by red arrows in figure 16 but neural networks often struggle to learn this mapping.

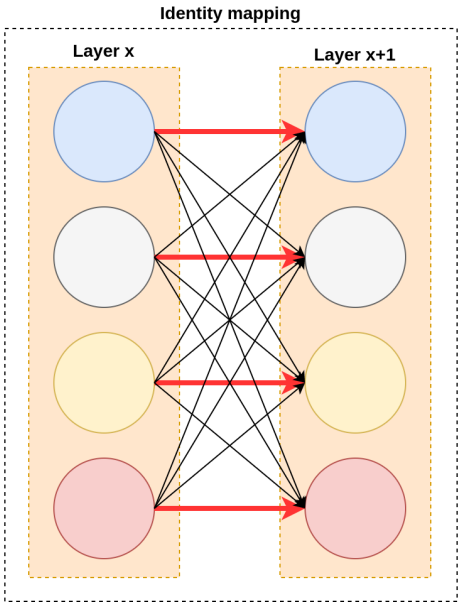


Figure 16: The identity mapping of one layer to another should be a simple task but the statistical nature of neural networks might make this difficult.

This problem has been approached by [25] with the invention of ResNets and taking inspiration from biological pyramidal cells in the cerebral cortex. This is done by introducing so-called "skip connections" or "residual blocks" that allow the flow of information from one layer to another by skipping intermediate layers as depicted in figure 17. ResNets explicitly reformulate the layers as learning residual functions regarding the layer inputs [25].

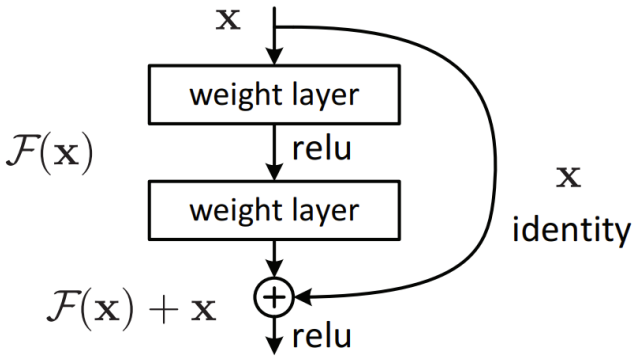


Figure 17: Residual learning: a building block. The diagram was taken from [25]

The overall idea is that instead of learning a desired underlying mapping $\mathcal{H}(x)$ directly where x is the input vector, the authors let the stacked layers fit another mapping of $\mathcal{F}(x) := \mathcal{H}(x) - x$ which is called the "residual function". The initial mapping can then be rewritten as $\mathcal{F}(x) + x$. To ensure that the dimensions of $\mathbf{x}$ and $\mathcal{F}$ stay equal through the convolution operations, a linear projection W_s is applied as defined in equation 2.2.1:

$$\mathbf{y} = \mathcal{F}(\mathbf{x}, \{W_i\}) + W_s\mathbf{x} \tag{2.2.1}$$

The so-called "skip connections" as depicted in figure 17 therefore pass the input vector $\mathbf{x}$ to the next layer as an output. The network only has to learn the residual $\mathcal{F}(\mathbf{x})$ which defines how far the input differs from the output. This makes learning the identity mapping easier because both the input and output are known to both layers.

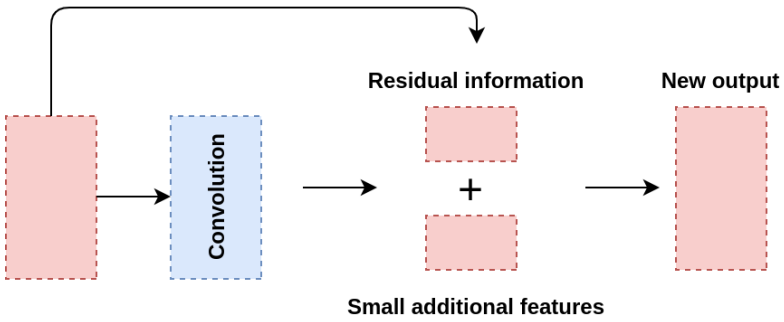


Figure 18: Simplified residual learning procedure

A simplified version of this residual learning procedure is depicted in figure 18 where the blue block can be a convolution operation and the red blocks are inputs and outputs.

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

Figure 19: ResNet architectures with 18-, 34-, 50-, 101- and 152-layer depth taken from [25]

Figure 19 shows different configurations of the ResNet architecture by using different amounts of layers. ResNet50 would mean that the ResNet is 50 layers deep. An ensemble of different ResNet architectures has achieved a 3.57% top-5 error on the ImageNet test set in 2015 which led to 1st place in the ILSVRC 2015 classification competition [25].

2.3 Active Learning

The Active Learning Literature Survey [80] gives a summarization of the most common active learning techniques. The author describes three main active learning scenarios, namely membership query synthesis, stream-based selective sampling, and pool-based active learning.

Active learning describes the process that a machine learning model actively takes part in the selection of its training data. The main hypothesis states that the learning

algorithm performs better with fewer training cycles if it is allowed to choose its training data. This idea differs from common supervised learning setups that could be seen as "passive" learning where the training set is fixed. Active learning is commonly used for machine learning problems where unlabeled data is available but the labeling process is expensive. The labeling bottleneck can be mitigated by asking a so-called oracle (e.g., a human annotator or another machine learning model) to label instances whose origin could be an unlabeled data set or a de novo synthesis of data. De novo is a Latin phrase and can be translated to "from scratch" or "from the beginning", i.e., data generated on the fly from scratch.

As mentioned before, there are different scenarios in which the learning algorithm can actively select queries to reduce the training effort. The three scenarios which are commonly used in the active learning literature are depicted in figure 20.

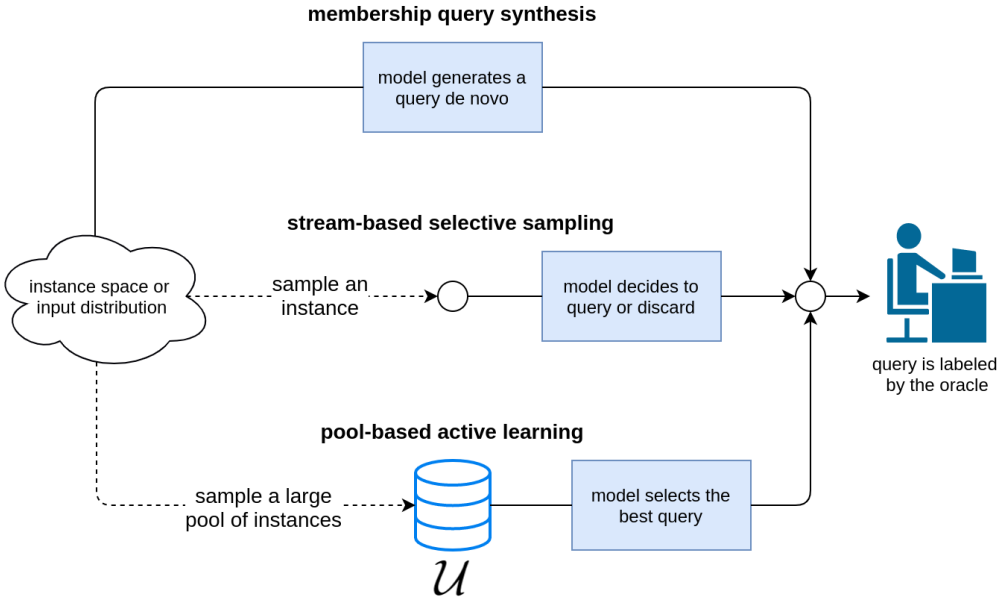


Figure 20: Diagram illustrating the three main active learning scenarios.

Membership Query Synthesis

The first scenario is called "membership query synthesis" and is one of the first active learning scenarios using membership queries [2]. The learning algorithm can request any unlabeled data instance out of the input space that is usually generated by the learning algorithm on the fly, i.e., de novo instead of sampling from a natural distribution. Although query synthesis applies to many problems, it can be hard for an oracle to label these instances properly given that their origin is not a natural distribution.

Stream-Based Selective Sampling

In contrast to query synthesis, stream-based selective sampling is using instances from an actual distribution and then decides whether or not to request their labels [11]. It is assumed that sampling unlabeled instances is an inexpensive operation and most computational effort is due to the decision process if the label for that instance should be requested. Given that unlabeled instances are usually drawn individually in a sequential way, this approach is sometimes referred to as stream-based or sequential

active learning. Different techniques exist to decide if the label for the instance should be requested. An instance could be evaluated based on an "informativeness measure" where more informative instances have a higher probability to be queried [15] or using a query strategy. Other approaches try to explore the sample space that still yields a high uncertainty to the learning algorithm [11]. Some authors use the term "selective sampling" to refer to pool-based active learning [93, 59] because both scenarios select instances from a real data distribution. However, pool-based active learning is evaluating and ranking the whole sample collection before selecting the best query while stream-based active learning is evaluating the data sequentially and also makes query decisions on an individual basis.

Pool-Based Active Learning

Pool-based active learning utilizes a small labeled training set $\mathcal{L}$ and a large pool of unlabeled data $\mathcal{U}$ as depicted in figure 21 and described by [53]. The learning algorithm begins with a small number of instances from $\mathcal{L}$ as training data. The learning algorithm then samples queries from $\mathcal{U}$ based on an informative measure or another query strategy for labeling purposes executed by an oracle. The labeled instances from these queries are then added to $\mathcal{L}$ to further train the learning algorithm in a supervised fashion. This cycle continues until the convergence of a certain metric is satisfied, e.g., accuracy or loss.

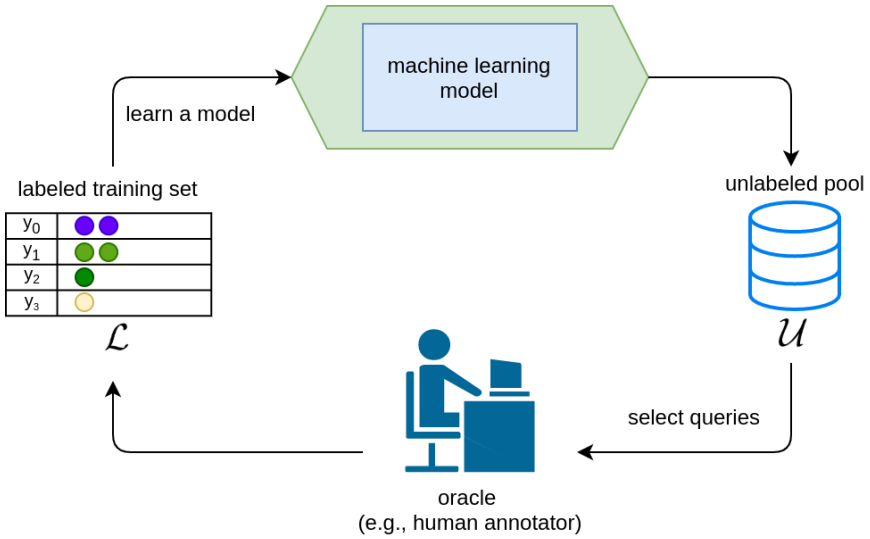


Figure 21: The pool-based active learning cycle.

This scenario applies to many real-world problems where large collections of unlabeled data are easy to acquire.

Query Strategy Frameworks

Before mentioned scenarios can be combined with different query strategy frameworks,i.e., query strategies. A query strategy evaluates in how far an unlabeled instance is informative or generally useful for the reduction of training effort.

Uncertainty sampling is the simplest and most common query strategy where the learning algorithm queries instances which it is least certain about how to label [53].

Uncertainty can be expressed by different measures but the most general uncertainty sampling strategy uses entropy [82] and is further explained in section 2.1:

$$H[x] = -\sum_{x} p(x) \log_b p(x) \tag{2.3.1}$$

where x can be a probability vector of predicted class labels.

Another closely related uncertainty sampling technique involves querying instances that the learning algorithm is least confident (LC) about:

$$LC = \underset{x}{\operatorname{argmin}}\, P(y|x)$$

where $LC = \operatorname{argmax}_y P(y|x)$ is the most likely class labeling.

Both approaches try to identify instances that lie close to the decision boundary and should therefore contribute to the informativeness factor. One problem with identifying instances that lie close to the decision boundary is due to clustering instances around that specific instance, i.e., the instance density in that region. Figure 22 illustrates this problem.

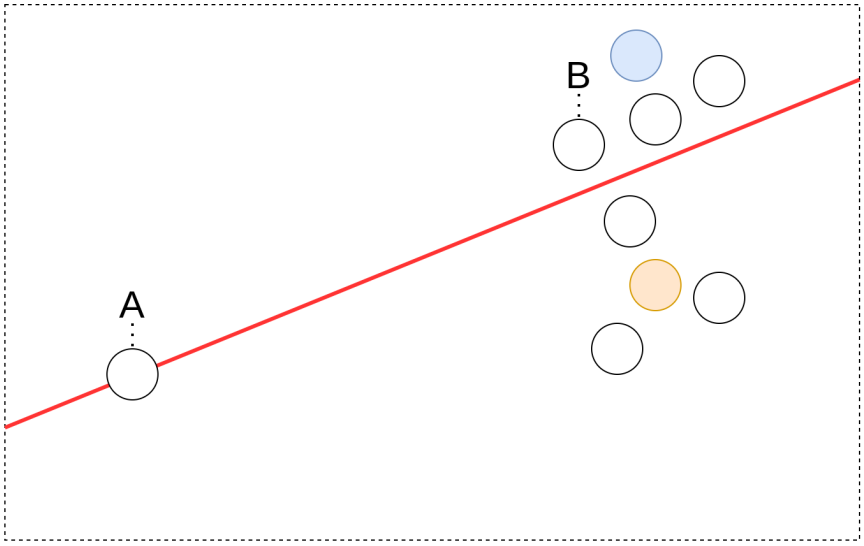


Figure 22: Uncertainty sampling can be a poor strategy when a sample lies close to a decision boundary but is not part of a bigger cluster. Colored circles represent labeled instances in $\mathcal{L}$ and white circles represent unlabeled instances in $\mathcal{U}$. Given that A lies directly on the decision boundary represented as the red line, the uncertainty strategy would select A. However, B would be a better choice because it is part of a cluster and more informative.

Another method to explore the sample space is based on the work by [79] for CNN architectures where active learning is defined as a core-set selection problem. The underlying algorithm is based on the greedy k-center algorithm [29]. This means that a set of training samples is selected as a subset that is competitive concerning the remaining samples. Therefore, the aim is to select a subset of samples to train the model that performs as closely as possible as a model that was trained on the whole dataset. The initial seed samples that are mostly selected on a random basis are used

as cluster centroids. In each iteration, the most distant samples from all centroids in terms of euclidean distance are selected next which leads to an active learning strategy that can be used as a diversity measurement. As depicted in figure 23, the blue circles represent the initial cluster centroids while the red circles represent the potential next training samples.

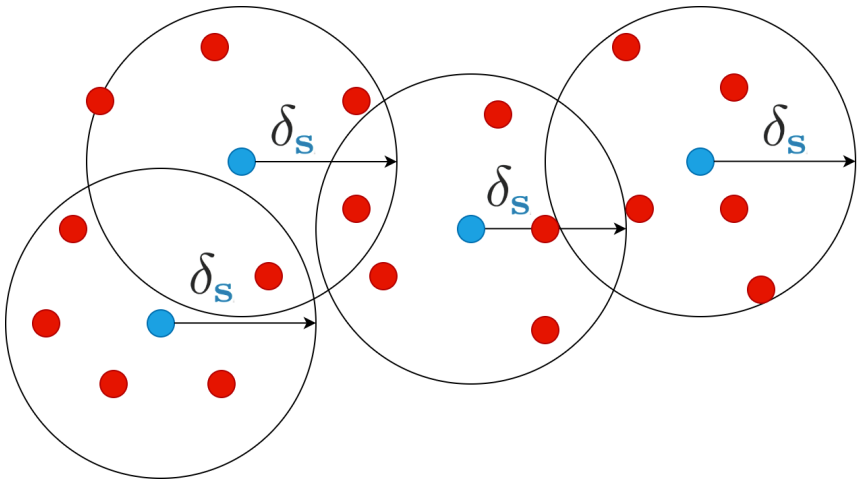


Figure 23: The core-set selection problem where blue circles represent the intitial cluster centroids and red circles represent the potential next trainig samples. δ represents a hyperparameter of the centroids maximum radius.

As most modern probabilistic machine learning models are using gradient-based optimization techniques that often require training cycles in batches, active learning approaches this case by so-called batch-mode active learning where instances are queried in groups. The main challenge is due to the construction of an optimal query batch $\mathcal{Q}$ that incorporates instances that have minimal overlap of information content. Density-based approaches that query cluster centroids that lie close to the decision boundary have been successfully used for SVMs [102]. Other approaches based on the Fisher information have been successfully applied to binary logistic regression and ensure that batches are both diverse and informative [31, 32].

Related research areas to active learning are model compression and model parroting. Model compression describes the case when the knowledge of a more complicated model gets transferred to a simpler model that often holds another set of properties. The idea of model compression has been used to transfer knowledge from computationally expensive models to more efficient model classes [9]. While active learning tries to reduce the number of training samples, model compression tries to reduce the complexity of the model.
Model parroting on the other hand is closely related to active learning and describes the case when the oracle is another machine learning model. This "oracle model" is queried by the "parrot model" to label any unlabeled data and to synthesize instances. The newly acquired training instances are then used to train the parrot model to mimic the behavior of the oracle model.

2.4 Adversarial Attacks

Deep neural networks such as CNNs and ResNets as described in section 2.2.1 and 2.2.2 respectively have shown to be effective function approximators when it comes to images.

However, research has shown that these networks are prone to small pixel perturbations which are almost unnoticeable to the human eye but can lead to misclassifications with high confidence scores [91]. These altered images are often referred to as "adversarial examples" or "adversarial attacks".

One popular choice to generate these adversarial examples is based on an algorithm called "Fast Gradient Sign Method" (FGSM) [23]. A demonstration of the algorithm is shown in figure 24 where FGSM is applied to GoogLeNet on ImageNet.

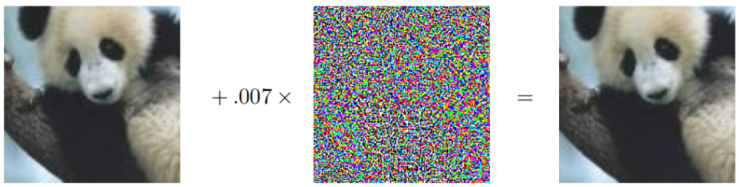


Figure 24: Fast Gradient Sign Method (FGSM) applied to GoogLeNet on ImageNet with ϵ of 0.007. The image was taken from [23].

The main idea behind FGSM is based on a small perturbation with magnitude ϵ that is added to the original image. By linearizing the cost function $J(\theta, x, y)$ of the neural network around the value of θ, the authors can obtain an optimal max-norm constrained perturbation as shown in equation 2.4.1.

$$\eta = \epsilon \text{sign}(\nabla_x J(\theta, x, y)) \tag{2.4.1}$$

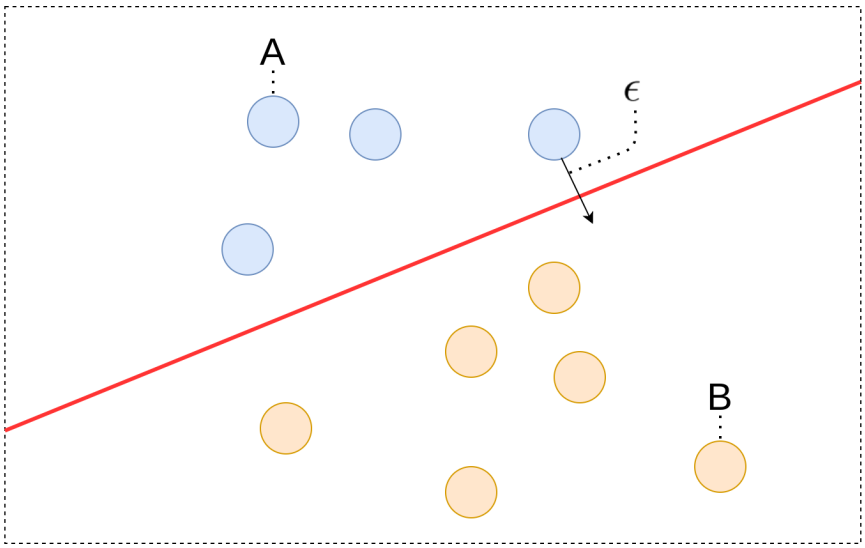


Figure 25: Pushing data point A over the decision boundary to be falsely classified as B with a perturbation of magnitude ϵ.

While x represents the input to the model with parameters θ, y represents the corresponding labels to x. The authors have shown that a shallow softmax classifier can have an error rate of 99.9% with an average confidence of 79.3% on the MNIST test set while using $\epsilon = 0.25$. While FGSM is the most known adversarial crafting technique, it

is not the only one. Iterative Fast Gradient Sign Method (I-FGSM) [46] and Momentum Iterative Fast Gradient Sign Method (MI-FGSM) [17] are also popular choices and are explained in section A.4. Although adversarial crafting techniques may differ slightly concerning their implementation, the goal of pushing one data point over the decision boundary to be falsely classified as another class is the same and is depicted in figure 25.

Furthermore, adversarial attacks have been used to evaluate the robustness of neural networks while DeepFool is the most prominent approach [58]. The authors argue that calculating the decision boundary of a neural network directly is an intractable problem but that adversarial attacks can be used as an approximation technique to determine how great the perturbation has to be to pass the decision boundary. According to [58], the authors define an adversarial perturbation as the minimal perturbation r that is necessary to change the predicted label $\hat{k}(x)$:

$$\Delta(x, \hat{k}) := \min_r ||r||_2 \text{ subject to } \hat{k}(x + r) \neq \hat{k}(x) \tag{2.4.2}$$

In equation 2.4.2, x is the input image and $\hat{k}(x)$ is the predicted label of the classifier. The robustness is generally described as $\Delta(x, \hat{k})$ of $\hat{k}$ at input image x and further defined in equation 2.4.3.

$$\rho_{adv}(\hat{k}) = \mathbb{E}_x \frac{\Delta(x; \hat{k})}{||x||_2} \tag{2.4.3}$$

$\mathbb{E}_x$ represents the expectation over the distribution of provided data samples.

2.5 Classifier Stealing

Stealing a classifier is an expression that can be seen as the equivalent to approximating a model f where the approximation $\hat{f}$ has to match the original as closely as possible. According to [62] the act of stealing a classifier has to be divided into different categories where one could steal parameters [95], hyperparameters [99], architecture [61], information about the training data [85] or the decision boundaries of the black-box model [64]. The approach by [62] concentrates on duplicating the behavior of the black-box model which is also referred to as a "victim" model. The general setup of these extraction attacks is depicted in figure 26.

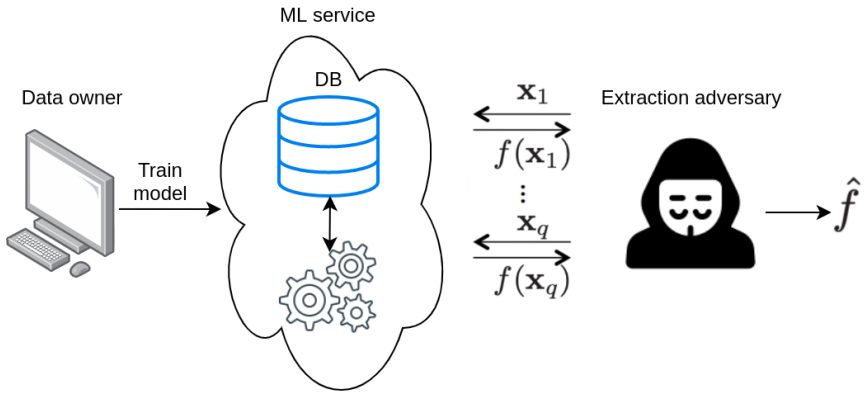


Figure 26: **Diagram of ML model extraction attacks.** A data owner has a model f trained on its data and allows others to make prediction queries. An adversary uses q prediction queries to extract an $\hat{f} \approx f$

A so-called "data owner" is training her model and deploys the trained model to a publicly available service. This service can receive queries in the form of data samples and returns classifications for these data samples. A so-called "extraction adversary" can try to query the deployed service to receive classifications or labels for the queried samples. The service acts as a so-called "oracle" that labels input data. The extraction adversary can then use these input-label tuples to train his approximation $\hat{f}$ of the original model and tries to use as few queries as possible [95]. This approach is also known as the "retraining approach" [74].
It has been shown that the retraining strategy can approximate models even when the approximation is less complex than the target model [62].

Although the internal workings of these deployed classifiers are mostly hidden from the public and are called "black boxes" in this case, there also exist various techniques to approximate models when prior knowledge is available [74]. A detailed literature review about classifier stealing techniques is available in the appendix in section A.

3 Data

Three different datasets were used to train the secret models. The MNIST dataset is a collection of handwritten digits and is further explained in section 3.1.
The second dataset called CIFAR-10 is explained in section 3.2 and consists of colored images of 10 different classes. The German Traffic Sign Recognition Benchmark (GTSRB) dataset is the third dataset and consists of 43 different classes of RGB images further explained in section 3.3. All three datasets have been chosen for training the baseline models since they were used in prior work presented in section 2. Although both datasets, MNIST and CIFAR-10 are publicly available, both datasets were loaded by using TensorFlow's dataset API version 2.3.0. GTSRB was loaded from its official website in raw image format.

Two commonly used dimensionality reduction techniques have been applied to 10% of the training set of MNIST, CIFAR-10, and GTSRB using a stratified sampling technique. The first technique is called Principal Component Analysis (PCA) [67, 34] and is an orthogonal linear transformation that tries to project the given data to a new coordinate system by using the greatest variance for the first principal component, i.e., the first coordinate.

While PCA is a linear non-probabilistic dimensionality reduction technique, t-Distributed Stochastic Neighbor Embedding (t-SNE) [56] is a non-linear probabilistic technique that assigns higher probabilities to similar data point pairs and tries to minimize the Kullback–Leibler divergence with regards to the locations of the points and their distributions. Both PCA and t-SNE reduced the data to only two components to enable an easier visualization in 2D space.

3.1 MNIST

Training samples	Test samples	Dimension	Channels
60,0000	10,000	28x28	1 (gray-scale)

Table 1: MNIST dataset.

MNIST[49] is a dataset of handwritten digits represented as single-channel grayscale images. It is publicly available[2] and was loaded by using TensorFlow's Keras API[3] in this project.
MNIST contains a training and test set of 60,000 and 10,000 examples, respectively. Each image is 28x28 pixels in size where the digit was centered by computing the center of mass of the pixels. A summary of before mentioned dataset characteristics is shown in table 1.

3.1.1 Exploratory Data Analysis

This section contains different data analysis techniques to explore the MNIST dataset a bit further.
A randomly chosen subset of examples from both the training and test set is shown in figure 27.

[2] `http://yann.lecun.com/exdb/mnist/`
[3] `https://www.tensorflow.org/api_docs/python/tf/keras/datasets/mnist`

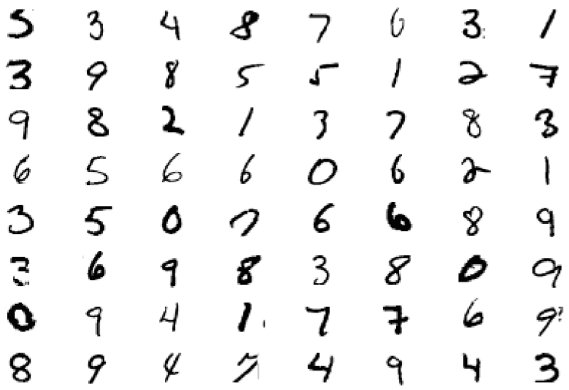
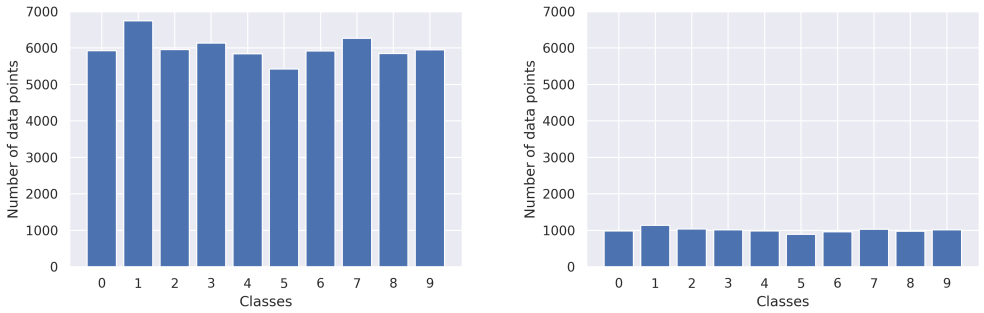


Figure 27: MNIST samples

Both the training and test set follow approximately the same class distribution. Although the distribution is not strictly uniform, the class distribution as depicted in figure 28 can be seen as balanced with minor fluctuations.

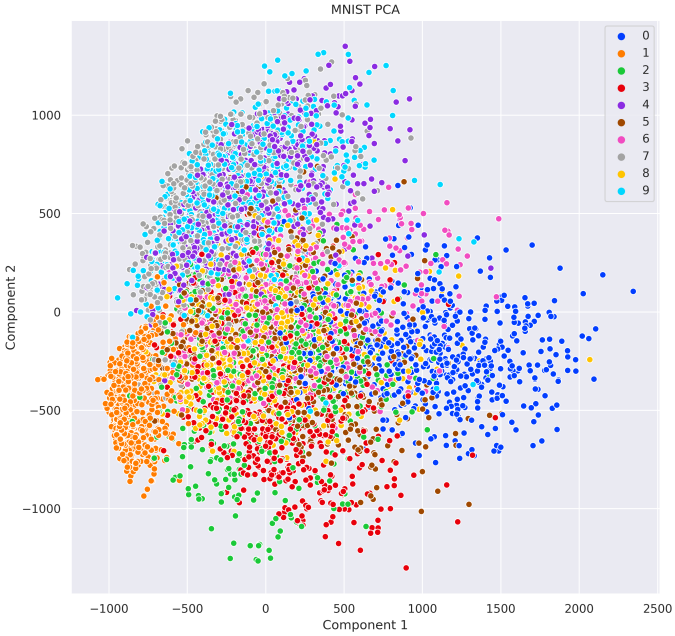


(a) Training set

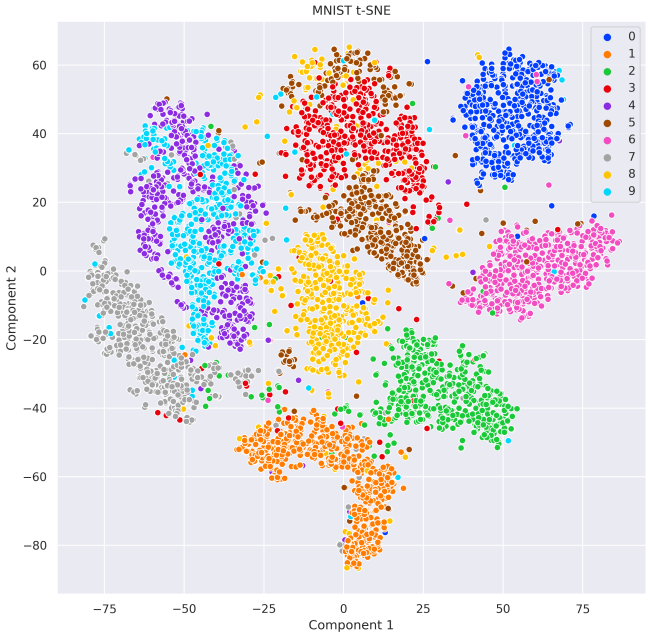
(b) Test set

Figure 28: MNIST class distribution

PCA reveals that the MNIST data is not strictly linearly separable but can be partly visualized into clear clusters, e.g., digit "1" in the lower-left corner. On the other hand, t-SNE can identify clusters for almost every class and shows that the data is non-linearly separable. Although there is some overlap between classes, e.g., "9" with "4" and "5" with "3", clear decision boundaries seem to exist. Figure 29 depicts both dimensionality reduction techniques applied to the MNIST subset.



(a) PCA



(b) t-SNE

Figure 29: MNIST - Visualization of reduced data

3.2 CIFAR-10

Training samples	Test samples	Dimension	Channels
50,0000	10,000	32x32	3 (RGB)

Table 2: CIFAR-10 dataset.

Similar to the previously presented MNIST dataset, CIFAR-10 [42] is publicly available[4], contains small images of 10 different classes, and was loaded by using TensorFlow's Keras API[5]. However, CIFAR-10 consists of color images that are 32x32 pixels in size. Since the images contain three channels, i.e., RGB channels, the actual dimensions of each image are 32x32x3. CIFAR-10 contains a training and test set of 50,000 and 10,000 examples, respectively. A summary of before mentioned dataset characteristics is shown in table 2.

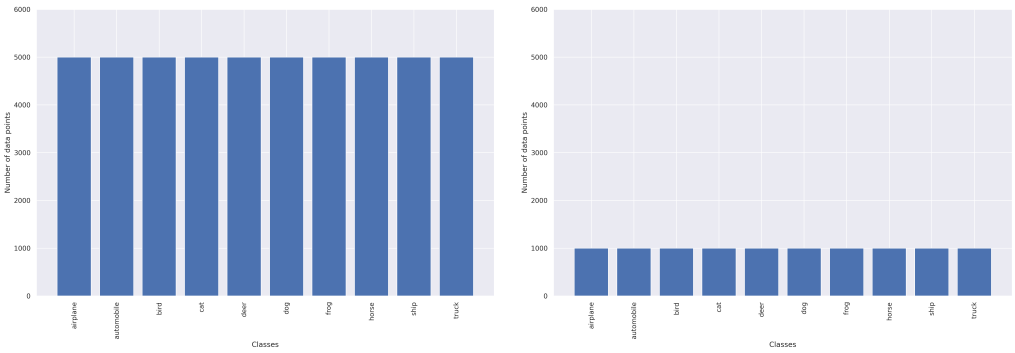
3.2.1 Exploratory Data Analysis

The actual class labels and 10 randomly chosen images of each class are shown in figure 30.

Figure 30: CIFAR-10 samples

In comparison to the MNIST class distribution, the class distribution of the training and test set of CIFAR-10 is precisely uniform as depicted in figure 31.

[4] https://www.cs.toronto.edu/~kriz/cifar.html
[5] https://www.tensorflow.org/api_docs/python/tf/keras/datasets/mnist

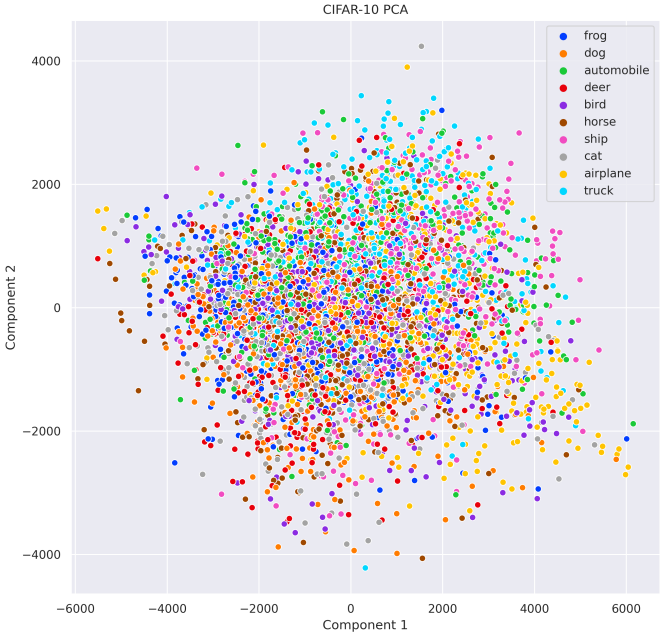


(a) Training set

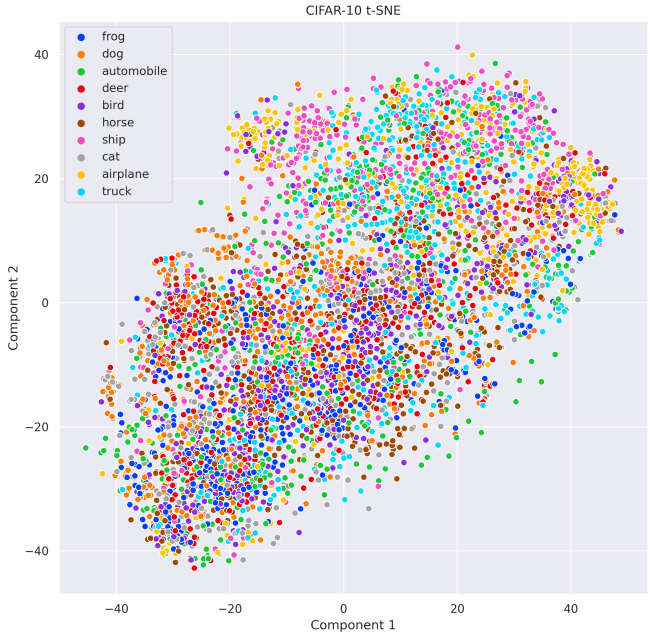
(b) Test set

Figure 31: CIFAR-10 class distribution

Unfortunately, both PCA and t-SNE are unable to produce separable clusters for CIFAR-10 as shown in figure 32. This is probably due to the higher dimensions based on the RGB channels and higher variance among each class. However, that does not necessarily mean that clear decision boundaries can not be learned by non-linear classifiers.



(a) PCA



(b) t-SNE

Figure 32: CIFAR-10 - Visualization of reduced data

3.3 GTSRB

Training samples	Test samples	Dimension	Channels
39,209	12,630	variable	3 (RGB)

Table 3: GTSRB dataset.

The German Traffic Sign Recognition Benchmark (GTSRB) [89] is a single-image, multi-class classification problem that contains a dataset[6] of 43 classes. The full list of all classes with their representation can be found in appendix D. In contrast to the previously presented datasets, GTSRB contains RGB images with variable sizes. The dimensions of images range from 25x25 to 243x225 pixels. A summary of before mentioned dataset characteristics is shown in table 3.

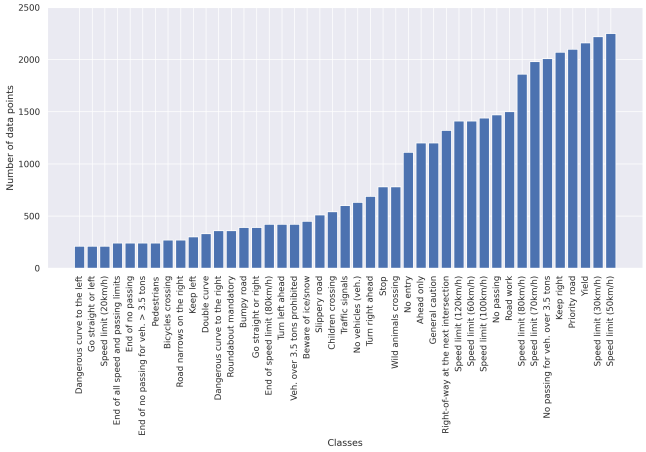
3.3.1 Exploratory Data Analysis

GTSRB contains a training and test set of 39,209 and 12,630 samples, respectively. The sorted distributions based on sample count of both sets is depicted in figure 33.

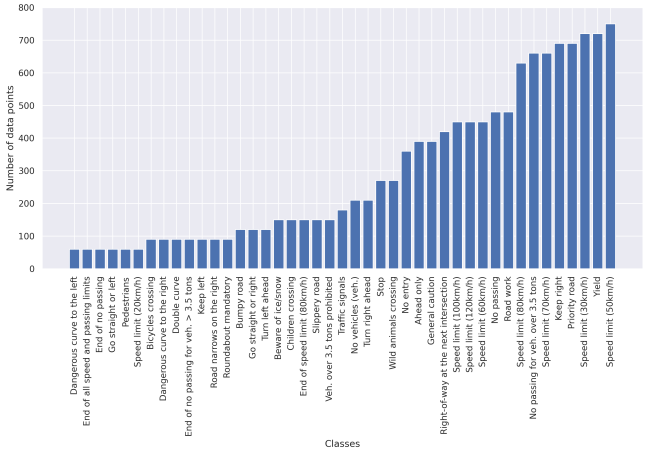
Based on the fact that GTSRB has 33 more classes than the other two datasets which makes it more difficult to have an adequate comparison, later on, a subset of GTSRB is used in this project. The 10 classes with the highest sample count were chosen as subsets for the new training and test set as shown in figure 34. Although the new subset dataset only contains 10 of the 43 original classes, the newly created training set contains 19,620 samples which represent 50% of the original training set. The newly created test set contains 6,480 samples which represent 51% of the original test set.

A list of random samples that were resized to a dimension of 32x32 and drawn from the new subset is depicted in figure 35. The figure shows that the samples vary in their brightness, angle, sharpness, and contrast.

[6] `https://benchmark.ini.rub.de/gtsrb_news.html`

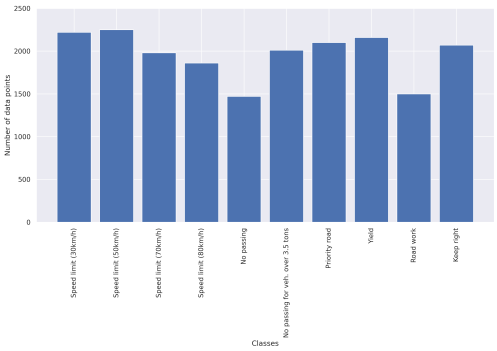


(a) Training set

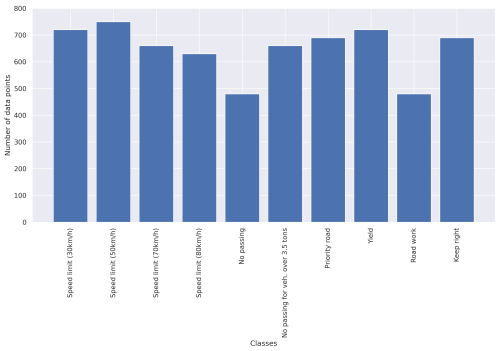


(b) Test set

Figure 33: GTSRB sorted class distribution



(a) Reduced training set



(b) Reduced test set

Figure 34: GTSRB subset

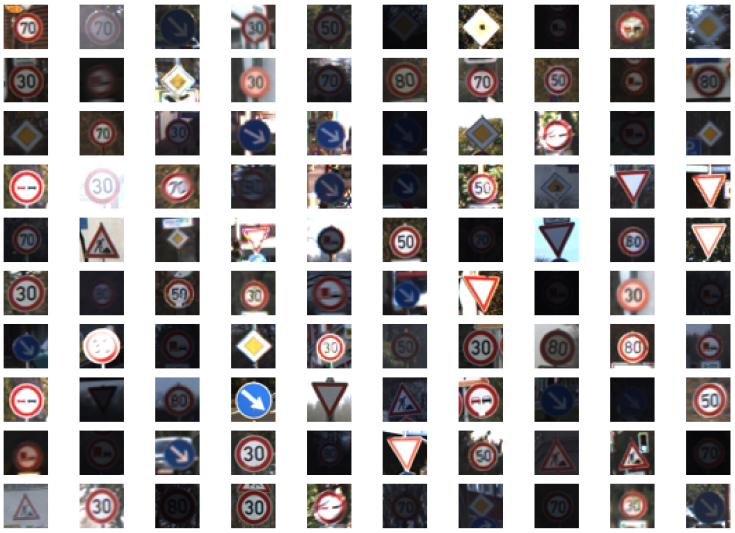


Figure 35: GTSRB random samples

Furthermore, 10 samples from each of the 10 classes have been sampled from the new subset while keeping the natural ordering of the samples in the training set. As depicted in figure 36, the series of samples for each class does not vary much which leads to the assumption that the images were taken from a video file.

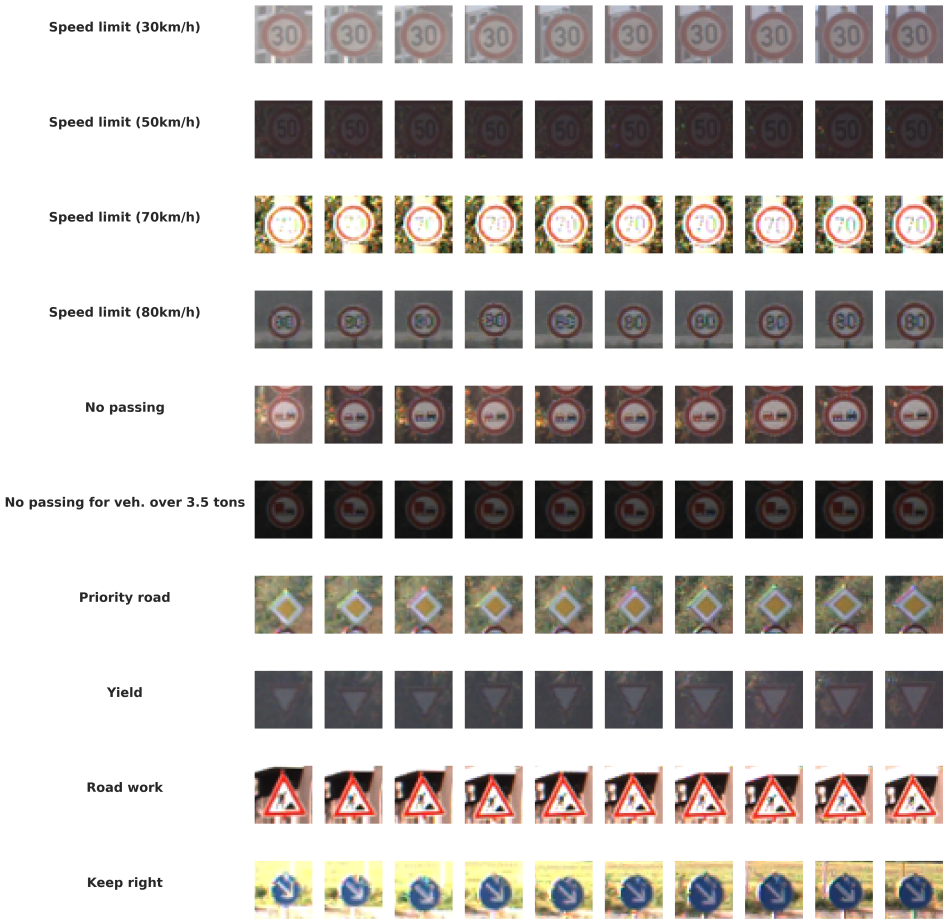
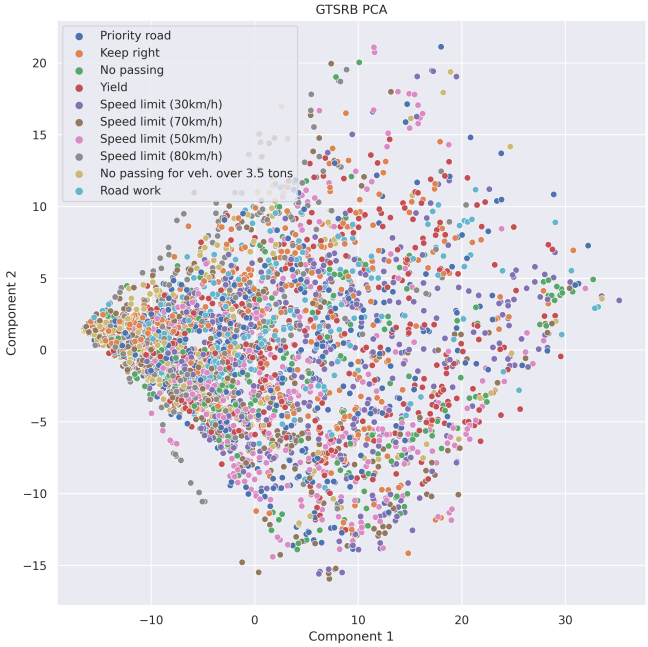
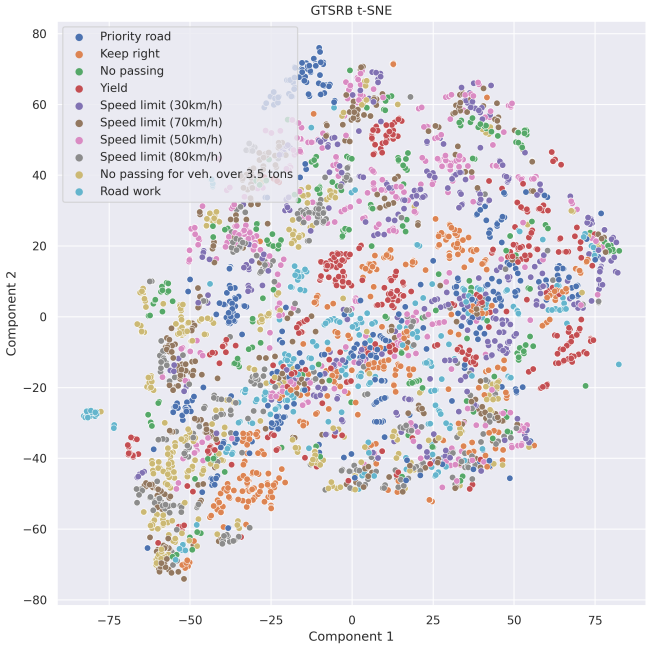


Figure 36: GTSRB sequenced samples

Again, PCA is not able to produce separable clusters for the GTSRB subset as shown in figure 37. However, in contrast to CIFAR-10, t-SNE can produce small clusters which lead to the assumption that the GTSRB subset is easier to learn by non-linear classifiers than CIFAR-10.



(a) PCA



(b) t-SNE

Figure 37: GTSRB - Visualization of reduced data

4 Methodology

The overall idea is to evaluate how far the VGG16 model architecture can approximate secret models trained on the three different datasets discussed in chapter 3. The main approximation procedure is based on different active learning strategies based on the work by [63]. The general experimental setup can be divided into three cases as shown in figure 38.

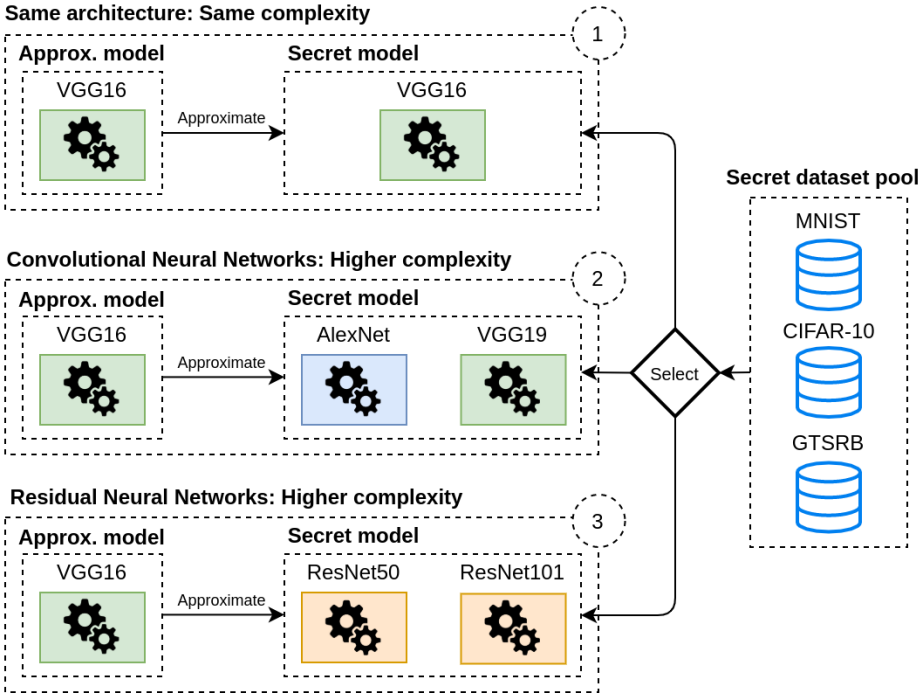


Figure 38: Three approximation cases with different architectures

The first case describes the scenario when both the approximated model and the secret model share the same architecture, i.e., both models are based on the VGG16 architecture. The second case shows the scenario when the approximated model based on VGG16 is less complex than the secret model but the secret model is still based on a Convolutional Neural Network architecture. In this case, the secret model can either be AlexNet or VGG19. The third case deals with the scenario when the approximated model based on VGG16 is less complex than the secret model and the secret model is based on a Residual Neural Network architecture. In this case, the secret model can either be ResNet50 or ResNet101.

The reasoning behind choosing VGG16 as an approximated model with the same or lower complexity than the secret model is that lower complexity approximations are more favorable to formal verification techniques in general. An overview of the complexities of all available models concerning their parameters is shown in table 4.

Every case uses the same general experimental setup that is further explained in the following section.

Model	Total parameters	Trainable parameters
VGG16	14,748,170	14,748,170
VGG19	20,057,866	20,057,866
ResNet50	23,719,498	23,666,378
AlexNet	25,730,506	25,709,350
ResNet101	42,789,962	42,684,618

Table 4: Parameter complexity of all available models.

4.1 General Experimental Setup

Figure 39 gives a broad overview of the general experimental setup that involves the training and approximation phase of the secret model as the two most important phases.

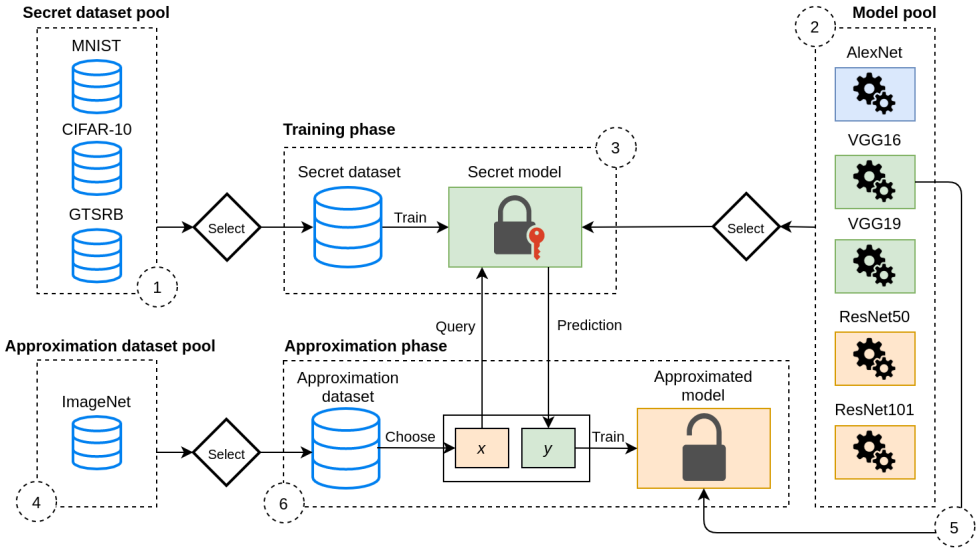


Figure 39: General overview of the model approximation process

1. In the first step, one of the three datasets gets selected out of the secret data set pool. The selected dataset is then used as the secret dataset.

2. The selection of the secret model out of the model pool is the second step. The chosen model is then used as the secret model.

3. After the secret dataset and secret model has been selected, the actual training phase can start as the third step. The training fold of the secret dataset is used to train the secret model.

4. The fourth step involves the selection of the approximation dataset out of the approximation dataset pool. In this case, only one specific dataset, namely ImageNet [16] is available. ImageNet represents a so-called "out-of-distribution" dataset, given that all secret models are solely trained on MNIST, CIFAR-10, or GTSRB.

5. The selection of the approximated model out of the model pool is the fifth step. Given that we only want to evaluate VGG16 concerning its approximation capabilities, VGG16 is selected as the approximated model.

6. The last step represents the approximation phase using active learning selection strategies. VGG16 as the approximated model is trained based on the data samples and corresponding labels that we received from the secret model as the oracle.

The experimental setup for training the secret models is further explained in section 4.2, while further details about the approximation phase can be found in section 4.3.

4.2 Experimental Setup I: Secret Model Training

Training the secret models is a unified approach based on 5-fold cross-validation and is the same for all three datasets. As depicted in figure 40, the chosen secret dataset gets split into a training and test fold which was mostly already done by the publishers of the dataset. The training fold is split once more into 80% reserved for the training and 20% for the validation set. The 5-fold cross-validation guarantees that each of the 5 training fold splits gets a chance to be used as the validation set in the next iteration. The training set is used in 20 epochs to train a secret model. After these 20 epochs, the model gets saved and the next iteration of the 5-fold cross-validation begins with a fresh model that can be trained from scratch. This leads to a collection of 5 different models which all have been trained by using the Adam optimizer with a learning rate of 0.001.

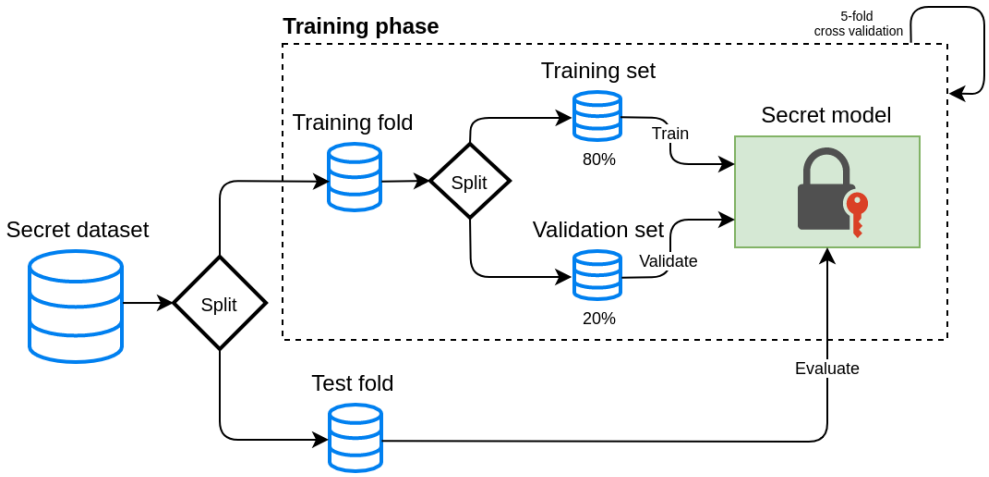


Figure 40: Training phase of the secret model

After the 5-fold cross-validation training phase has been finished, each of the 5 models gets evaluated based on the test fold that none of the models has seen before. The best performing model is then selected as the final secret model for later experiments. Different metrics to monitor the training process like accuracy and loss based on categorical cross-entropy are used during the training phase. The evaluation phase is using the general accuracy of the model using the test fold and a confusion matrix to determine the overall precision qualitatively.

4.3 Experimental Setup II: Approximated Model Training

The approximation phase is the second step and requires already trained secret models as described in section 4.2. Figure 41 shows a detailed overview of the approximation phase that is based on the work by [63].

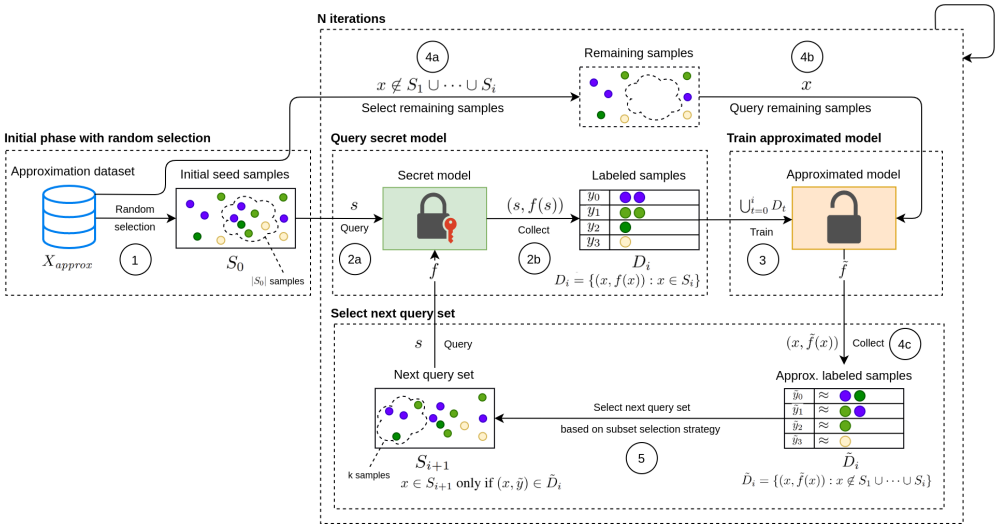


Figure 41: General overview of the model approximation process

1. An initial seed sample subset S_0 gets selected from the approximation dataset based on a random selection strategy. The number of initial seed samples $|S_0|$ is a hyperparameter.

2. Step two is part of the overall iteration loop i where $i = 0, 1, 2, ...N$.

   (a) The subset of samples S_i is used to query the secret model f.

   (b) The secret model returns the tuples $(s, f(s))$ which represent a collection of labeled samples $D_i = \{(x, f(x)) : x \in S_i\}$.

3. The approximated model $\tilde{f}$ is trained iteratively on the union of before mentioned labeled samples which can be represented by the expression $\bigcup_{t=0}^{i} D_t$.

4. After the approximated model has been trained in the current iteration, the approximation dataset is used to select the remaining samples.

   (a) The remaining samples of the approximation dataset are selected in such a way that they are not part of previous selections and can be expressed as $x \notin S_1 \cup ... \cup S_i$

   (b) The remaining samples are used to query the approximated model.

   (c) The approximated model is returnning a collection of tuples $(x, \tilde{f}(x))$ of the remaining samples and can be expressed as $\tilde{D}_i = \{(x, \tilde{f}(x)) : x \notin S_1 \cup ... \cup S_i\}$. In contrast to the labeled samples of the secret model that only contain the top class label, the approximated labeled samples contain the full probability distribution of the classifications.

5. The last step involves the selection of the next query set S_{i+1} of k samples that is part of the approximated labeled samples. This condition can be expressed as $x \in S_{i+1}$ only if $(x, \tilde{y}) \in \tilde{D}_i$. The selection is based on five different active learning strategies.

The five different active learning strategies that were used by [63] are also used for this experiment.

1. **Random strategy**: The first strategy is based on random sampling where k samples are selected uniformly at random.

2. **Uncertainty strategy**: The second strategy is based on uncertainty sampling and is further explained in section 2.3. The k samples with the highest entropy values calculated by $-\sum_j \tilde{y}_{n,j} \log \tilde{y}_{n,j}$ are selected for the next iteration, where j is the index of the label and $\tilde{y}_n$ are the predicted probability vectors.

3. **K-center strategy**: The K-center strategy is a solution to the core-set selection problem discussed in section 2.3. The greedy k-center algorithm [79] uses the initial seed samples as cluster centroids. In the following iterations, k samples are selected next which are the most distant to $\tilde{y}_n$ of all centroids.

4. **DFAL strategy**: The DFAL (DeepFool Active Learning) strategy is based on the work of [58] and is further discussed in section 2.4. DeepFool is applied to every sample x_n. The resulting perturbated sample $\hat{x}$ should then get misclassified by the approximated model. After that, the perturbation $\alpha_n = ||x_n - \hat{x}_n||_2^2$ gets computed. The k samples x_n with the lowest perturbation α_n are then selected for the next iteration.

5. **DFAL + K-center strategy**: This strategy is a combination of the DFAL and K-center strategy. While DFAL can explore the decision boundary effectively using adversarial attacks, it does not account for a redundant selection of data points. The K-center strategy can be used as a diversity measure to reduce the redundancy of the DFAL strategy. The DFAL strategy is used first and selects an initial subset of samples. Out of these samples, a subset of size k is selected next by the K-center strategy.

The success of the approximation phase is measured by two metrics, namely the agreement of both models on a hold-out test set as depicted in equation 4.3.2 where $\mathbf{I}(\cdot)$ is the Indicator function (equation 4.3.1) and the transferability of adversarial examples [91] which have been crafted on the approximated model to the secret model. Adversarial examples were generated using the FGSM attack [23] at a rate of $\epsilon = 0.25$ where transferability is calculated as the fraction of perturbed secret dataset samples which are misclassified by the secret model. The FGSM attack is further explained in section 2.4.

$$\mathbb{1}_A(x) := \begin{cases} 1 & \text{if } x \in A \\ 0 & \text{if } x \notin A \end{cases} \tag{4.3.1}$$

$$\text{Agreement}(f, \tilde{f}) = \frac{1}{|X^{test}_{secret}|} \sum_{x \in X^{test}_{secret}} \mathbf{I}(f(x) = \tilde{f}(x)) \tag{4.3.2}$$

5 Implementation

The overall implementation of this project is based on Python version 3.7.1 and Tensorflow version 2.4.1. The Keras backend of Tensorflow was used to implement all model architectures that are depicted in section B. Adversarial attacks such as FGSM and DeepFool were implemented by using the library Cleverhans version 4.0.0[7].

[7] `https://github.com/cleverhans-lab/cleverhans`

5.1 Data Preprocessing

To have a unified pipeline, it was necessary to preprocess all images of all three datasets into the same dimensions and value ranges. Therefore, all images have been normalized into the range between 0 and 1 and have matching dimensions of 32x32x3. Given that MNIST contains gray-scale images of only one channel, this channel was duplicated twice to mimic the dimensions of an RGB image as depicted in figure 42.

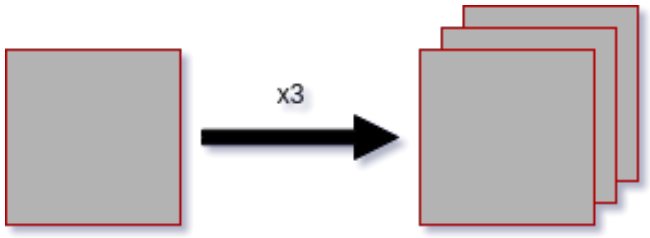


Figure 42: Duplication of gray-scale channel to mimick dimensions of RGB images.

Image augmentation or special cropping techniques that preserve the aspect ratios of the images have not been used.

5.2 Training and Approximation Phase

The training phase is using the Adam optimizer with a learning rate of 0.001. All secret models have been trained using 5-fold cross-validation for 20 epochs and a batch size of 256 images. No regularizer or early stopping criterion has been applied. 80% of the training set has been used as a training set, while the remaining 20% were used as a validation set.

The approximation phase is using query budgets of the size 2.5k, 5k, and 10k for all active learning strategies which are less than the experiments by [63]. The FGSM attack is used with $\epsilon = 0.25$. A downsampled variant of ImageNet[8] with the dimensions of 32x32x3 has been used as the approximation dataset.

[8] `http://www.image-net.org/download-images`

6 Experiment I - Results

This section contains the results of the secret model training phase where all five models have been trained on the three datasets that were introduced in chapter 3. The accuracies of the best performing models on all three datasets together with the corresponding means and standard deviations are depicted in table 5. The best-performing model for each dataset is highlighted in bold.

VGG16	**Top acc**	**Mean**	**Std**
MNIST	99.291	99.032	0.309
CIFAR-10	**80.909**	79.734	0.621
GTSRB	99.796	99.501	0.284

VGG19	**Top acc**	**Mean**	**Std**
MNIST	**99.4**	98.995	0.274
CIFAR-10	80.269	79.206	1.286
GTSRB	99.745	81.677	35.284

AlexNet	**Top acc**	**Mean**	**Std**
MNIST	94.383	86.128	4.947
CIFAR-10	56.05	51.042	4.416
GTSRB	99.209	76.442	19.917

ResNet50	**Top acc**	**Mean**	**Std**
MNIST	99.233	96.530	5.129
CIFAR-10	79.369	77.052	1.605
GTSRB	99.847	97.253	4.091

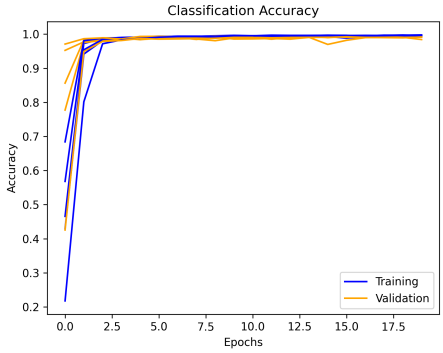
ResNet101	**Top acc**	**Mean**	**Std**
MNIST	99.175	98.867	0.334
CIFAR-10	77.63	76.304	0.953
GTSRB	**99.872**	99.067	0.963

Table 5: Results of the training phase for all models on all three datasets after 20 epochs.

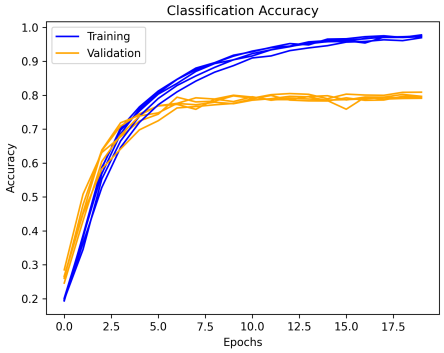
The VGG16 architecture has the highest validation accuracy on the CIFAR-10 dataset with a score of 80.909% which is slightly better than the VGG19 architecture with 80.269%. On the other hand, VGG19 has the highest validation accuracy on the MNIST dataset with a score of 99.4% which is slightly better than VGG16 with 99.291%. The ResNet101 architecture is the winner when it comes to the GTSRB dataset with a validation accuracy of 99.872% although all other models also seem to perform similarly in the 99% range. The worst performing model for the MNIST and CIFAR-10 dataset is AlexNet with a validation accuracy of 94.383 and 56.05, respectively.

6.1 VGG16

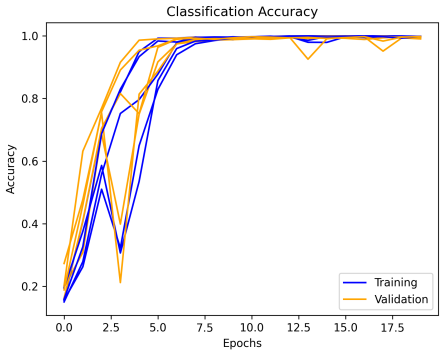
According to table 5 the best performing model for CIFAR-10 is VGG16 with an accuracy of 80.909%. This accuracy is only slightly ahead of VGG19 with an accuracy of 80.269% and ResNet50 with an accuracy of 79.369%. Although VGG16 seems to be the model with the lowest complexity, it seems to perform the best overall and does not suffer from large standard deviations like VGG19 or AlexNet on GTSRB. The corresponding accuracy scores concerning the epoch for VGG16 for all three datasets are depicted in figure 43.



(a) VGG16 accuracy on MNIST



(b) VGG16 accuracy on CIFAR-10

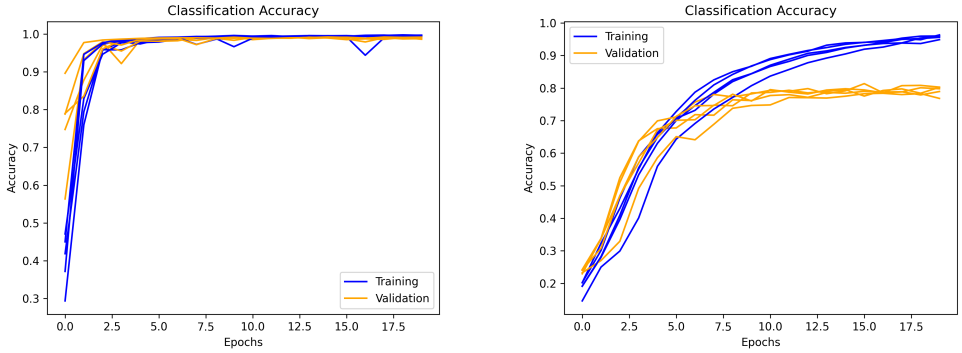


(c) VGG16 accuracy on GTSRB

Figure 43: VGG16 training performance on all three datasets with regard to accuracy.

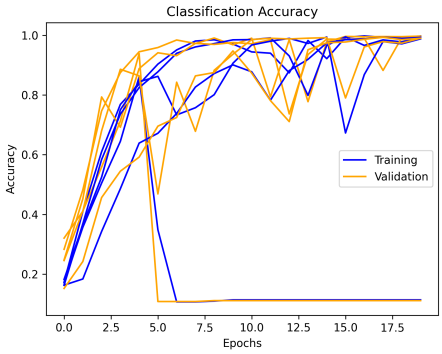
6.2 VGG19

VGG19 is the best performing model on the MNIST dataset with an accuracy of 99.4%. The training accuracy for the MNIST dataset is almost equal to the validation accuracy. Figure 44 shows that the training accuracy is almost the same as the validation accuracy but suffers from a large variance in terms of accuracy on the GTSRB dataset. The validation accuracy for the CIFAR-10 dataset is almost 20% worse than the training accuracy.



(a) VGG19 accuracy on MNIST

(b) VGG19 accuracy on CIFAR-10

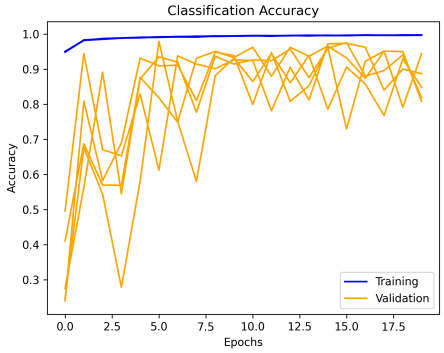


(c) VGG19 accuracy on GTSRB

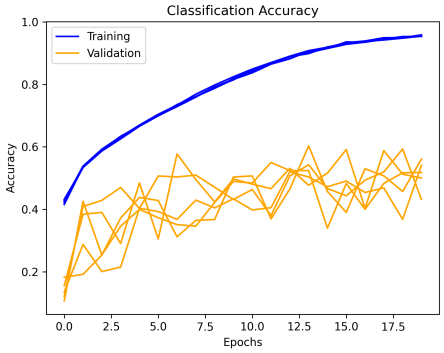
Figure 44: VGG19 training performance on all three datasets with regard to accuracy.

6.3 AlexNet

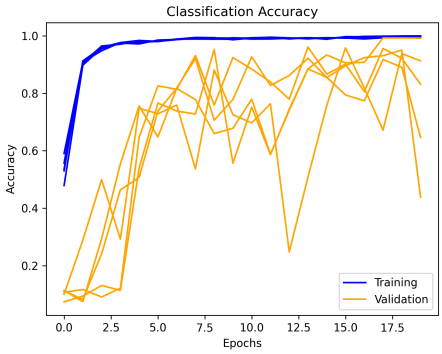
AlexNet does not seem to outperform any other model on any other dataset that was used. Figure 45 shows that AlexNet seems to suffer from a large variance in validation accuracy which is particularly visible on the GTSRB dataset where the standard deviation is 19.917. Furthermore, AlexNet does not seem to converge to a particular value for the GTSRB dataset when only the validation set is inspected.



(a) AlexNet accuracy on MNIST



(b) AlexNet accuracy on CIFAR-10

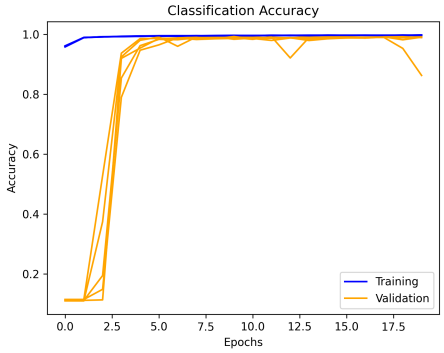


(c) AlexNet accuracy on GTSRB

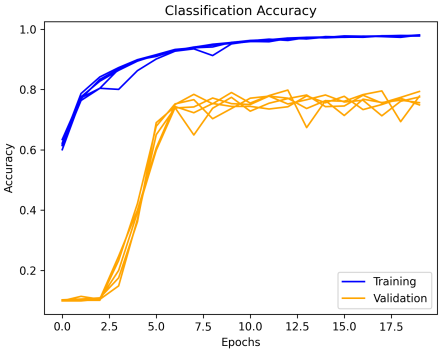
Figure 45: AlexNet training performance on all three datasets with regard to accuracy.

6.4 ResNet50

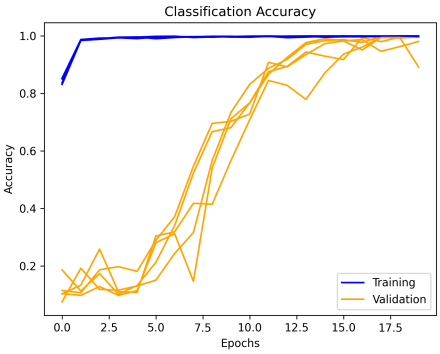
The ResNet50 architecture is performing only slightly worse than the other architectures on the MNIST data but both the training and validation accuracy seem to converge to the same accuracy of around 99% as depicted in figure 46. ResNet50 is the second-best performing model on the GTSRB dataset with an accuracy of 99.847% right behind the ResNet101 architecture but only seems to accomplish a mediocre score of 79.369% on the CIFAR-10 dataset.



(a) ResNet50 accuracy on MNIST



(b) ResNet50 accuracy on CIFAR-10

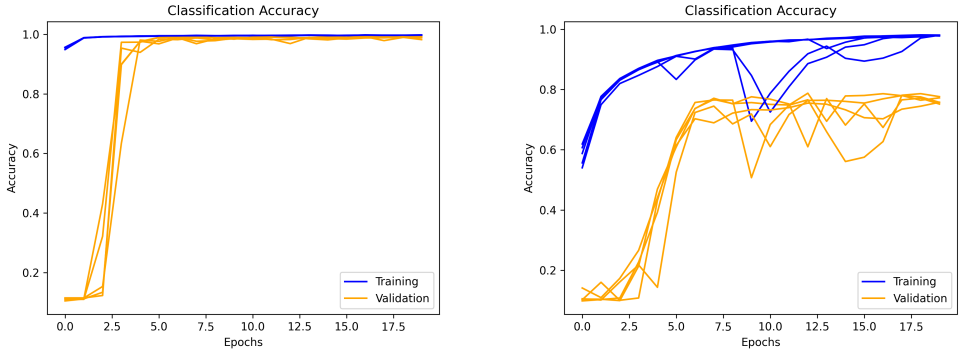


(c) ResNet50 accuracy on GTSRB

Figure 46: ResNet50 training performance on all three datasets with regard to accuracy.

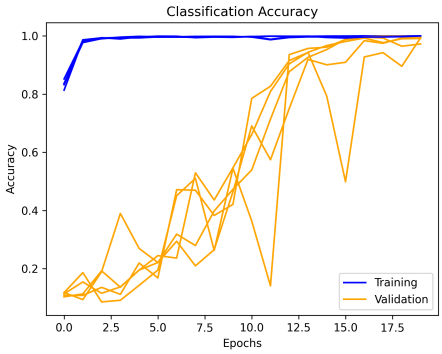
6.5 ResNet101

ResNet101 is the best-performing model architecture on the GTSRB dataset with an accuracy of 99.872% where ResNet50 is performing only slightly worse. Figure 47 also shows that ResNet101 does not seem to suffer from large standard deviations like the related ResNet50 architecture and seems to be more resilient when it comes to weight initializations.



(a) ResNet101 accuracy on MNIST

(b) ResNet101 accuracy on CIFAR-10



(c) ResNet101 accuracy on GTSRB

Figure 47: ResNet101 training performance on all three datasets with regard to accuracy.

7 Experiment II - Results

This chapter summarizes the results of the approximation phase where the VGG16 architecture is used to approximate all other secret model architectures using the five different active learning strategies. The performance of the approximation phase is measured by the overall agreement after certain query budgets of 2500 (2.5k), 5000 (5k), and 10,000 (10k). Furthermore, the transferability is measured using the indicator function and the FGSM attack.

7.1 Agreement

The accuracy performances of the different model architectures using the five different active learning strategies on the MNIST dataset are depicted in table 6. The best performing strategy is highlighted in bold and was achieved by the VGG16 architecture on the same VGG16 architecture using a combination of DFAL and K-center. This combination achieved an agreement of 89.75% on the MNIST hold-out test set after a query budget of 10,000 samples. The worst performing active learning strategy is random sampling by using the AlexNet architecture with an agreement of only 70.5% after 10,000 samples. This is probably due to the lower accuracy of AlexNet on the MNIST dataset with a score of 94.383% as depicted in table 5.

The same agreement overview as it is shown for MNIST in table 6 is also shown in

VGG16	**2.5k**	**5k**	**10k**	**VGG19**	**2.5k**	**5k**	**10k**
Random	39.8	59.55	85.63	Random	38.95	60.83	84.2
Uncertainty	45.79	67.8	87.4	Uncertainty	46.10	65.77	86.92
K-center	46.78	65.88	88.32	K-center	47.22	63.58	87.21
DFAL	59.6	69.48	89.1	DFAL	60.7	68.13	89.45
DFAL+K-center	62.32	78.45	**89.75**	DFAL+K-center	62.32	78.45	89.62

AlexNet	**2.5k**	**5k**	**10k**	**ResNet50**	**2.5k**	**5k**	**10k**
Random	29.77	42.33	70.5	Random	37.25	55.12	81.85
Uncertainty	36.91	55.10	71.86	Uncertainty	44.25	62.78	82.98
K-center	38.41	59.18	72.78	K-center	45.78	63.54	84.13
DFAL	39.09	58.07	72.90	DFAL	59.54	64.08	84.18
DFAL+K-center	38.67	58.17	72.44	DFAL+K-center	58.10	64.78	85.11

ResNet101	**2.5k**	**5k**	**10k**
Random	39.11	59.7	84.1
Uncertainty	45.91	65.14	86.35
K-center	47.75	64.2	86.19
DFAL	61.80	68.01	86.96
DFAL+K-center	61.90	77.84	87.47

Table 6: The agreement in percentages of the five different approximated models on the secret test set of MNIST. The first column represents the name of the used model together with the active learning strategy. Columns 2, 3, and 4 represent the query budgets that have been used to train the approximated model.

table 7 for the CIFAR-10 dataset. The best performing active learning strategy is again achieved by the combination of DFAL and K-center on the VGG16 architecture with an agreement of 63.79%. The worst performing strategy is again random sampling achieved on the AlexNet architecture with an agreement of only 45.12% after 10,000 queries. The overall bad agreement performance is probably due to the CIFAR-10 dataset itself since all models were unable to achieve an accuracy score in the 90% range. Furthermore, chapter 3 also shows that the CIFAR-10 dataset is hard to cluster when it comes to dimensionality reduction techniques such as PCA and t-SNE.

Table 8 shows the agreement overview of the different active learning strategies for the GTSRB dataset. It is prominent that the best active learning strategy is again a combination of DFAL and K-center on the VGG16 architecture with an agreement of 71.88%. A similar overview is shown for the MNIST dataset in table 6 where AlexNet is again the worst performing architecture to approximate. The worst performing active learning strategy is again random sampling on the AlexNet architecture with an agreement of 67.40% after 10,000 queries.

VGG16	**2.5k**	**5k**	**10k**	**VGG19**	**2.5k**	**5k**	**10k**
Random	23.55	48.96	60.85	Random	23.78	47.15	60.41
Uncertainty	25.89	49.14	63.89	Uncertainty	24.99	51.12	63.87
K-center	27.84	50.54	64.85	K-center	27.99	49.15	63.79
DFAL	29.44	52.19	62.94	DFAL	29.38	51.98	62.84
DFAL+K-center	26.78	51.79	**63.79**	DFAL+K-center	26.11	50.49	61.13

AlexNet	**2.5k**	**5k**	**10k**	**ResNet50**	**2.5k**	**5k**	**10k**
Random	15.79	30.15	45.12	Random	21.96	46.08	59.49
Uncertainty	16.78	32.74	46.77	Uncertainty	23.45	50.09	60.41
K-center	19.45	33.49	46.36	K-center	25.13	50.56	62.74
DFAL	21.49	34.94	47.84	DFAL	28.94	50.14	62.75
DFAL+K-center	20.48	33.79	46.79	DFAL+K-center	25.74	49.79	61.94

ResNet101	**2.5k**	**5k**	**10k**
Random	20.45	42.18	58.02
Uncertainty	22.31	48.19	59.66
K-center	23.23	49.46	60.79
DFAL	27.71	48.33	61.20
DFAL+K-center	26.76	47.88	60.94

Table 7: The agreement in percentages of the five different approximated models on the secret test set of CIFAR-10. The first column represents the name of the used model together with the active learning strategy. Columns 2, 3, and 4 represent the query budgets that have been used to train the approximated model.

VGG16	**2.5k**	**5k**	**10k**	**VGG19**	**2.5k**	**5k**	**10k**
Random	40.1	59.79	68.89	Random	38.22	59.91	69.12
Uncertainty	44.81	67.95	69.56	Uncertainty	46.57	65.89	70.10
K-center	45.98	66.21	70.49	K-center	48.18	64.95	70.88
DFAL	58.78	69.10	71.19	DFAL	55.78	65.79	71.16
DFAL+K-center	60.75	66.20	**71.88**	DFAL+K-center	62.39	68.87	70.94

AlexNet	**2.5k**	**5k**	**10k**	**ResNet50**	**2.5k**	**5k**	**10k**
Random	29.10	41.96	67.40	Random	29.15	41.71	68.95
Uncertainty	37.44	53.49	68.59	Uncertainty	37.89	51.92	69.13
K-center	37.89	58.19	70.79	K-center	38.24	58.56	69.14
DFAL	38.13	59.58	70.51	DFAL	38.79	60.24	69.79
DFAL+K-center	37.95	58.54	71.75	DFAL+K-center	38.83	59.21	69.59

ResNet101	**2.5k**	**5k**	**10k**
Random	40.79	61.79	68.10
Uncertainty	47.54	66.79	70.51
K-center	49.26	65.74	69.06
DFAL	55.97	66.21	70.34
DFAL+K-center	59.74	67.14	70.11

Table 8: The agreement in percentages of the five different approximated models on the secret test set of GTSRB. The first column represents the name of the used model together with the active learning strategy. Columns 2, 3, and 4 represent the query budgets that have been used to train the approximated model.

7.2 Transferability

The transferability is measured by using the FGSM attack [46] with $\epsilon = 0.25$ on the approximated VGG16 model and then using the same perturbed samples on the secret model. The agreement on the misclassified samples is then measured in percentages between the approximated model and the secret model that has been approximated with 10,000 queries. Table 9 shows the transferability scores of the active learning techniques applied to all secret models that have been trained on MNIST. The highest transferability was achieved by the DFAL technique on the VGG16 architecture with a percentage of 49.23%. The worst performing model was approximated by using the random sampling technique where the secret model was AlexNet with a percentage of only 21.45%.

	VGG16	**VGG19**	**AlexNet**	**ResNet50**	**ResNet101**
Random	33.98	32.02	21.45	32.54	33.74
Uncertainty	36.45	37.43	24.43	35.46	36.74
K-center	42.12	40.30	26.27	41.32	41.56
DFAL	**49.23**	47.45	30.29	45.53	44.33
DFAL+K-center	46.08	45.74	28.42	43.95	42.84

Table 9: Transferability on the MNIST dataset using the FGSM attack after an approximation phase of 10,000 samples.

Table 10 shows the transferability scores for the secret models that have been trained on the CIFAR-10 dataset. The highest transferability was again achieved by the DFAL technique on the VGG16 architecture with a percentage of 59.05%. Even though CIFAR-10 seems to be harder to learn according to the results in table 5, the overall transferability appears to be higher than for the MNIST dataset. Again, the worst-performing transferability is achieved by the AlexNet approximation using random sampling with a percentage of 33.37%.

	VGG16	**VGG19**	**AlexNet**	**ResNet50**	**ResNet101**
Random	43.28	41.46	33.37	41.57	42.57
Uncertainty	46.47	47.47	42.68	45.63	45.67
K-center	51.56	31.41	36.74	51.46	51.03
DFAL	**59.05**	57.35	40.58	55.07	56.08
DFAL+K-center	54.26	56.57	37.46	53.27	53.69

Table 10: Transferability on the CIFAR-10 dataset using the FGSM attack after an approximation phase of 10,000 samples.

The overall transferability for the GTSRB dataset seems to be higher than for MNIST or CIFAR-10 according to table 10. The best performing approximated model is again VGG16 in conjunction with the DFAL technique achieving a transferability score of 68.67%. The worst transferability is again based on the AlexNet approximation using random sampling with a transferability score of 42.22%.

	VGG16	**VGG19**	**AlexNet**	**ResNet50**	**ResNet101**
Random	53.63	52.84	42.22	52.92	53.84
Uncertainty	55.35	58.65	53.74	56.36	55.42
K-center	62.67	40.96	45.98	62.52	61.58
DFAL	**68.67**	66.38	51.56	66.34	65.01
DFAL+K-center	65.79	66.17	48.34	62.47	62.16

Table 11: Transferability on the GTSRB dataset using the FGSM attack after an approximation phase of 10,000 samples.

8 Discussion

The results of chapter 7 show that it is possible to approximate a black-box deep neural network to a certain degree while a combination of DFAL and K-center yields the highest agreement percentage of 89.75% with the VGG16 architecture on the MNIST dataset. This agreement percentage has been achieved by only using 10,000 data samples. This is significantly less than the number of data samples that were used to train the secret models in chapter 6. A quick calculation reveals how big the reduction of data samples is. Given that each model has been trained for 20 epochs where an epoch is defined as one training cycle using the whole dataset, then training a model on the MNIST dataset took $60000 \times 0.8 \times 20 = 960000$ samples. 60,000 is the original size of the training set. This training set has been used in a 5-fold cross-validation setup and therefore, only 80% of the training set has been used for training while the other 20% were used as a validation set. This means that only $\frac{10000}{960000} \approx 0.0104 \approx 1\%$ of the original dataset has been used to achieve a classifier with an agreement of 89.75% where the secret model has an accuracy of 99.291%. However, this is only the case for the MNIST dataset when the approximated model and the secret model share the same architecture. The reduction in training samples for the other datasets is also significant but only accounts for roughly 30% of the original dataset size. The assumption is that the higher the query budget for the active learning strategy is, the higher the overall agreement. However, the reduction in training samples would still be significant.

While the combination of DFAL and the K-center algorithm yields the best results for achieving high model agreements, it is not the best choice for high transferability. According to the results in section 7.2, the best active learning strategy to achieve a high transferability is the DFAL algorithm based on DeepFool. DFAL reached a transferability percentage of 49.23% for MNIST, 59.05% for CIFAR-10, and 68.67% for GTSRB when both models share the VGG16 architecture. A possible explanation would be that DFAL is based on adversarial crafting techniques that are especially suited for exploring decision boundaries. These explored decision boundaries are crucial components for transferability. On the other hand, the overall agreement on the test set is maximized by using a combination of DFAL and K-center since K-center reduces the number of redundant samples and explores the sample space globally. However, one curious observation is that the transferability for the GTSRB and CIFAR-10 datasets is generally higher than for MNIST. One logical assumption would be that crafting adversarial examples on a dataset like MNIST would yield better results than for GTSRB or CIFAR-10 but the opposite is the case. This is probably due to the more versatile feature space of both GTSRB and CIFAR-10 given that they both contain RGB images while MNIST originally only contains gray-scale images. This more versatile feature space may offer a larger attack vector for adversarial crafting techniques and therefore yield better transferability results. All these findings are in line with the work of [63].

9 Conclusion and Future Work

It was shown that active learning strategies can reduce the number of training samples that are needed to approximate a black-box deep neural network. The best performing strategy is a combination of DFAL and the K-center strategy when the approximated and the secret model share the same architecture, e.g., when both models are based on VGG16. The worst performing active learning strategy is random sampling for all evaluated setups. While the overall performance of random sampling may be bad for all models, it is especially bad in the situation when VGG16 tries to approximate the AlexNet architecture that already has a low accuracy score of 56.05% on the CIFAR-10 dataset. The combination of DFAL and K-center seems to be effective because DFAL is based on adversarial crafting techniques to explore the decision boundaries and K-center acts as a diversity measurement that reduces the number of redundant samples. The approximation of black-box deep neural networks is therefore feasible with a lower query budget by using active learning techniques. The approximated model seems to share similar decision boundaries which are visible by inspecting the agreement and transferability scores. The initial research questions can be answered as follows:

1. Is it possible to approximate black-box machine learning models by using active learning and adversarial attacks to evaluate the robustness?

   - Answer: A combination of DFAL and the greedy K-center algorithm as an active learning subset selection strategy can be used to approximate black-box deep neural networks to a certain degree. The higher the query budget the better the overall agreement of the models on a hold-out test set. While DFAL is able to explore the decision boundaries, K-center is able to explore the sample space more globally. Robustness can therefore be approximated by a combination of both algorithms.

2. Do approximated machine learning models share the same decision boundaries as their black-box counterpart models?

   - Answer: Given that the decision boundaries are not directly visible, transferability can be used as a proxy measurement for shared decision boundaries between models. The results show that the DFAL algorithm is best suited to explore the decision boundaries and crafting adversarial attacks that work on both the approximated and the secret model.

3. Do the transferability and the agreement between the black-box and approximated model vary when the architectures differ?

   - Answer: There is a slight difference between transferability and agreement between model architectures according to the results in chapter 7. The most prominent case is the combination of AlexNet as the secret model and VGG16 as the approximated model. Although the difference in both metrics is high in this case, it may be also due to the overall poor performance of AlexNet on all three datasets. The assumption here is that the transferability and agreement only vary slightly when the architectures differ and that the highest deviation in these two metrics is due to a low training accuracy of the original secret model and not the difference in architectures.

Future work may involve even simpler neural network architectures as an approximation than VGG16. LeNet may be an appropriate candidate since simple network architectures can be used for formal verification methods [39].
Another research direction may involve the synthesis of an entire approximation dataset which may make the use of an out-of-distribution dataset like ImageNet superfluous. Techniques like generative adversarial networks (GANs) may be used to generate samples on the fly and could be used in conjunction with active learning techniques.

References

[1] Diabetic retinopathy detection. `https://www.kaggle.com/c/diabetic-retinopathy-detection/data`. Accessed: 2020-03-15.

[2] Dana Angluin. Queries and concept learning. *Machine learning*, 2(4):319–342, 1988.

[3] Giuseppe Ateniese, Giovanni Felici, Luigi V Mancini, Angelo Spognardi, Antonio Villani, and Domenico Vitali. Hacking smart machines with smarter ones: How to extract meaningful data from machine learning classifiers. *arXiv preprint arXiv:1306.4447*, 2013.

[4] Buse Gul Atli, Sebastian Szyller, Mika Juuti, Samuel Marchal, and N Asokan. Extraction of complex dnn models: Real threat or boogeyman? *arXiv preprint arXiv:1910.05429*, 2019.

[5] Gyora M Benedek and Alon Itai. Learnability with respect to fixed distributions. *Theoretical Computer Science*, 86(2):377–389, 1991.

[6] Battista Biggio, Igino Corona, Davide Maiorca, Blaine Nelson, Nedim Šrndić, Pavel Laskov, Giorgio Giacinto, and Fabio Roli. Evasion attacks against machine learning at test time. In *Joint European conference on machine learning and knowledge discovery in databases*, pages 387–402. Springer, 2013.

[7] Christopher M Bishop. *Pattern recognition and machine learning*. springer, 2006.

[8] Anselm Blumer, Andrzej Ehrenfeucht, David Haussler, and Manfred K Warmuth. Occam's razor. *Information processing letters*, 24(6):377–380, 1987.

[9] Cristian Buciluǎ, Rich Caruana, and Alexandru Niculescu-Mizil. Model compression. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 535–541, 2006.

[10] Patryk Chrabaszcz, Ilya Loshchilov, and Frank Hutter. A downsampled variant of imagenet as an alternative to the cifar datasets. *arXiv preprint arXiv:1707.08819*, 2017.

[11] David Cohn, Les Atlas, and Richard Ladner. Improving generalization with active learning. *Machine learning*, 15(2):201–221, 1994.

[12] David A Cohn, Zoubin Ghahramani, and Michael I Jordan. Active learning with statistical models. *Journal of artificial intelligence research*, 4:129–145, 1996.

[13] Daniel Cullina, Arjun Nitin Bhagoji, and Prateek Mittal. Pac-learning in the presence of adversaries. In *Advances in Neural Information Processing Systems*, pages 230–241, 2018.

[14] George Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, 2(4):303–314, 1989.

[15] Ido Dagan and Sean P Engelson. Committee-based sampling for training probabilistic classifiers. In *Machine Learning Proceedings 1995*, pages 150–157. Elsevier, 1995.

[16] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

[17] Yinpeng Dong, Fangzhou Liao, Tianyu Pang, Hang Su, Jun Zhu, Xiaolin Hu, and Jianguo Li. Boosting adversarial attacks with momentum. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 9185–9193, 2018.

[18] Melanie Ducoffe and Frederic Precioso. Adversarial active learning for deep networks: a margin based approach. *arXiv preprint arXiv:1802.09841*, 2018.

[19] Vasisht Duddu, Debasis Samanta, D Vijay Rao, and Valentina E Balas. Stealing neural networks via timing side channels. *arXiv preprint arXiv:1812.11720*, 2018.

[20] Laurene V Fausett. *Fundamentals of neural networks: architectures, algorithms and applications*. Pearson Education India, 2006.

[21] Matt Fredrikson, Somesh Jha, and Thomas Ristenpart. Model inversion attacks that exploit confidence information and basic countermeasures. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, pages 1322–1333, 2015.

[22] Tommaso Furlanello, Zachary C Lipton, Michael Tschannen, Laurent Itti, and Anima Anandkumar. Born again neural networks. *arXiv preprint arXiv:1805.04770*, 2018.

[23] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*, 2014.

[24] Gregory Griffin, Alex Holub, and Pietro Perona. Caltech-256 object category dataset. 2007.

[25] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[26] Dan Hendrycks and Kevin Gimpel. A baseline for detecting misclassified and out-of-distribution examples in neural networks. *arXiv preprint arXiv:1610.02136*, 2016.

[27] John A Hertz. *Introduction to the theory of neural computation*. CRC Press, 2018.

[28] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.

[29] Dorit S Hochbaum and David B Shmoys. A best possible heuristic for the k-center problem. *Mathematics of operations research*, 10(2):180–184, 1985.

[30] Sepp Hochreiter. The vanishing gradient problem during learning recurrent neural nets and problem solutions. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 6(02):107–116, 1998.

[31] Steven CH Hoi, Rong Jin, and Michael R Lyu. Large-scale text categorization by batch mode active learning. In *Proceedings of the 15th international conference on World Wide Web*, pages 633–642, 2006.

[32] Steven CH Hoi, Rong Jin, Jianke Zhu, and Michael R Lyu. Batch mode active learning and its application to medical image classification. In *Proceedings of the 23rd international conference on Machine learning*, pages 417–424, 2006.

[33] Kurt Hornik, Maxwell Stinchcombe, Halbert White, et al. Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366, 1989.

[34] Harold Hotelling. Analysis of a complex of statistical variables into principal components. *Journal of educational psychology*, 24(6):417, 1933.

[35] David H Hubel and Torsten N Wiesel. Receptive fields of single neurones in the cat's striate cortex. *The Journal of physiology*, 148(3):574–591, 1959.

[36] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pages 448–456. PMLR, 2015.

[37] Ajay J Joshi, Fatih Porikli, and Nikolaos Papanikolopoulos. Multi-class active learning for image classification. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 2372–2379. IEEE, 2009.

[38] Mika Juuti, Sebastian Szyller, Samuel Marchal, and N Asokan. Prada: protecting against dnn model stealing attacks. In *2019 IEEE European Symposium on Security and Privacy (EuroS&P)*, pages 512–527. IEEE, 2019.

[39] Guy Katz, Clark Barrett, David L Dill, Kyle Julian, and Mykel J Kochenderfer. Reluplex: An efficient smt solver for verifying deep neural networks. In *International Conference on Computer Aided Verification*, pages 97–117. Springer, 2017.

[40] Manish Kesarwani, Bhaskar Mukhoty, Vijay Arya, and Sameep Mehta. Model extraction warning in mlaas paradigm. In *Proceedings of the 34th Annual Computer Security Applications Conference*, pages 371–380, 2018.

[41] Simon Kornblith, Jonathon Shlens, and Quoc V Le. Do better imagenet models transfer better? In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2661–2671, 2019.

[42] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009.

[43] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25:1097–1105, 2012.

[44] Solomon Kullback and Richard A Leibler. On information and sufficiency. *The annals of mathematical statistics*, 22(1):79–86, 1951.

[45] Bogdan Kulynych, Jamie Hayes, Nikita Samarin, and Carmela Troncoso. Evading classifiers in discrete domains with provable optimality guarantees. *arXiv preprint arXiv:1810.10939*, 2018.

[46] Alexey Kurakin, Ian Goodfellow, and Samy Bengio. Adversarial examples in the physical world. *arXiv preprint arXiv:1607.02533*, 2016.

[47] Alina Kuznetsova, Hassan Rom, Neil Alldrin, Jasper Uijlings, Ivan Krasin, Jordi Pont-Tuset, Shahab Kamali, Stefan Popov, Matteo Malloci, Tom Duerig, et al. The open images dataset v4: Unified image classification, object detection, and visual relationship detection at scale. *arXiv preprint arXiv:1811.00982*, 2018.

[48] Yann LeCun, Bernhard Boser, John S Denker, Donnie Henderson, Richard E Howard, Wayne Hubbard, and Lawrence D Jackel. Backpropagation applied to handwritten zip code recognition. *Neural computation*, 1(4):541–551, 1989.

[49] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[50] Kimin Lee, Kibok Lee, Honglak Lee, and Jinwoo Shin. A simple unified framework for detecting out-of-distribution samples and adversarial attacks. In *Advances in Neural Information Processing Systems*, pages 7167–7177, 2018.

[51] Taesung Lee, Benjamin Edwards, Ian Molloy, and Dong Su. Defending against model stealing attacks using deceptive perturbations. *arXiv preprint arXiv:1806.00054*, 2018.

[52] David Lewis et al. Reuters-21578. *Test Collections*, 1, 1987.

[53] David D Lewis and William A Gale. A sequential algorithm for training text classifiers. In *SIGIR'94*, pages 3–12. Springer, 1994.

[54] Shiyu Liang, Yixuan Li, and R Srikant. Principled detection of out-of-distribution examples in neural networks. *arXiv preprint arXiv:1706.02690*, 2017.

[55] Daniel Lowd and Christopher Meek. Adversarial learning. In *Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining*, pages 641–647, 2005.

[56] Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of machine learning research*, 9(Nov):2579–2605, 2008.

[57] Brad Miller, Alex Kantchelian, Sadia Afroz, Rekha Bachwani, Edwin Dauber, Ling Huang, Michael Carl Tschantz, Anthony D Joseph, and J Doug Tygar. Adversarial active learning. In *Proceedings of the 2014 Workshop on Artificial Intelligent and Security Workshop*, pages 3–14, 2014.

[58] Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, and Pascal Frossard. Deepfool: a simple and accurate method to fool deep neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2574–2582, 2016.

[59] Robert Moskovitch, Nir Nissim, Dima Stopel, Clint Feher, Roman Englert, and Yuval Elovici. Improving the detection of unknown computer worms activity using active learning. In *Annual Conference on Artificial Intelligence*, pages 489–493. Springer, 2007.

[60] Chigozie Nwankpa, Winifred Ijomah, Anthony Gachagan, and Stephen Marshall. Activation functions: Comparison of trends in practice and research for deep learning. *arXiv preprint arXiv:1811.03378*, 2018.

[61] Seong Joon Oh, Bernt Schiele, and Mario Fritz. Towards reverse-engineering black-box neural networks. In *Explainable AI: Interpreting, Explaining and Visualizing Deep Learning*, pages 121–144. Springer, 2019.

[62] Tribhuvanesh Orekondy, Bernt Schiele, and Mario Fritz. Knockoff nets: Stealing functionality of black-box models. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4954–4963, 2019.

[63] Soham Pal, Yash Gupta, Aditya Shukla, Aditya Kanade, Shirish Shevade, and Vinod Ganapathy. Activethief: Model extraction using active learning and unannotated public data.

[64] Nicolas Papernot, Patrick McDaniel, Ian Goodfellow, Somesh Jha, Z Berkay Celik, and Ananthram Swami. Practical black-box attacks against machine learning. In *Proceedings of the 2017 ACM on Asia conference on computer and communications security*, pages 506–519. ACM, 2017.

[65] Nicolas Papernot, Patrick McDaniel, Somesh Jha, Matt Fredrikson, Z Berkay Celik, and Ananthram Swami. The limitations of deep learning in adversarial settings. In *2016 IEEE European symposium on security and privacy (EuroS&P)*, pages 372–387. IEEE, 2016.

[66] Nicolas Papernot, Patrick McDaniel, Xi Wu, Somesh Jha, and Ananthram Swami. Distillation as a defense to adversarial perturbations against deep neural networks. In *2016 IEEE Symposium on Security and Privacy (SP)*, pages 582–597. IEEE, 2016.

[67] Karl Pearson. Liii. on lines and planes of closest fit to systems of points in space. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 2(11):559–572, 1901.

[68] Li Pengcheng, Jinfeng Yi, and Lijun Zhang. Query-efficient black-box attack by active learning. In *2018 IEEE International Conference on Data Mining (ICDM)*, pages 1200–1205. IEEE, 2018.

[69] Lei Pi, Zhuo Lu, Yalin Sagduyu, and Su Chen. Defending active learning against adversarial inputs in automated document classification. In *2016 IEEE Global Conference on Signal and Information Processing (GlobalSIP)*, pages 257–261. IEEE, 2016.

[70] Luca Pulina and Armando Tacchella. An abstraction-refinement approach to verification of artificial neural networks. In *International Conference on Computer Aided Verification*, pages 243–257. Springer, 2010.

[71] Luca Pulina and Armando Tacchella. Challenging smt solvers to verify neural networks. *Ai Communications*, 25(2):117–135, 2012.

[72] Ariadna Quattoni and Antonio Torralba. Recognizing indoor scenes. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 413–420. IEEE, 2009.

[73] Erwin Quiring, Daniel Arp, and Konrad Rieck. Forgotten siblings: Unifying attacks on machine learning and digital watermarking. In *2018 IEEE European Symposium on Security and Privacy (EuroS&P)*, pages 488–502. IEEE, 2018.

[74] Robert Nikolai Reith, Thomas Schneider, and Oleksandr Tkachenko. Efficiently stealing your machine learning models. In *Proceedings of the 18th ACM Workshop on Privacy in the Electronic Society*, pages 198–210. ACM, 2019.

[75] Frank Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958.

[76] Sebastian Ruder. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*, 2016.

[77] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. *International journal of computer vision*, 115(3):211–252, 2015.

[78] Franco Scarselli and Ah Chung Tsoi. Universal approximation using feedforward neural networks: A survey of some existing methods, and some new results. *Neural networks*, 11(1):15–37, 1998.

[79] Ozan Sener and Silvio Savarese. Active learning for convolutional neural networks: A core-set approach. *arXiv preprint arXiv:1708.00489*, 2017.

[80] Burr Settles. Active learning literature survey. Technical report, University of Wisconsin-Madison Department of Computer Sciences, 2009.

[81] Burr Settles and Mark Craven. An analysis of active learning strategies for sequence labeling tasks. In *Proceedings of the 2008 Conference on Empirical Methods in Natural Language Processing*, pages 1070–1079, 2008.

[82] Claude E Shannon. A mathematical theory of communication. *Bell system technical journal*, 27(3):379–423, 1948.

[83] Samuel Sanford Shapiro and Martin B Wilk. An analysis of variance test for normality (complete samples). *Biometrika*, 52(3/4):591–611, 1965.

[84] Yi Shi, Yalin Sagduyu, and Alexander Grushin. How to steal a machine learning classifier with deep learning. In *2017 IEEE International Symposium on Technologies for Homeland Security (HST)*, pages 1–5. IEEE, 2017.

[85] Reza Shokri, Marco Stronati, Congzheng Song, and Vitaly Shmatikov. Membership inference attacks against machine learning models. In *2017 IEEE Symposium on Security and Privacy (SP)*, pages 3–18. IEEE, 2017.

[86] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[87] Jasper Snoek, Hugo Larochelle, and Ryan P Adams. Practical bayesian optimization of machine learning algorithms. In *Advances in neural information processing systems*, pages 2951–2959, 2012.

[88] Johannes Stallkamp, Marc Schlipsing, Jan Salmen, and Christian Igel. The german traffic sign recognition benchmark: a multi-class classification competition. In *The 2011 international joint conference on neural networks*, pages 1453–1460. IEEE, 2011.

[89] Johannes Stallkamp, Marc Schlipsing, Jan Salmen, and Christian Igel. Man vs. computer: Benchmarking machine learning algorithms for traffic sign recognition. *Neural networks*, 32:323–332, 2012.

[90] Richard S Sutton, Andrew G Barto, et al. *Introduction to reinforcement learning*, volume 135. MIT press Cambridge, 1998.

[91] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199*, 2013.

[92] Sebastian Szyller, Buse Gul Atli, Samuel Marchal, and N Asokan. Dawn: Dynamic adversarial watermarking of neural networks. *arXiv preprint arXiv:1906.00830*, 2019.

[93] Cynthia A Thompson, Mary Elaine Califf, and Raymond J Mooney. Active learning for natural language parsing and information extraction. In *ICML*, pages 406–414. Citeseer, 1999.

[94] Simon Tong and Daphne Koller. Support vector machine active learning with applications to text classification. *Journal of machine learning research*, 2(Nov):45–66, 2001.

[95] Florian Tramèr, Fan Zhang, Ari Juels, Michael K Reiter, and Thomas Ristenpart. Stealing machine learning models via prediction apis. In *25th {USENIX} Security Symposium ({USENIX} Security 16)*, pages 601–618, 2016.

[96] Leslie G Valiant. A theory of the learnable. *Communications of the ACM*, 27(11):1134–1142, 1984.

[97] Jeffrey S Vitter. Random sampling with a reservoir. *ACM Transactions on Mathematical Software (TOMS)*, 11(1):37–57, 1985.

[98] Catherine Wah, Steve Branson, Peter Welinder, Pietro Perona, and Serge Belongie. The caltech-ucsd birds-200-2011 dataset. 2011.

[99] Binghui Wang and Neil Zhenqiang Gong. Stealing hyperparameters in machine learning. In *2018 IEEE Symposium on Security and Privacy (SP)*, pages 36–52. IEEE, 2018.

[100] Paul J Werbos. Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, 78(10):1550–1560, 1990.

[101] Bing Xu, Naiyan Wang, Tianqi Chen, and Mu Li. Empirical evaluation of rectified activations in convolutional network. *arXiv preprint arXiv:1505.00853*, 2015.

[102] Zuobing Xu, Ram Akella, and Yi Zhang. Incorporating diversity and density in active learning for relevance feedback. In *European Conference on Information Retrieval*, pages 246–257. Springer, 2007.

[103] Jason Yosinski, Jeff Clune, Yoshua Bengio, and Hod Lipson. How transferable are features in deep neural networks? In *Advances in neural information processing systems*, pages 3320–3328, 2014.

[104] Xiaoyong Yuan, Pan He, Qile Zhu, and Xiaolin Li. Adversarial examples: Attacks and defenses for deep learning. *IEEE transactions on neural networks and learning systems*, 2019.

[105] Ciyou Zhu, Richard H Byrd, Peihuang Lu, and Jorge Nocedal. Algorithm 778: L-bfgs-b: Fortran subroutines for large-scale bound-constrained optimization. *ACM Transactions on Mathematical Software (TOMS)*, 23(4):550–560, 1997.

A Literature Review

A.1 Practical Black-Box Attacks against Machine Learning

The work of [64] focuses on black-box model attacks where the adversary has no prior knowledge about the internals of the target model like architecture design, hyperparameters, or training data. Given that the target model is used as an instance for labeling inputs, the target model is also referred to as an oracle. The general approach can be divided into two parts.
The first part concentrates on the extraction of the oracle by training the substitute model with inputs labeled by the oracle. The adversary uses synthetic inputs as queries for the oracle which were selected by a Jacobian-based heuristic named "Jacobian-based Dataset Augmentation". The oracle then returns predictions in the form of class labels instead of full probability vectors. When the oracle O is queried with an input $\vec{x}$, it returns a label $\tilde{O}(\vec{x})$ by selecting the j-th component with the highest probability:

$$\tilde{O}(\vec{x}) = \arg \max_{j \in 0..N-1} O_j(\vec{x})$$

The usage of these predictions as training data for the substitute model leads to a substitute model that approximates the decision boundaries of the oracle. Prior research has shown that access to an independently collected labeled training set from the same distribution as the oracle could be used to train a substitute model with a different architecture [91]. This justifies the usage of the Jacobian-based Dataset Augmentation technique in conjunction with initial seed samples that follow a similar training data distribution as the oracle. The main obstacles arise when the adversary has to select an architecture for the substitute model and the amount of queries has to be kept to a minimum to ensure the feasibility of the approach. The main framework of the training process is depicted in figure 48 and is taken from [64].

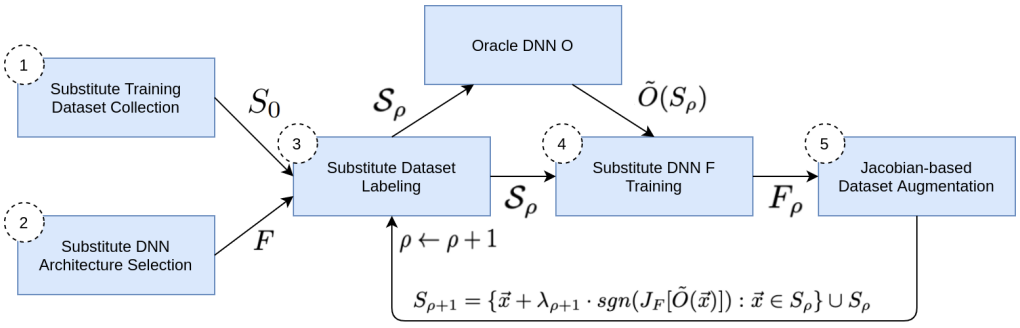


Figure 48: **Training of the substitute DNN** F: the attacker (1) collects an initital substitute training set S_0 and (2) selects an architecture F. Using oracle $\tilde{O}$, the attacker (3) labels S_0 and (4) trains substitute F. After (5) Jacobian-based dataset augmentation, steps (3) through (5) are repeated for several substitute epochs ρ.

Jacobian-based Dataset Augmentation does not apply to black-box models because it identifies the directions in which the model's output is varying the most using an initial set of training points and the Jacobian matrix J_F. The technique evaluates the sign of the Jacobian matrix dimension corresponding to the label assigned to input $\vec{x}$. The new synthetic sample is generated by adding a term $\lambda \cdot sgn(J_F[\tilde{O}(\vec{x})])$ to the original point $\vec{x}$. However, the transferability property enables the usage of the Jacobian-based Dataset Augmentation technique in conjunction with the substitute model that offers

white-box access. The substitute model is iteratively refined in the directions identified using the Jacobian matrix. It should be noted that this technique is not designed to maximize the accuracy of the substitute model but to ensure the approximation of the oracle's decision boundaries with a minimum amount of label queries.
A pseudo-code representation of the five-step training procedure is shown in Algorithm 1 and is taken from [64]. It starts with an initial data collection phase of a balanced set of seed samples. The second step involves the selection of a substitute architecture but taking an informed guess about the problem domain,e.g., a Convolutional Neural Network (CNN) would be a good fit for an image classification problem.

Algorithm 1 - Substitute DNN Training: for oracle $\tilde{O}$, a maximum number max_ρ of substitute training epochs, a substitute architecture F, and an initital training set S_0.

Input: $\tilde{O}, max_\rho, S_0, \lambda$
1: **for** $\rho \in 0..max_\rho - 1$ **do**
2: // Label the substitute training set
3: $D \leftarrow \{(\vec{x}, \tilde{O}(\vec{x})) : \vec{x} \in S_\rho\}$
4: // Train F on D to evaluate parameters θ_F
5: $\theta_F \leftarrow train(F, D)$
6: // Perform Jacobian-based dataset augmentation
7: $S_{\rho+1} \leftarrow \{\vec{x} + \lambda \cdot sgn(J_F[\tilde{O}(\vec{x})]) : \vec{x} \in S_\rho\} \cup S_\rho$
8: **end for**
9: **return** θ_F

The third step uses the oracle to label the samples which are used in the fourth step for training the substitute model. The last step uses the Jacobian-based dataset augmentation technique to produce synthetic samples which are then used as samples for step three. Step three to four are repeated until convergence of accuracy and similarity of decision boundaries.

The second part exploits the fact that the substitute model approximates the decision boundaries of the oracle by crafting adversarial samples, which are misclassified by the oracle based on the transferability of these adversarial samples [23, 91]. The overall goal of the adversary is to create a minimally altered version of input $\vec{x}$ that evades human detection and is named an adversarial sample. The adversarial sample is denoted as $\vec{x}^*$ and should be misclassified by the oracle: $\tilde{O}(\vec{x}^*) \neq \tilde{O}(\vec{x})$. The adversarial sample solves the following optimization problem that adds a minimal perturbation $\delta_{\vec{x}}$ to the original input $\vec{x}$:

$$\vec{x}^* = \vec{x} + \arg\min\{\vec{z} : \tilde{O}(\vec{x} + \vec{z}) \neq \tilde{O}(\vec{x})\} = \vec{x} + \delta_{\vec{x}}$$

In this paper, two different adversarial crafting algorithms are used on top of the substitute training part, namely the Goodfellow et al. algorithm [23] and the Papernot et al. algorithm [65]. While the Goodfellow algorithm can craft adversarial samples rather quickly using the fast gradient sign method with large perturbations, it is potentially easier to detect than the Papernot algorithm. On the other hand, the Papernot algorithm is computationally more expensive while reducing perturbations. The success rate of both techniques is measured by a fixed L1 norm of the perturbation $\delta_{\vec{x}}$ which is defined as

$$||\delta_{\vec{x}}||_1 = \varepsilon \cdot ||\delta_{\vec{x}}||_0$$

where $||\delta_{\vec{x}}||_0$ is the number of input components selected in the perturbation $\delta_{\vec{x}}$, and ε the input variation introduced to each component perturbed. This fixed L1 norm is necessary since both crafting techniques are fine-tuned by different amounts of parameters. While the Goodfellow algorithm relies on the input variation parameter ε, the Papernot algorithm additionally uses a maximum distortion parameter Υ to the input variation parameter. While the Goodfellow algorithm uses the cost function gradient to craft adversarial samples, the Papernot algorithm uses input components with decreasing adversarial saliency values which are added to the perturbation $\delta_{\vec{x}}$. The addition of input components is continued until the adversarial sample is misclassified.

The attack is validated on remote and local classifiers by using the MetaMind, Google, and Amazon service providers and local deployments of oracles. The MNIST [49] and GTSRB [88] datasets are used in both scenarios. The attack forced the DNN trained on MetaMind on the MNIST dataset to misclassify 84.24% of adversarial examples, while a second, locally trained DNN on GTSRB was forced to misclassify more than 64.24%. Even though two different sets for initial substitute training were used, a proper subset of MNIST and one handcrafted set, the success rate, and transferability are similar. The evaluation of the attack on the GTSRB dataset also shows that there is no strong correlation between substitute accuracy and transferability. The evaluation of the attacks on the Amazon oracle with an initial accuracy of 92.17% also shows the success of the attack with a 96.19% misclassification rate. The evaluation of the initial accuracy of the Google oracle with 92% shows similar success with an 88.94% misclassification rate.

It is further shown that the architecture of the substitute model has a limited impact on transferability as long as the complexity is similar or higher. Both adversarial crafting techniques, Goodfellow and Papernot, yield similar results in terms of transferability, although a value above 0.4 for the input variation parameter ε should be avoided for the Goodfellow algorithm since it does not increase transferability any further. A reduction of oracle querying can be achieved by using reservoir sampling [97] that randomly selects κ samples from a list of samples.
A generalization of the attack can also be achieved by replacing the substitute DNN with a logistic regression model or replacing the DNN oracle with logistic regression, SVM, decision trees, and nearest neighbor classifiers.

Three different defense strategies against the proposed attack were evaluated but do not seem to provide any kind of protection.
Gradient masking describes a technique that constructs a model that does not have useful gradients, e.g., nearest neighbor classifiers. Given that the substitute DNN model is approximating the decision boundaries of the oracle and provides gradients, the proposed attack can overcome this defense strategy.
Adversarial training is another defense strategy and incorporates adversarial samples during the training phase to make the model more robust [91, 23]. It is shown that this strategy is effective when larger and finite perturbations are used during the training process instead of small and infinitesimal ones.
Defensive distillation [66] is only effective in a scenario where the fast gradient sign method is directly performed on the distilled model. However, since the proposed attack uses a black-box approach with a substitute model, the fast gradient sign method can still evade this defense strategy.

A.2 How to Steal a Machine Learning Classifier with Deep Learning

The authors of [84] show a naive exploratory machine learning attack which is based on an adversary who uses a deep learning classifier in order to steal the functionality of a black-box model. The focus of an exploratory attack lies on uncovering information about the inner workings of a model [3], [21], [95] and therefore differs from causative or often called poisoning attacks [65], [69] and evasion attacks [6], [64], [46]. The proposed exploratory attack follows a three-step approach without prior knowledge about the type, structure, or underlying parameters of the original classifier as depicted in figure 49. This three-step approach makes it possible to steal the functionality of an SVM or Naive Bayes classifier by using a more complex deep learning classifier while the reverse is not possible.
The first step involves polling the black-box classifier with input data which is then labeled by the classifier as the second step. The third step is using the labeled input data as a training set for the deep learning classifier.

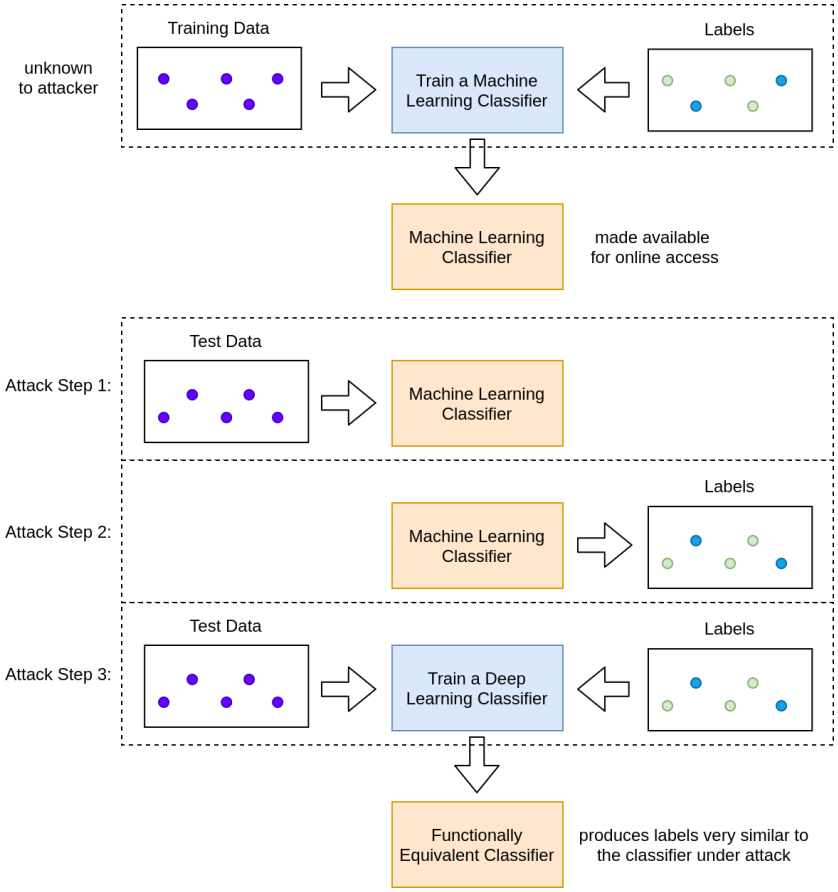


Figure 49: Steps to steal an online machine learning classifier

The experiments are restricted to the task of text classification where the Reuters-21578 dataset [52] is used. The dataset is split into two equally sized sets where the first set is used for the three-step attack and acquiring the labeled training set. After training the deep learning classifier, the second set is used for testing the similarity of labels returned by the black-box and deep learning classifier. The error of similarity is calculated as the percentage of mismatched output labels. It is shown that the higher the

number of labeled training samples, the lower the adversarial classifier error inferring Naive Bayes or SVM. The overall error of the deep learning classifier inferring Naive Bayes with a training set size of 859 samples is around 2.10% while the overall error for SVM is around 2.56%. While the deep learning classifier can infer simpler models like Naive Bayes or SVM, the reverse is not true and is in line with the work of [78], stating that a complex enough feedforward neural network can act as a universal approximator. Further improvements to this naive strategy could be made by using an active learning approach [57], [94].

A.3 Stealing Machine Learning Models via Predicition APIs

In [95] the authors evaluate two different extraction scenarios. The first scenario describes a machine learning prediction API that returns class labels with corresponding confidence values. Newly proposed equation-solving and decision tree path-finding attacks are evaluated by extracting different model types. In the second scenario, the API only returns class labels and omits confidence values completely. An extended version of the Lowd-Meek attack and three different retraining approaches are evaluated in this scenario.
The overall goal of an adversary is to approximate a model f using as few queries as possible. The approximation $\hat{f}$ has to match f as closely as possible by reducing two error measures, namely the average error R_{test} and the uniform error R_{unif}. The general setup of the different extraction attacks is depicted in figure 50 and is taken from [95].

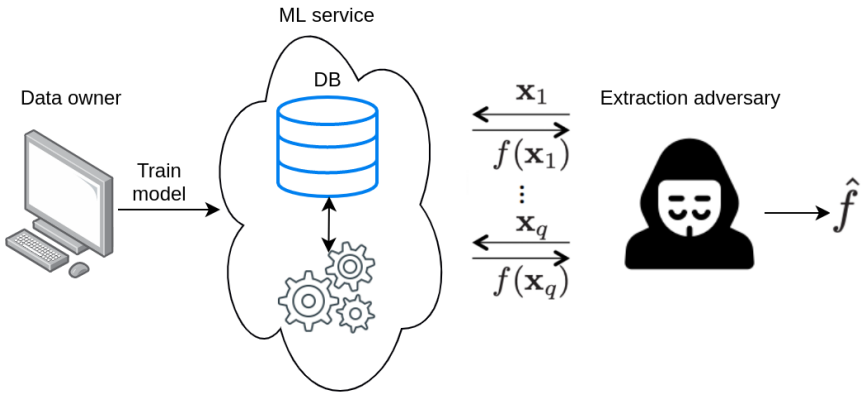


Figure 50: **Diagram of ML model extraction attacks.** A data owner has a model f trained on its data and allows others to make prediction queries. An adversary uses q prediction queries to extract an $\hat{f} \approx f$

Most of the prediction APIs return confidence values together with their corresponding classes. The output of a query, therefore, falls in the range of $[0, 1]^c$ where c is the number of classes. Two different extraction techniques are used for this scenario, namely equation-solving attacks for logistic models and path-finding attacks for decision trees.

For an adversary, returned probabilities act as samples $(x, f(x))$ which can be seen as equations in the unknown model parameters. Simple linear classifiers like binary logistic regression are prone to an equation-solving attack which works based on the returned samples. Binary logistic regression defines a hyperplane of the form $w \cdot x + \beta = 0$ in the feature space $\mathcal{X}$ which is able to separate two classes. Logistic regression is defined mainly by two parameters, namely $w \in \mathbb{R}^d$ and $\beta \in \mathbb{R}$. The output of logistic regression is a probability defined by $f_1(x) = \sigma(w \cdot x + \beta)$, where $\sigma(t) = \frac{1}{(1+e^{-t})}$. The linear equation

$w \cdot x + \beta = \sigma^{-1}(f_1(x))$ can then be solved to recover w and β by using $d+1$ number of queries where d is the number of dimensions in the feature space. The authors are able to extract a binary logistic regression model with $R_{test} = R_{unif} = 0$ which is an 100% accuracy while using $d+1$ predicitions. When the scenario is extended to multiclass logistic regression models and multilayer perceptrons, the equation-solving attack has to deal with a system of non-linear equations. Since there is no analytic solution, a common method for solving this system is by minimizing a loss function like logistic loss. While the loss function for multilayer logistic regression is strongly convex and converges to a global minimum, it is not the case for multilayer perceptrons. Therefore, optimization techniques for multilayer perceptrons may converge to a local minimum which makes it harder for the equation-solving attack to work efficiently. The authors can perfectly extract the multiclass logistic regression model with $k = c \cdot (d+1)$ parameters, where c is the number of classes. In the case of multilayer perceptrons with 20 hidden nodes, an accuracy of $> 99.9\%$ is achieved while using 5,410 samples on average. To further improve the equation-solving attack and lowering the needed amount of queries, the inversion attack explored by [21] is used. By using a combination of first extracting a model $\hat{f} \approx f$ and then using the white-box inversion attack [21], it is possible to extract a softmax model which is faster and needs 20x fewer online queries than the attack proposed by [21].

When it comes to decision trees, they do not calculate class probabilities as a continuous function of their input like logistic models. Decision trees partition their input space into discrete regions where every region is assigned a label and confidence score. Therefore, the before-discussed equation-solving attack is not applicable. By using the returned predictions as pseudo-identifiers, it is possible to calculate the path that the input traversed in the tree. In the proposed path-finding algorithm, the LINE_SEARCH procedure is used for continuous features and CAT_SPLIT for categorical features. Besides this algorithm, a top-down approach is proposed which exploits queries over partial inputs. Starting at the root node, the tree is extracted "layer by layer". Both approaches are evaluated in a real-world scenario by using an online extraction attack on a deployed black-box decision tree on the BigML platform. Both attacks can extract a decision tree with 100% accuracy while the top-down approach is slightly more efficient. For the case when the architecture is not known a priori by the adversary, the authors describe an extract-and-test strategy. Each attack is applied and followed by an evaluation of the R_{test} and R_{unif} metrics. The overall attack is successful when any of the attacks achieves a low error rate. The scenario where model architectures f and $\hat{f}$ do not match is also known as improper extraction. Besides the extract-and-test strategy, it would also make sense to use a neural network as $\hat{f}$, given that neural networks can closely approximate any continuous function [14], [33]. However, it is questionable if this strategy is optimal because a neural network $\hat{f}$ which closely approximates f might have a more complex representation. Tests have shown that the accuracy is indeed inferior to proper extraction attacks. Even with 1000 hidden nodes, the extracted model $\hat{f}$ is a weaker approximation of f with an accuracy of 99.5%. Reducing the extracted model $\hat{f}$ to a simpler architecture seems more efficient and is known in learning theory as Occam Learning [8].

When only class labels are returned by a prediction API, two different approaches are used for model extraction. The first approach is the Lowd-Meek attack [55] and can extract linear models when access to a black-box oracle is given. The oracle is queried with input data and returns the predicted class without confidence values. This restricted case where a model f returns class labels only, describes the membership query setting in learning theory [2], [5], [55], e.g., PAC learning [96]. This attack uses line search to find points that are arbitrary close to the decision boundary of the oracle

or victim. One major drawback of this approach is its restriction to be only applicable to linear classifiers. In [95], this attack is extended to work with non-linear models such as polynomial kernel SVMs by reducing the extraction to a linear SVM in the transformed feature space. While this extended version of the Lowd-Meek attack applies to a polynomial kernel SVM, it cannot be used for kernels with a transformed feature space of infinite dimensions like the radial-basis function (RBF).

The second extraction technique is using retraining approaches. By retraining a local model with training data that was labeled by an oracle and then trying to achieve a low training error, the authors hope to effectively approximate the decision boundaries of the target model. Three different querying strategies are available. The first strategy is retraining with uniform queries where m points $x_i \in \mathcal{X}$ are sampled uniformly at random. These samples are used to query the oracle and to train a local model $\hat{f}$. The second strategy is line-search retraining and is a model-agnostic generalization of the Lowd-Meek attack. m adaptive queries are chosen by using the line-search technique to find samples that are close to the decision boundaries of f. The oracle labels these samples and is used as training data for the local model $\hat{f}$. The third strategy is adaptive retraining and applies techniques from active learning [11], [80]. In the initial phase, the adversary defines an amount of r rounds and a query budget m. The oracle is queried with $\frac{m}{r}$ uniform points which get labeled as the initial training set for model $\hat{f}$. Over a total amount of r rounds, $\frac{m}{r}$ new points are selected for labeling which the local model $\hat{f}$ is least certain about. It is assumed that points which model $\hat{f}$ is least certain about, lie close to the decision boundaries of $\hat{f}$. These points are labeled by the oracle before the local model $\hat{f}$ is retrained.

The extended Lowd-Meek attack and retraining approaches were evaluated against different models and datasets. Out of the three retraining strategies, adaptive retraining is the most efficient when it comes to linear binary models such as logistic regression trained over binary data sets. Adaptive retraining can extract the target model with a 99% accuracy over a test set. When the query budget is large enough, the Lowd-Meek attack outperforms the retraining approach but needs 50x more queries than the newly proposed equation-solving attack. When it comes to multiclass logistic regression models, the Lowd-Meek attack is not applicable in multiclass settings. It is shown that adaptive retraining is again achieving the highest accuracy of 99.9% out of the three strategies but uses 100x more queries than the equation-solving attack. When it comes to neural networks, line-search and adaptive retraining have almost no advantage over uniform retraining. When the query budget is $100 \cdot k$, where k is the number of model parameters, $R_{test} = 99.16\%$ and $R_{unif} = 98.24\%$. This means that this high query budget might prevent any monetary advantages when extracting the model. Another class of non-linear models is SVM with RBF kernels. Since RBF kernels map the original feature space into infinite space, the Lowd-Meek attack is not applicable. Therefore, retraining is the only viable approach in this scenario. Combined with an extract-and-test approach, adaptive retraining is again able to extract the model with an accuracy of over 99%.
Extraction countermeasures such as rounding confidence values to 4 or 5 decimals do not affect the equation-solving or decision tree path-finding attack. Further rounding to 2 or 3 decimals weakens the attacks but they are still more effective than adaptive retraining using class labels only.
The code for the proposed extraction techniques is publicly available under `https://github.com/ftramer/steal-ml`.

A.4 PRADA: Protecting Against DNN Model Stealing Attacks

The work by [38] can be divided into two parts. The first part evaluates novel model extraction attacks by using synthetic sample crafting techniques, while the second part concentrates on detection mechanisms of these and similar extraction attacks.

The first part describes the feasibility of new model extraction attacks on Deep Neural Networks (DNNs). The proposed attacks, named T-RND and COLOR are based on novel approaches for generating synthetic queries and optimizing hyperparameters. The success of the proposed attacks is measured in terms of transferability for the targeted and non-targeted case, and accuracy based on RU- and Test-agreement. Prior work by Tramer [95] and Papernot [64] is used as a baseline to investigate the new techniques with regards to the MNIST [49] and GTSRB datasets [88].
The general strategy of an adversary on how to extract a target model can be split up into basic steps. The first step involves the querying process of the target model with special samples which were crafted to gain a maximum of information in the form of predictions. Predictions can be returned as labels-only or as a full set of probabilities over possible classes. The information gain can be measured by different approaches like certainty, uncertainty, and others. In the second step, the substitute model is trained with the received predictions and will be gradually trained again during the whole process with more samples. The third step is using the substitute model to craft further queries for the target model and refining the substitute model. These steps should be executed by using a minimum amount of queries. A more refined description of the general extraction process is depicted in Algorithm 2 and is explained in greater detail by [38].

Algorithm 2 Model extraction process with the goal of extracting classifier F, given initital unlabeled seed samples X and a substitute model F' (inititally random)

procedure LABEL($\{x_1, ..., x_n\}, F$)
    return $\{\hat{F}(x_1), ..., \hat{F}(x_n)\}$ ▷ Return predictions
end procedure

procedure EXTRACTMODEL(F)
    $U \leftarrow$ Initital data collection
    $L \leftarrow \{U, \text{LABEL}(U, F)\}$
    $F' \leftarrow$ Select architecture
    $H \leftarrow$ Resolve hyperparameters
    $F' \leftarrow \text{INITIALIZE}(F')$ ▷ Set random weights
    $F' \leftarrow \text{TRAIN}(F'|L, H)$
    for $i \leftarrow 1, \rho$ **do** ▷ ρ duplication rounds
        $U \leftarrow$ Create synthetic samples
        $L \leftarrow \{L \cup \{U, \text{LABEL}(U, F)\}\}$
        $F' \leftarrow \text{TRAIN}(F'|L, H)$
    end for
    return F'
end procedure

The success of model extraction attacks is measured by using two metrics, namely prediction accuracy of the substitute model and transferability of adversarial examples. Prediction accuracy of the substitute model has to match the target model and is approximated by using a Random Uniform Agreement (RU-agreement) that is measured

by randomly sampling the input space and a held-out Test Set Agreement (Test-agreement) that both classifiers have not seen before. This ensures that the behavior of the substitute model matches the target model in both seen and unseen sample scenarios.
Transferability of adversarial examples describes a scenario where adversarial examples are crafted by minimal modifications ϵ to an input x of class c such that $x' = x + \epsilon$ is classified as $c' \neq c$ by a substitute model F' while the same effect can be achieved on the target model F by transferring the modified input x from F' to F. Adversarial examples can further be divided into targeted and non-targeted cases. Targeted adversarial examples are crafted by $x + \epsilon$ to change the classification of x from c to a specific class c', while non-targeted adversarial examples change the classification of x to any other class c'.

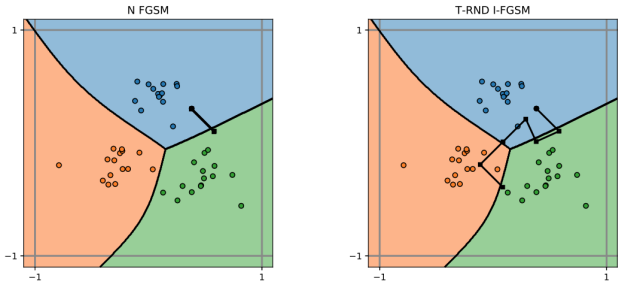
A general model extraction process such as Algorithm 2 and success measurements as mentioned before building the foundation for comparing prior work with the proposed attacks in this paper. Prior work by Tramer [95] investigated transferability on simple models and proposed equation solving attacks on logistic regression models and pathfinding attacks on decision trees. Furthermore, Tramer evaluated different types of retraining attacks in scenarios where equation solving and pathfinding attacks were not applicable. Work by Papernot [64] concentrated on transferability of specific types of adversarial examples such as non-targeted adversarial ones. They further introduced the Jacobian-based Dataset Augmentation (JbDA) technique for generating synthetic samples which is similar to the Fast Gradient Sign Method (FGSM) [23].

Contributions by Tramer and Papernot serve as a baseline for further comparison and to investigate approaches on how to select hyperparameters and generating synthetic samples.
Bayesian hyperparameter optimization[9] [87] is used to test potential candidates for hyperparameter combinations with high expected value or high uncertainty. These candidates are then evaluated by using a five-fold cross-validation approach and is referred to as "CV-SEARCH". The hyperparameter combination that passes both evaluation steps with the highest accuracy is used for the rest of the attack.

Popular choices for adversarial crafting techniques are the Fast Gradient Sign Method (FGSM) [23], Iterative Fast Gradient Sign Method (I-FGSM) [46], and Momentum Iterative Fast Gradient Sign Method (MI-FGSM) [17]. MI-FGSM proved to be the strongest method by winning both the targeted and non-targeted adversarial example challenge at NIPS 2017 Adversarial Attack competition.
In this paper, the authors use two different approaches to generating synthetic samples. The first one constructs synthetic samples by using a partially trained substitute model and is referred to as Jacobian-based Synthetic Sample Generation. Synthetic samples converge towards the decision boundary for all presented techniques but overlapping behavior can be avoided by the newly proposed T-RND technique which steps in a targeted randomly chosen direction. This differs from the approach from Papernot [64] where a non-targeted FGSM technique is used. A graphical comparison between both techniques is depicted in figure 51 and was taken from [38].
The second construction technique is independent of the substitute model and randomly perturbs color channels by step size λ. This technique is referred to as COLOR.

The MNIST [49] and GTSRB datasets [88] were used for evaluating the DNN model extraction attacks because they were also used before by Papernot [64]. The datasets were split up into test, target model training, and attacker set, while 10 different target

[9] https://github.com/fmfn/BayesianOptimization



(a) Non-targeted FGSM (b) T-RND I-FGSM

Figure 51: Synthetic sample generation against a multi-layer perceptron. We show six sequential steps. Left: the nontargeted FGSM [64] does not generate novel data points after the first step. Right: T-RND I-FGSM avoids this by varying the contribution of features, and targeting random classe

models with increasing complexity were trained. The models achieved on average a 98% accuracy on MNIST while a 95% accuracy was achieved on GTSRB. While Papernot used 0.1% natural seed samples and 99.9% synthetic samples, Tramer initially queried random samples which were followed by a line-search to find synthetic samples between existing ones.

Experiments have shown that the learning rate and the number of training epochs of the substitute model do not have to match the target model. For both metrics, agreement, and transferability, CV-SEARCH was able to produce similar or better results.
To achieve a high Test-agreement on the substitute model, it is necessary to have natural seed samples in the initial phase.
Even though the usage of synthetic samples improves agreement, it is less efficient than using natural samples. Transferability can be improved by using a relevant synthetic sample generation method. The proposed T-RND method can improve both agreement and transferability. When it comes to the MNIST dataset, T-RND FGSM was able to improve in terms of Test-agreement and targeted adversarial example crafting. T-RND I-FGSM outperformed before mentioned method in terms of non-targeted adversarial examples. In the more complicated GTSRB dataset, COLOR outperformed all other synthetic crafting methods in terms of Test-agreement, while T-RND I-FGSM achieved better results for targeted adversarial samples.
The usage of full probability vectors improves transferability compared to the usage of labels-only. However, it does not improve the agreement.
A matching architecture between substitute and target model achieves higher transferability. If the architecture of the substitute model is higher or similar as the target model, the substitute model can achieve high predictive performance.
Since the proposed attacks were performed on models which were trained from scratch, it is questionable if they can be applied to models with a higher complexity or models which were trained on more complicated datasets. Prior work evaded this problem by using pre-trained models as both the substitute and target model and only concentrated on fine-tuning these models [62], [68].

The proposed techniques outperformed the work by Papernot [64] on the MNIST dataset by approximately 30% for the targeted and non-targeted adversarial sample case.

Furthermore, the authors were able to improve Test-agreement and targeted adversarial sample crafting on GTSRB by approximately 45%.

The second part concentrates on a generic defense against model extraction attacks by monitoring successive queries of an API user. If the distribution of queries is deviating from a Gaussian distribution, the successive queries are classified as a malicious attack. On the other hand, every query distribution which follows a Gaussian distribution up to a certain threshold is classified as benign. The Shapiro-Wilk test [83] has been chosen as the preferred normality test to check deviations from the Gaussian distribution. PRADA can detect most extraction attacks but can be circumvented by attacks that imitate a Gaussian distribution by using dummy queries or Sybil attacks.

A.5 Efficiently Stealing your Machine Learning Models

The authors of [74] propose a new equation-solving attack on linear and quadratic Support Vector Regression Machines (SVRs) which extend the work of [55] and [95]. They furthermore propose a retraining approach that is suited for nonlinear kernels such as the Radial Basis Function (RBF). While these two attack types are merely used for white-box models, it is also possible to execute a kernel agnostic extraction using the retraining approach by training multiple models at once where each is based on a different kernel. To further increase the efficiency of such an exploratory attack, an evasion attack [45] could be executed beforehand which tries to find input data that results in incorrect predictions by the victim classifier. To evaluate the accuracy of the attacks for both SVMs and SVRs, a new framework [10] was implemented in Python which consists of a Server, Client, Adversary and Supervisor class, depicted in figure 52.

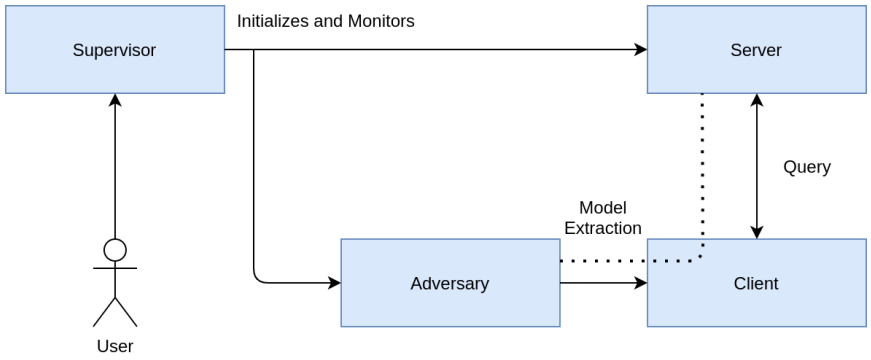


Figure 52: Architecture of the model extraction attack framework.

The framework is used to evaluate the attacks on victim models which were trained on four different datasets. The evaluation is based on model approximation accuracy in terms of an error rate, Mean Squared Error (MSE), and Relative Mean Squared Error (RMSE). Moreover, the costs in terms of the number of queries and computation time is taken into account to evaluate the efficiency of the attacks. In the light of Secure Multi-Party Computation (SMPC) protocols, the proposed attacks are further evaluated in realistic scenarios. The authors show that Wi-Fi localization schemes using SVRs are vulnerable to the proposed attacks despite the supposed protection given by SMPC protocols.

[10] `github.com/robre/attacking-mlaas`

The first attack type focuses on the extraction of SVMs and SVRs which are using a linear or quadratic kernel by using an equation-solving approach. The foundation for the extraction of linear kernels is based on the Lowd-Meek attack [55] which was the first extraction attack on linear classifiers. The Lowd-Meek attack is using an oracle for predictions and both positive and negative samples in the beginning. By changing the feature vector entries of the positive sample individually and then querying the oracle for predictions, the attack tries to craft a combination that lets the oracle falsely classify a positive sample as a negative one. By using a line-search strategy, the attack can find feature vectors that are close to the decision boundary. The Lowd-Meek attack was later on extended by [95] to make it also applicable to some nonlinear kernels, such as the polynomial kernel. This attack is then executed in the transformed feature space where the model is effectively linear. The novelty of the proposed equation-solving attack by [74] is because it enables the attacker to extract SVRs with quadratic kernels. Although the extraction is happening in the transformed feature space, the queries have to be executed in the original one. This requires a computation of the inverse of the transformation function $\phi(x)$ which the authors solve by incorporating the usage of zero vectors and incremental queries.

The second attack type is based on retraining an adversarial model with a training set that is labeled by the victim model. This general setup was already proposed by [95] while using a query budget m. The amount of samples is limited by this budget factor. To find an effective query budget, a so-called "budget factor" a was introduced. The query budget is then expressed as $m = a(d + 1)$ where d is the number of features and $0.5 \leq a \leq 100$. [95] made the observation that most models can be extracted with an accuracy of 99% when $a = 50$. They furthermore introduce three possible approaches to retraining. The first one is a naive approach using a set of m uniformly random samples. The second one is a line-searching retraining strategy as used in the Lowd-Meek attack which tries to find samples that are close to the decision boundary. The last approach is an adaptive retraining attack which uses active learning [11] to improve extraction efficiency. In the first round, a set of uniformly random samples is chosen to be classified by the victim model. The adversarial model is then trained on the before mentioned dataset and the next chosen samples are those which the adversarial model is least certain about. Those samples mostly lie close to the decision boundary.

The authors show that they can extract linear, polynomial, and quadratic SVRs with an accuracy of 99% by using their newly proposed equation-solving attack. Other nonlinear kernels like RBF could be extracted using the adaptive retraining approach although it is significantly slower than the equation-solving attack. Countermeasures like rounding confidence values can weaken the equation-solving attack but do not prevent it entirely. Other countermeasures like extraction monitor, monitoring for suspicious queries, and server-side feature translation might be able to prevent the equation-solving attack but the adaptive retraining attack is still a feasible attack and yields similar results to the equation-solving attack.

A.6 Knockkoff Nets: Stealing Functionality of Black-Box Models

The authors of the paper "Knockoff Nets: Stealing Functionality of Black-Box Models" [62] differentiate the act of classifier stealing into different categories where you could steal parameters [95], hyperparameters [99], architecture [61], information about the training data [85] or the decision boundaries of the black-box model [64]. Their approach concentrates more on duplicating the behavior of the black-box model which is also referred to as a "victim" or a complex function that describes the behavior of the

victim: F_V. The surrogate model which is also called a "knockoff" is then a model which imitates the behavior of F_V and describes a function that was learned by the "adversarial": F_A. Therefore, F_A should be an approximation of F_V while the concrete architecture of both models does not necessarily have to match.

The main difficulty which arises is the acquisition of training data for the knockoff model F_A to imitate the behavior of F_V. The authors make the assumptions that the training and test data, the internal workings of the black-box model, and the semantics of the black-box model outputs are not known a priori. However, the victim model provides access through its API to get predictions for queried image inputs in the form of a $K-$dim posterior probability vector $y \in [0,1]^K, \sum_y y_k = 1$. This puts the adversarial in such a situation where an educated guess about an appropriate, initial dataset has to be made which is then used to query the victim model to get predicted labels. The input and prediction pairs obtained from the victim model are then used as training data for the knockoff and are referred to as "transfer set". The initial dataset should comprise the same input format as the training set of F_V e.g., images. The best scenario for the adversarial would occur when the selection of training images of the victim $x_i \sim P_V(X)$ is identical to the selection of the adversarial $x_i \sim P_A(X)$ to generate a transfer set $\{(x_i, F_V(x_i))\}$ which follows the same distribution as the training set of the victim $D_V = \{(x_i, y_i)\}$. This assures that only informative images are used for querying. The transfer set can also be seen as a training set with pseudo-labels. Given that this is an unrealistic but possible scenario, the authors divide their experimental setup into three different scenarios.
The first scenario describes the case when $P_A = P_V$ which means that the queried images are sampled from the same distribution as the training set used for F_V.
The second scenario is referred to as "Closed-World" where the training data P_V is a proper subset of the image universe P_A. This initial dataset is called D^2 because it is a union of all six datasets which were used in the experiments.
The last scenario is the most challenging one and is referred to as "Open-World" where the adversarial only has access to two datasets that were not used by any victim model for training, namely ILSVRC [16] [77] and OpenImages [47].

In the experiments, four different victim CNNs based on the ResNet-34 [25] architecture with pre-trained ImageNet [16] weights were used. All CNNs were trained for image classification tasks and used four different datasets, namely Caltech-256 [24], CUBS-200-2011 [98], Indoor-Scenes [72] and Diabetic-Retinopathy [1]. Two hold-out datasets were never used for training the victim models but serve the adversarial as initial datasets to choose from to construct the transfer set, namely ILSVRC and OpenImages. To evaluate both the victim and knockoff model, test images of each of these datasets $\mathcal{D}_V^{test}$ were used concerning top-1 accuracy and sample efficiency.

To solve the problem of constructing a transfer set in a query efficient way, the authors evaluate a random and adaptive sampling strategy in the context of the before mentioned three scenarios. The approach is similar to active learning (AL) strategies [12], [94] and is a special case of pool-based AL [81] where the adversary samples from a pool of unlabeled data and takes the samples as inputs for the victim to label. The goal of this approach is to reduce labeling effort to produce a training set for the adversarial to train F_A. The concrete AL strategy is split up into two approaches. The random strategy samples images from distribution without replacement to query F_V which favors exploration.
The adaptive strategy is based on a reinforcement learning approach to learn a policy π which samples images ($x_t \sim \mathbb{P}_\pi(\{x_i, y_i\}_{i=1}^{t-1})$) in a query efficient way while also

aiding interpretability of F_V. The selection of an image is equivalent to action in the reinforcement learning context. Three different rewards are used to evaluate the quality of a sampled image.

The certainty reward is a margin-based measure [37], [81] which favors queries where F_V is confident and therefore indicates that the image is probably from the same image distribution that F_V was trained on. The certainty reward $R^{cert}(y_t) = P(y_{t,k_1}|x_t) - P(y_{t,k_2}|x_t)$ where k_i s the ith-most confident class, would result in an exploting sampling strategy with favored images. Therefore a second reward has to be used which encourages exploration.
The diversity reward $R^{div}(y_{1:t}) = \sum_k \max(0, y_{t,k} - \bar{y}_{t:t-\Delta,k})$ prevents exploitation of a single label.
The last reward favors a high Cross-Entropy (CE) loss $R^{\mathcal{L}}(y_t, \hat{y}_t) = \mathcal{L}(y_t, \hat{y}_t)$ which equals $-\sum_k p(y_k) \log p(\hat{y}_k)$ so that $F_A(x_t)$ does not imitate F_V. Furthermore, the chosen P_A gets supplemented with coarse-to-fine hierarchy information which is used together with the three rewards to update the policy π using the gradient bandit algorithm [90].

The authors discovered that a random query strategy that takes images from a different distribution than the black-box training data results in a "well-performing" knockoff. Moreover, the knockoff model does not have to share the same architecture as the victim model to imitate functionality and their reinforcement learning approach improves query efficiency in a closed-world setting. The "Open-World" scenario is the one producing the worst accuracy and efficiency based on the fact that the initial dataset is not a subset of the training data that F_V was trained on. The adaptive strategy yields the best results where it reaches 68.3% accuracy at the CUBS200 data set 6 times quicker than the random strategy. Moreover, the certainty reward is producing the best results while a combination of all three displays none-to-marginal improvements. Defense techniques for the victim model like only returning top-k unnormalized posterior probabilities, rounding posteriors to r decimals or the extreme case of only returning $k = argmax_k y_k$ is minimally impacting the approach. The choice of the architecture for F_A is sufficient to imitate F_V as long as it is more complex than F_V. This contrasts the findings in the domain of knowledge distillation [22], [28] where a simpler "student" model tries to learn from a "complex" teacher model. However, reasonable complex architecture, such as ResNet [25] or VGG [86] seem to be sufficiently robust when it comes to model compression and serving as a student model in a real-world scenario to capture discriminative features of F_V.

A.7 Extraction of Complex DNN Models

The authors of [4] extend the work of [62] and ask the question if Knockoff Nets are a real threat to modern machine learning models which are deployed as MLaaS solutions. Knockoff Nets have the potential to be a real threat to real-world models because they do not necessarily require knowledge about the architecture, the training data, internal parameters, or hyperparameters of the victim. The authors reproduce and confirm the results of [62] against more complex models and add two different datasets to their setup. Additionally, the authors introduce a new detection technique that can differentiate between queries that are using in- and out-of-distribution samples using a binary classifier based on the ResNet architecture. This new detection technique is also performing well against Knockoff Nets in its original scenario with a detection accuracy of up to 99% while previous defense and detection strategies were only effective against simple models ([64], [38]) and simple attacks ([51], [73]). Furthermore, the authors evaluate the effectiveness of Knockoff Nets in more realistic scenarios and show

limitations of their approach when the adversary has access to unlabeled, natural data which follows a similar distribution as the victim's training set.

The first scenario describes an adversary who uses a surrogate model whose architecture differs from the victim. It is shown that the extraction attack is still effective even when the architectures of both the victim and adversarial do not match under the condition that both architectures use a pre-trained DNN model and are not trained from scratch. The effectiveness of the attack decreases however when the victim model is not based on a pre-trained DNN model but was trained from scratch and tuned for its specific task. The second scenario focuses on the output of the victim model and what happens when a full probability vector as a prediction is reduced to a single predicted class or truncated results such as top-k predictions. It is shown that the effectiveness of Knockoff Nets has a negative correlation with the number of classes when such reduction and truncation techniques are applied. The higher the number of classes, the lower the effectiveness of the Knockoff Net in such a scenario. Commercial prediction APIs often return top-k predictions while at the same time incorporating more than 10000 classes. The effectiveness in such a scenario will probably decrease heavily. On the other hand, when the number of classes is relatively low like below the 200 mark, Knockoff Nets are still effective.
The third scenario investigates the importance of the queries diversity. It is shown that a proper transfer set consisting of natural data which follows a similar distribution as the victim's model training set is a crucial requirement for Knockoff Nets. If the transfer set is not a subset of the training set or consists of unbalanced and underrepresented classes the degradation of effectiveness is severe.
Furthermore, it is shown that approximate knowledge about the victim's training data distribution and access to a large, unlabeled dataset that follows this distribution is sufficient to circumvent the author's proposed detection model to differentiate between in- and out-of-distribution samples [4]. The usage of natural data makes it a difficult task for the detection mechanism to function properly.

The authors evaluate their detection mechanism based on a binary classifier by comparing two different types of models. The first one is a ResNet model trained from scratch while the second one is a pre-trained ResNet model with frozen weights. The final layer of the pre-trained ResNet model is a binary logistic regression layer which was trained with the LBFGS solver [105]. The ImageNet and OpenImages datasets were used as the adversary's initial transfer set to, later on, differentiate between images that come from the victim's or the adversarial's distribution. It is shown that a pre-trained ResNet architecture with 101 layers (RN101*LR) is producing the best accuracy of 93% which is consistent with findings by [103] and [41] who state that pre-trained DNN transfer features better when the tasks are similar. The proposed technique has a high True Positive Rate (TPR) of over 90% and outperforms other state-of-the-art-approaches. However, this performance is only maintained when the prediction APIs have a specific task and are not general-purpose classifiers with thousand of classes. The main goal of this approach is therefore to measure predictive uncertainty by detecting out-of-distribution samples in the domain of image recognition. The binary classifier could act as a filter for a victim and in case of an out-of-distribution sample, returns a random shuffling of the original prediction as a defense. Other methods to measure predictive uncertainty exist like Baseline [26], ODIN [54] or using the Mahalanobis distance [50], [7] but their performance with Knockoff Nets is unknown. This could be the target of future work.

Related work on detection and defense mechanisms focuses on restricting the returned

predictions to classes without confidence scores [95] but it was shown that model extraction attacks against simple models are still effective even when only predicted classes are used [38]. Three main detection approaches exist where the first one focuses on detecting malicious queries which do not follow a normal distribution [38]. The second one is detecting queries that are exploring a large region of the model's input space [40], while the third one identifies queries that are close to the classes' decision boundary [73]. Another approach does not focus on detection or defense directly but rather concentrates on watermarking its property by using an approach named "DAWN" [92]. This approach uses a poisoning attack by returning slightly off predictions to the adversary on purpose so that these predictions get incorporated into the surrogate model. This results in watermarking the surrogate model and can be claimed by the original authors.

Despite the effectiveness of the related work on other models like decision trees, they cannot effectively prevent or detect Knockoff Nets. The authors' proposed detection mechanism is capable of detecting Knockoff Nets by differentiating between in- and out-of-distribution queries and is closely related to PRADA [38] while PRADA fails to detect Knockoff Nets. Nevertheless, this detection mechanism only works properly when the adversary has no access to a large, unlabeled dataset that follows a similar distribution as the victim's training set.

A.8 ActiveThief: Model Extraction using Active Learning and Unannotated Public Data

One of the most recent model extraction approaches is called "ActiveThief" [63] and can extract models efficiently using active learning and unannotated public datasets. Although the authors experimented with both image and textual data, only the results regarding image data are of interest in this project. The three main contributions of the authors make this paper an interesting candidate to consider.

The first one is because one universal thief dataset for each domain is sufficient to extract multiple, separate deep classifiers. Given that the dataset on which the secret model was trained on is inaccessible to the attacker, an educated guess has to be made to decide on a dataset used for querying which is called "thief dataset". ActiveThief uses natural non-problem domain (NNPD) data for their thief dataset in contrast to [95] and [64] who are using synthetic non-problem domain (SNPD) and problem domain (PD) data, respectively. NNPD data may not follow the same distribution as the secret dataset but can be effectively used for model extraction as long as it is of the same type as the secret dataset, e.g. image data. This has the advantage that well known public datasets like a downsampled variant of imagenet [10] or public data crawled from the internet can be used as a thief dataset to query models which have been trained on datasets like MNIST [49], CIFAR-10 [42] or GTSRB [89]. ActiveThief uses an unannotated subset of the training fold of the downsampled variant of imagenet as a proxy for public image data which also may lack proper labels. This yields a 3.4x average improvement in agreement over SNPD data with uniform noise as a baseline.

The second contribution describes the advantage of an active learning subset selection strategy which leads to a reduction of 70% - 90% of the total data used while still achieving 61.52% - 98.18% agreement and a better transferability of adversarial attacks compared to other approaches. The agreement could be improved even more by using more iterations but still using only 30% of the thief dataset. ActiveThief is hereby using a common active learning setup as described by [80] where the attacker achieves greater accuracy with fewer labeled training instances by querying a so-called "oracle" with

unlabeled data to be labeled. A common oracle would be a human annotator but in this setup, the role of the oracle is taken by the secret model. The labeled data is then used to train the substitute model iteratively while every iteration relies on a subset selection strategy to decide which subset to use next for the oracle to label. The ActiveThief framework is using the number of samples to label in each iteration k, the number of iterations N, and the number of initial seed samples $|S_o|$ as hyperparameters. A limited query budget requires a query strategy which is using as few queries as possible while still being effective in extracting the secret model. Exploiting some hierarchical annotations of the thief dataset as described by [62] is not an option because not every dataset exhibits this property and getting those high-quality annotations limits the attacker in the acquisition of suitable data. Therefore, five different active learning subset selection strategies have been evaluated while a combined approach named "DFAL + K-center strategy" is the best performing one. The random strategy and uncertainty strategy that is based on the entropy of predicted probability vectors were the worst-performing strategies. The K-center strategy is a clustering approach based on the greedy K-center algorithm [79] which is using initial seed samples as centroids and then selects k samples which are the most distant to these centroids. This assures that chosen samples are not redundant. The DeepFool-based Active Learning (DFAL) algorithm [18] where DeepFool [58] is applied to every sample, ensures an informative, initial subset selection where informative describes the property that each sample is close to the decision boundary. This strategy is simply referred to as "DFAL" strategy. The combined strategy "DFAL + K-center strategy" is therefore striving for informative and unique samples.

The third one shows that ActiveThief can defeat most of the defense mechanisms like perturbed output probabilities as proposed by [51] or restricting the access to output probabilities by only granting access to the top-1 label [95]. Even though ActiveThief shows better performance by using unperturbed output probabilities, it also works with only the top-1 label. Furthermore, the defense mechanism called "PRADA" [38] which is observing the made queries to the secret model and checking for normality of the distribution of distances between successive queries is unable to detect ActiveThief.

Furthermore, it is shown that limited knowledge about the architecture of the secret model is not a big obstacle and that a successful extraction can be conducted even when the architectures of the secret and substitute model differ. Although it may be possible for an attacker to get information about the model architecture using reverse-engineering [61] [19], the discrepancy in agreement terms is marginal and can be disregarded.
The success of the extraction attack is measured by two metrics, namely the agreement of both models on a test set as depicted in equation A.8.1 where $\mathbf{I}(\cdot)$ is the Indicator function and the transferability of adversarial examples [91] which have been crafted on the substitute model to the secret model. Adversarial examples were generated using the FGSM attack [23] at a rate of $\epsilon = 0.25$ where transferability is calculated as the fraction of perturbed secret dataset samples which are misclassified by the secret model.

$$\text{Agreement}(f, \tilde{f}) = \frac{1}{|X^{test}_{secret}|} \sum_{x \in X^{test}_{secret}} \mathbf{I}(f(x) = \tilde{f}(x)) \tag{A.8.1}$$

The secret test data is only available to ActiveThief during the evaluation process but not in the model extraction process.

B Model Architectures

B.1 AlexNet

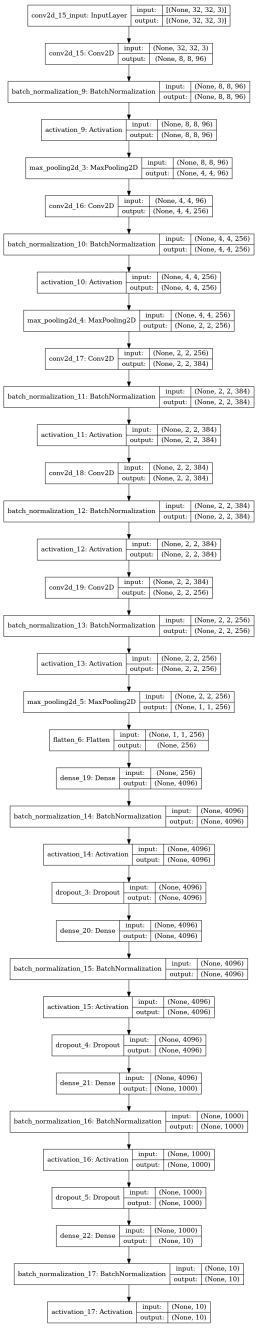


Figure 53: AlexNet

B.2 VGG16

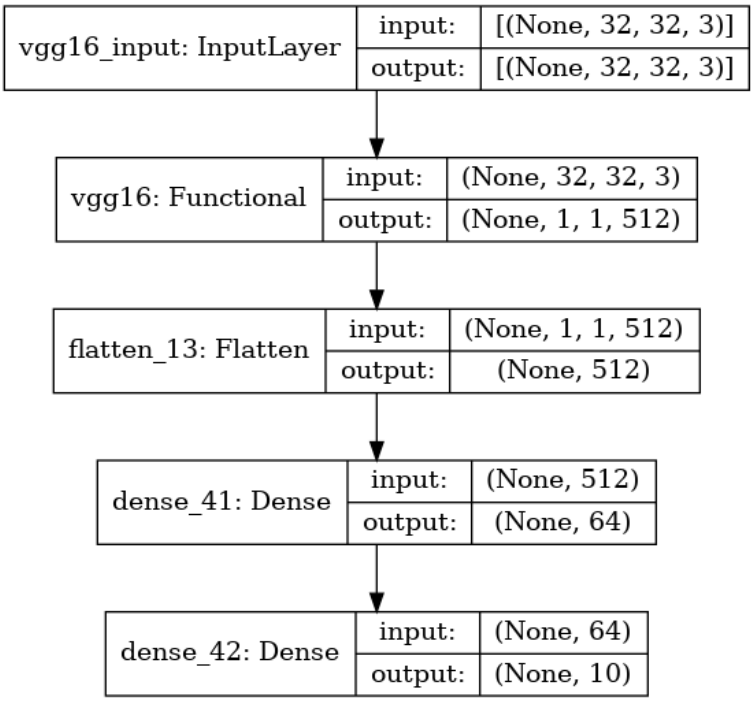


Figure 54: VGG16

B.3 ResNet

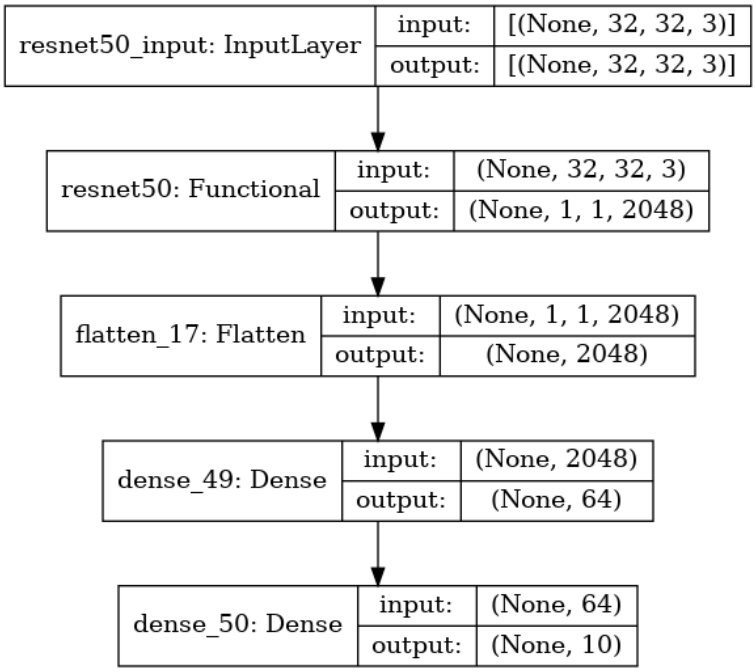
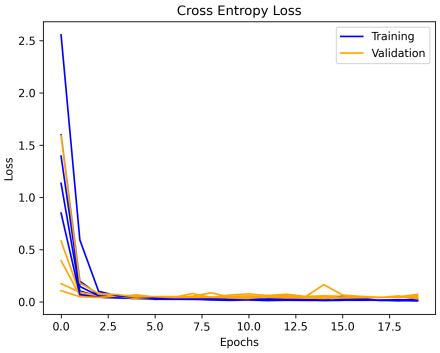


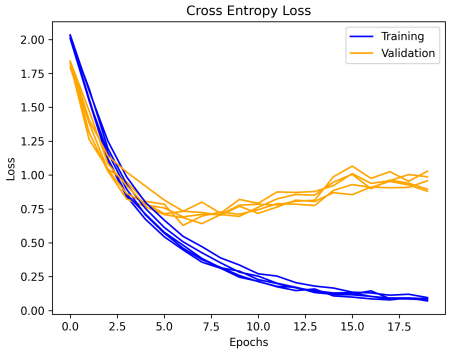
Figure 55: ResNet-50

C Secret Model Training Phase

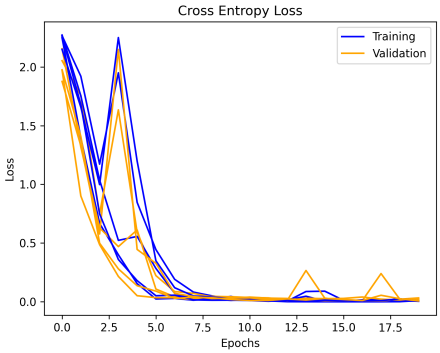
C.1 VGG16



(a) VGG16 loss on MNIST

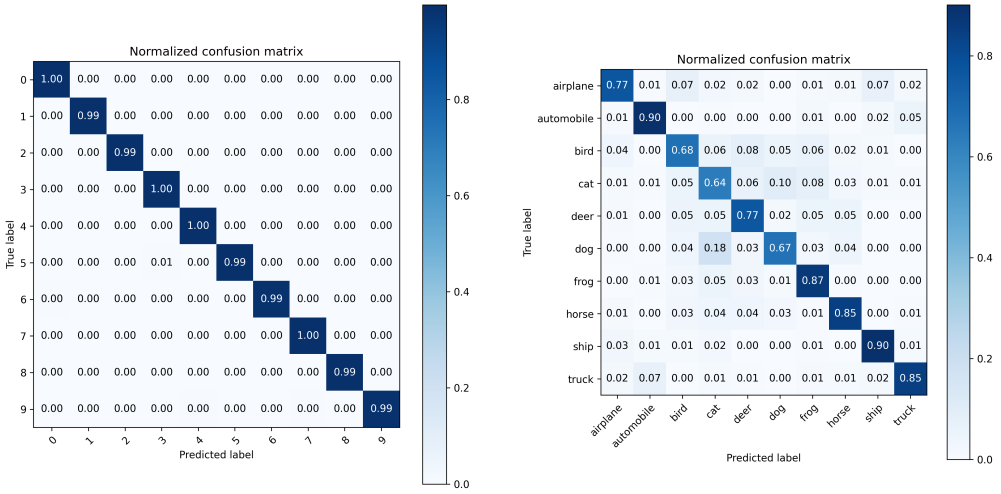


(b) VGG16 loss on CIFAR-10



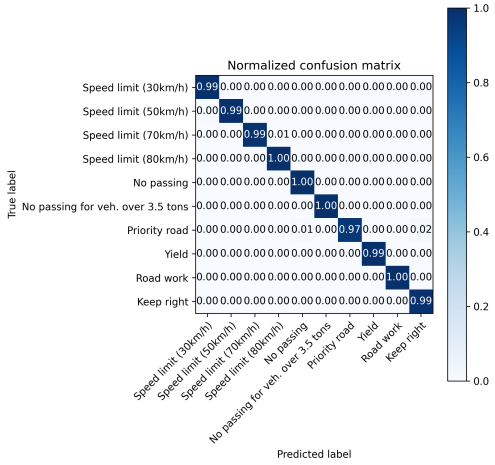
(c) VGG16 loss on GTSRB

Figure 56: VGG16 training performance on all three datasets with regard to the loss.



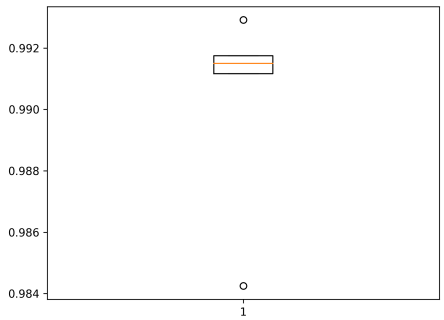
(a) VGG16 confusion matrix on MNIST with hold-out test set

(b) VGG16 confusion matrix on CIFAR-10 with hold-out test set

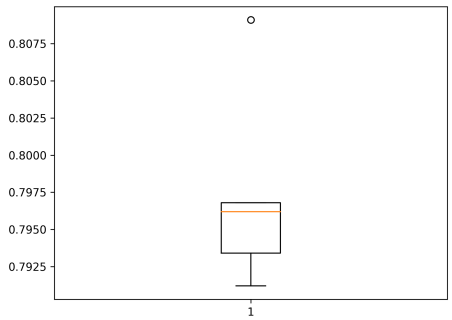


(c) VGG16 confusion matrix on GTSRB with hold-out test set

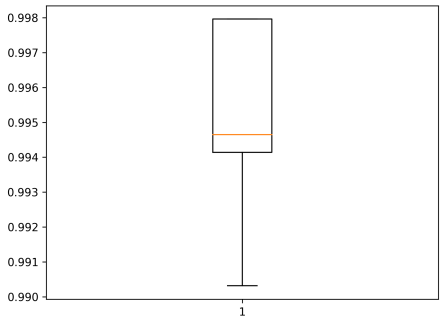
Figure 57: VGG16 confusion matrices on all three hold-out test sets.



(a) VGG16 boxplot representation of the mean and standard deviation on MNIST with validation set



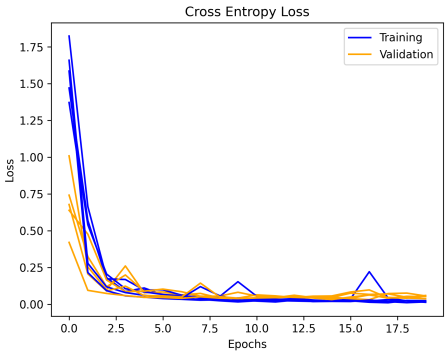
(b) VGG16 boxplot representation of the mean and standard deviation on CIFAR-10 with validation set



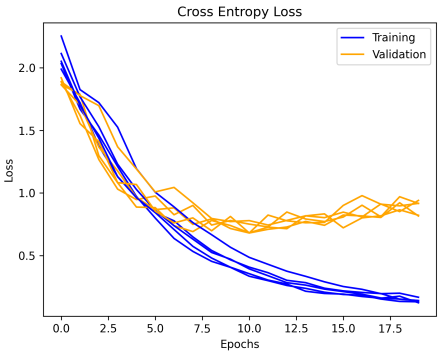
(c) VGG16 boxplot representation of the mean and standard deviation on GTSRB with validation set

Figure 58: VGG16 boxplot representations of the mean and standard deviation on all three datasets using the validation set.

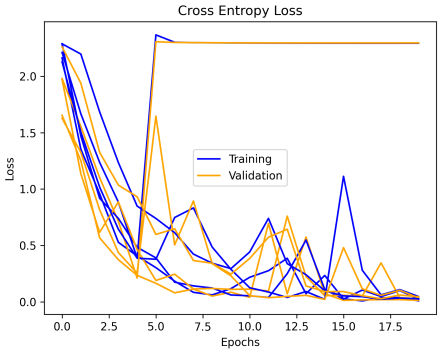
C.2 VGG19



(a) VGG19 loss on MNIST

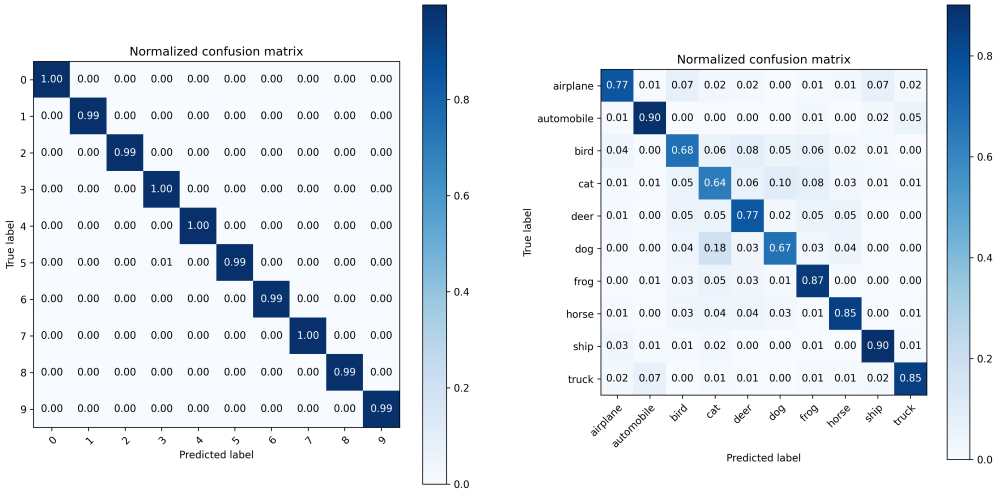


(b) VGG19 loss on CIFAR-10



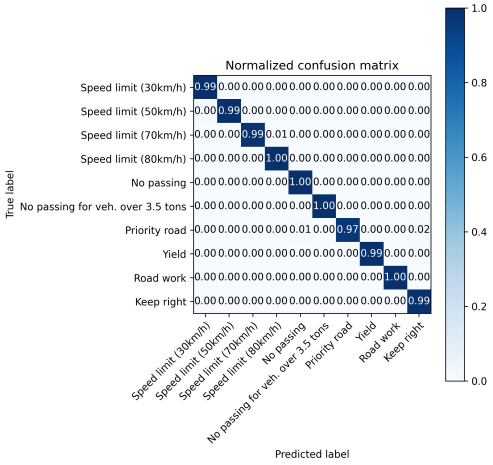
(c) VGG19 loss on GTSRB

Figure 59: VGG19 training performance on all three datasets with regard to the loss.



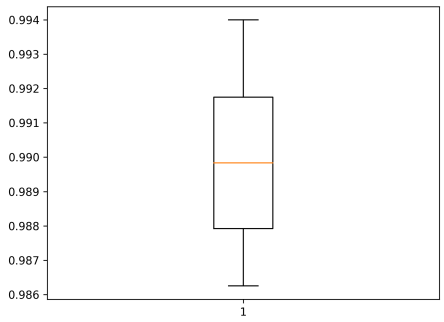
(a) VGG19 confusion matrix on MNIST with hold-out test set

(b) VGG19 confusion matrix on CIFAR-10 with hold-out test set

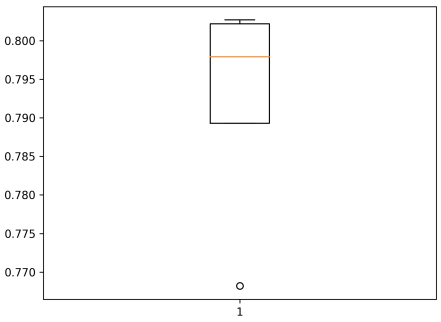


(c) VGG19 confusion matrix on GTSRB with hold-out test set

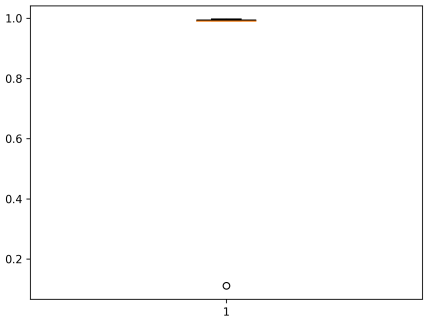
Figure 60: VGG19 confusion matrices on all three hold-out test sets.



(a) VGG19 boxplot representation of the mean and standard deviation on MNIST with validation set



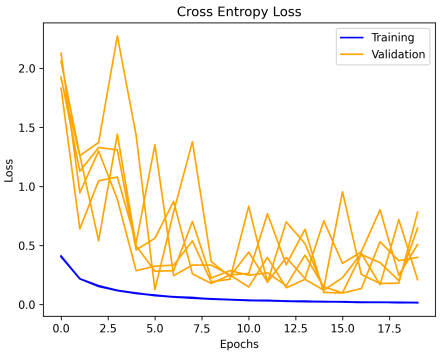
(b) VGG19 boxplot representation of the mean and standard deviation on CIFAR-10 with validation set



(c) VGG19 boxplot representation of the mean and standard deviation on GTSRB with validation set

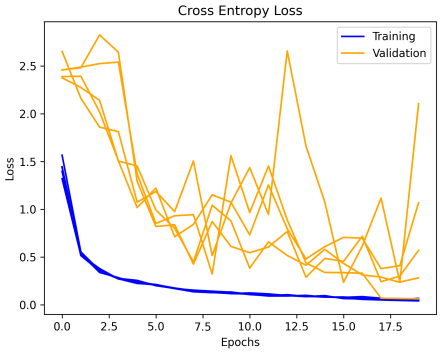
Figure 61: VGG19 boxplot representations of the mean and standard deviation on all three datasets using the validation set.

C.3 AlexNet



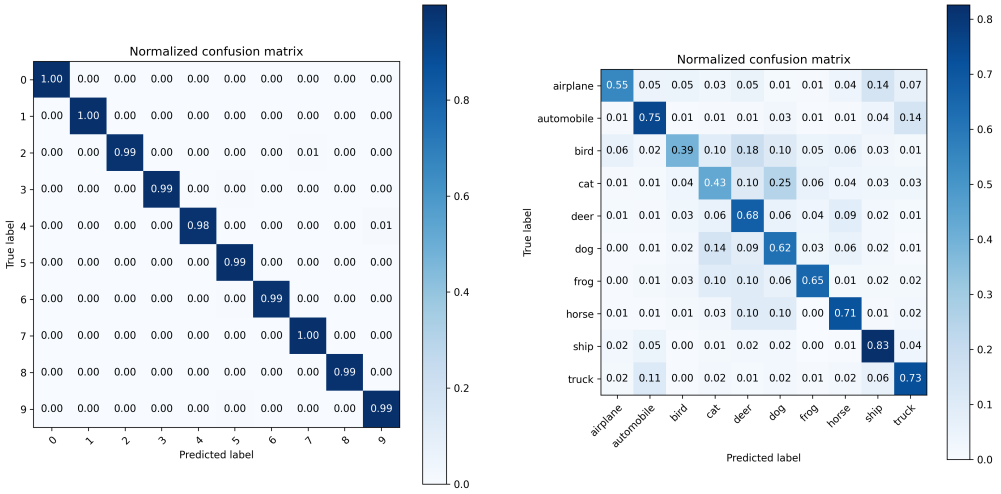
(a) AlexNet loss on MNIST

(b) AlexNet loss on CIFAR-10



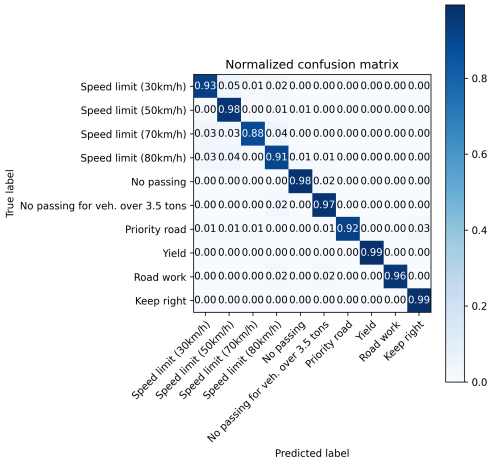
(c) AlexNet loss on GTSRB

Figure 62: AlexNet training performance on all three datasets with regard to the loss.



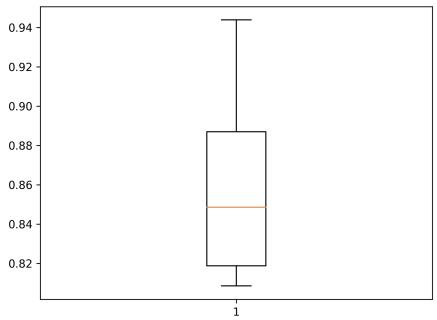
(a) AlexNet confusion matrix on MNIST with hold-out test set

(b) AlexNet confusion matrix on CIFAR-10 with hold-out test set

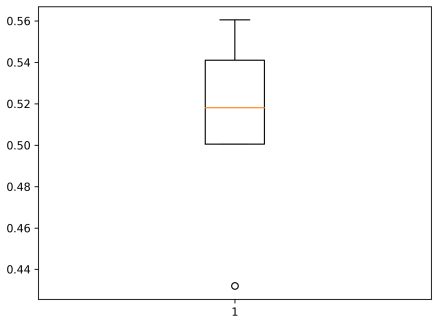


(c) AlexNet confusion matrix on GTSRB with hold-out test set

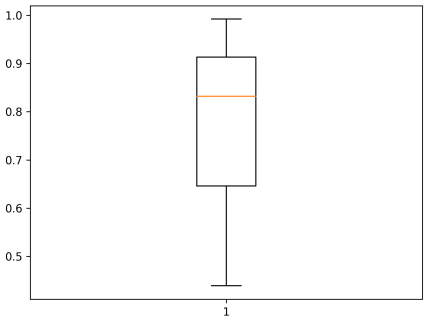
Figure 63: AlexNet confusion matrices on all three hold-out test sets.



(a) AlexNet boxplot representation of the mean and standard deviation on MNIST with validation set



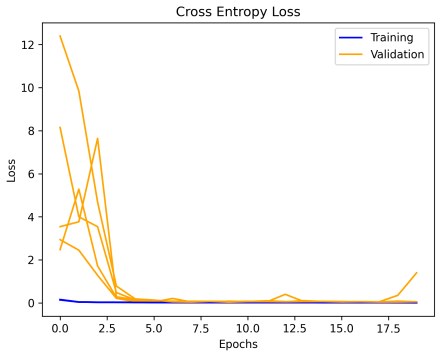
(b) AlexNet boxplot representation of the mean and standard deviation on CIFAR-10 with validation set



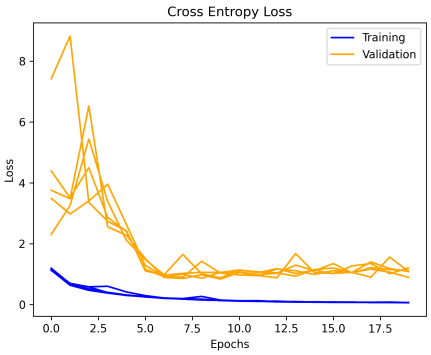
(c) AlexNet boxplot representation of the mean and standard deviation on GTSRB with validation set

Figure 64: AlexNet boxplot representations of the mean and standard deviation on all three datasets using the validation set.

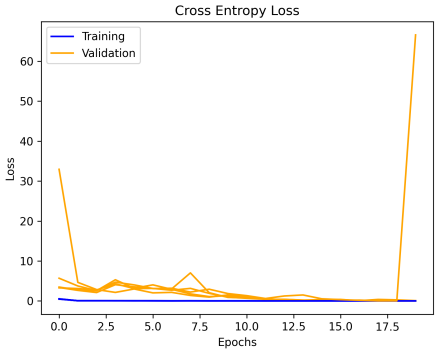
C.4 ResNet50



(a) ResNet50 loss on MNIST

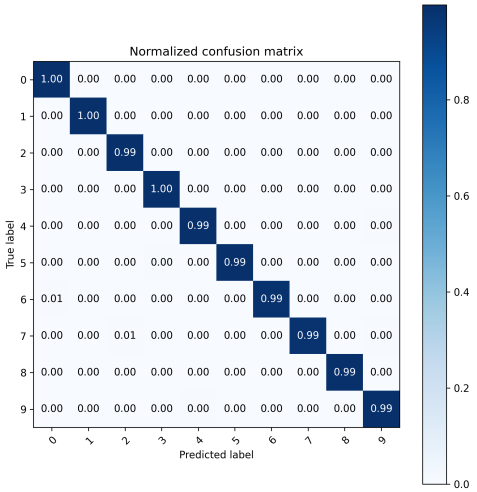


(b) ResNet50 loss on CIFAR-10

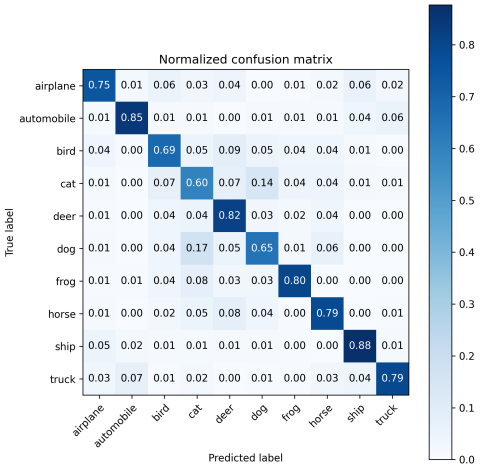


(c) ResNet50 loss on GTSRB

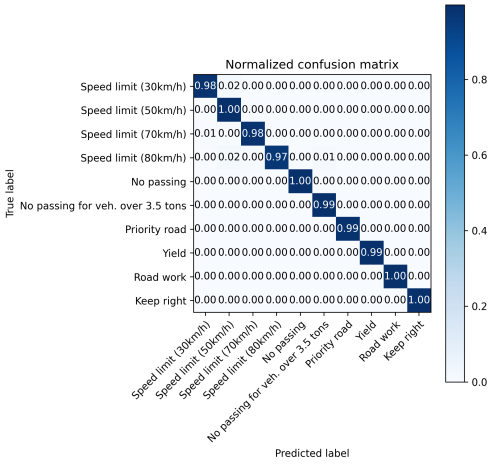
Figure 65: ResNet50 training performance on all three datasets with regard to the loss.



(a) ResNet50 confusion matrix on MNIST with hold-out test set

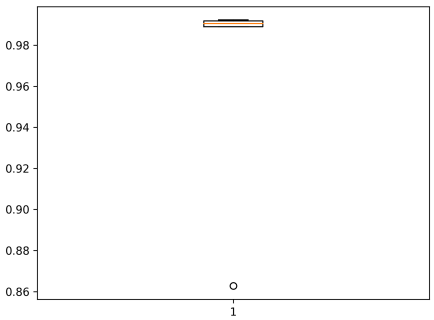


(b) ResNet50 confusion matrix on CIFAR-10 with hold-out test set

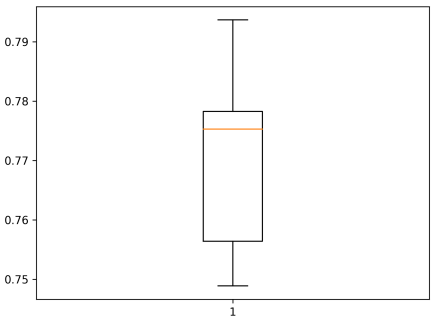


(c) ResNet50 confusion matrix on GTSRB with hold-out test set

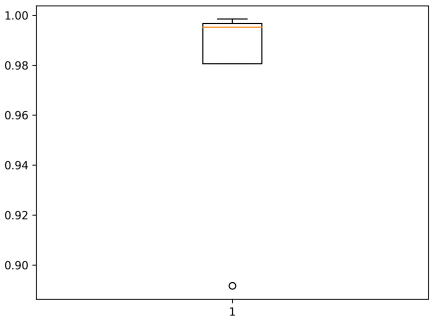
Figure 66: ResNet50 confusion matrices on all three hold-out test sets.



(a) ResNet50 boxplot representation of the mean and standard deviation on MNIST with validation set



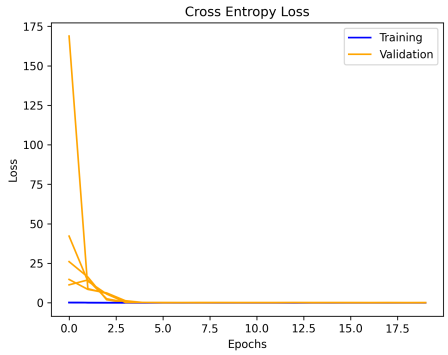
(b) ResNet50 boxplot representation of the mean and standard deviation on CIFAR-10 with validation set



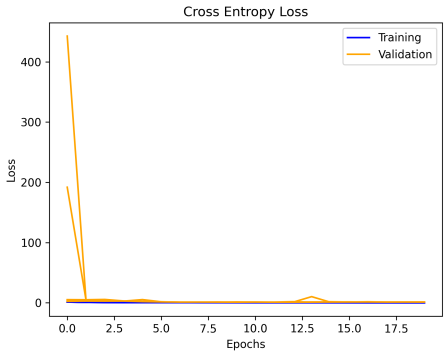
(c) ResNet50 boxplot representation of the mean and standard deviation on GTSRB with validation set

Figure 67: ResNet50 boxplot representations of the mean and standard deviation on all three datasets using the validation set.

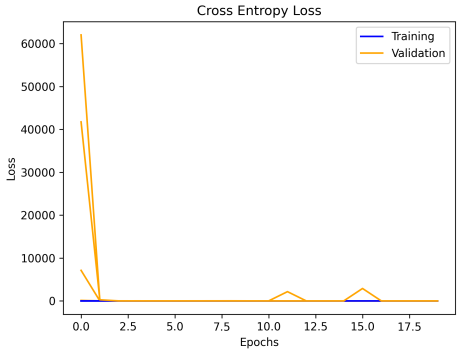
C.5 ResNet101



(a) ResNet101 loss on MNIST

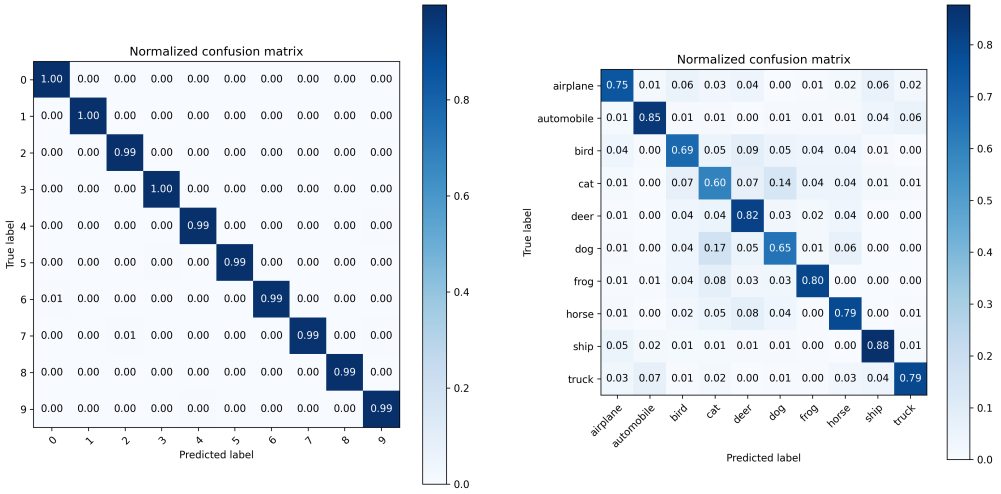


(b) ResNet101 loss on CIFAR-10



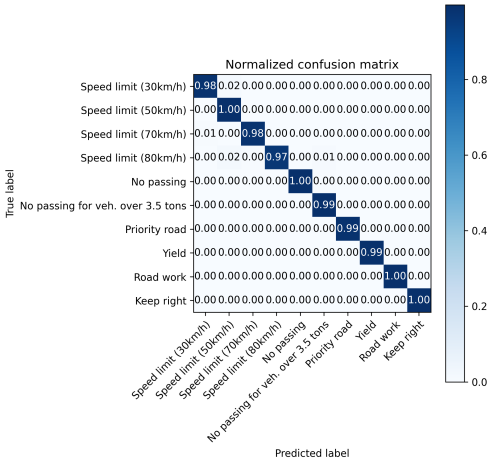
(c) ResNet101 loss on GTSRB

Figure 68: ResNet101 training performance on all three datasets with regard to the loss.



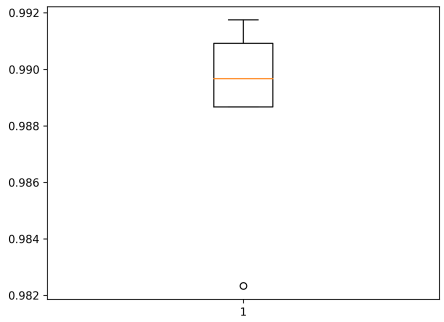
(a) ResNet101 confusion matrix on MNIST with hold-out test set

(b) ResNet101 confusion matrix on CIFAR-10 with hold-out test set

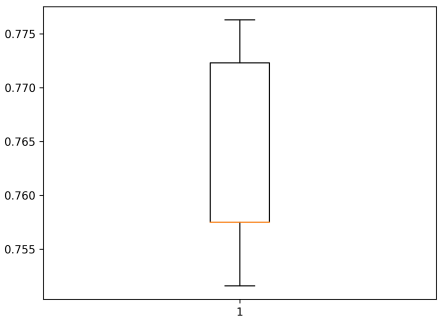


(c) ResNet101 confusion matrix on GTSRB with hold-out test set

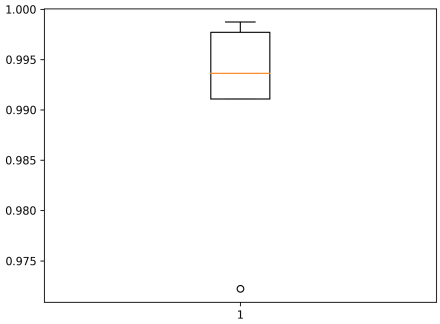
Figure 69: ResNet101 confusion matrices on all three hold-out test sets.



(a) ResNet101 boxplot representation of the mean and standard deviation on MNIST with validation set



(b) ResNet101 boxplot representation of the mean and standard deviation on CIFAR-10 with validation set



(c) ResNet101 boxplot representation of the mean and standard deviation on GTSRB with validation set

Figure 70: ResNet101 boxplot representations of the mean and standard deviation on all three datasets using the validation set.

D GTSRB Labels

Class index	Class name	Meta image
0	Speed limit (20km/h)	
1	Speed limit (30km/h)	
2	Speed limit (50km/h)	
3	Speed limit (60km/h)	
4	Speed limit (70km/h)	
5	Speed limit (80km/h)	
6	End of speed limit (80km/h)	
7	Speed limit (100km/h)	
8	Speed limit (120km/h)	
9	No passing	
10	No passing for vechiles over 3.5 metric tons	
11	Right-of-way at the next intersection	
12	Priority road	

Class index	Class name	Meta image
13	Yield	
14	Stop	
15	No vechiles	
16	Vechiles over 3.5 metric tons prohibited	
17	No entry	
18	General caution	
19	Dangerous curve to the left	
20	Dangerous curve to the right	
21	Double curve	
22	Bumpy road	
23	Slippery road	
24	Road narrows on the right	
25	Road work	
26	Traffic signals	
27	Pedestrians	
28	Children crossing	
29	Bicycles crossing	
30	Beware of ice/snow	

Class index	Class name	Meta image
31	Wild animals crossing	
32	End of all speed and passing limits	
33	Turn right ahead	
34	Turn left ahead	
35	Ahead only	
36	Go straight or right	
37	Go straight or left	
38	Keep right	
39	Keep left	
40	Roundabout mandatory	
41	End of no passing	
42	End of no passing by vechiles over 3.5 metric tons