Informatics

Man of Steal: Exploring Model Stealing Attacks against Image Classifiers

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieurin

im Rahmen des Studiums

Logic and Computation

eingereicht von

Daryna Oliynyk
Matrikelnummer 11932433

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Ao.Univ.Prof. Dipl.-Ing. Dr.techn. Andreas Rauber
Mitwirkung: Univ.Lektor Mag.rer.soc.oec. Dipl.-Ing. Rudolf Mayer

Wien, 1. Februar 2023

Daryna Oliynyk

Andreas Rauber

Informatics

Man of Steal: Exploring Model Stealing Attacks against Image Classifiers

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieurin

in

Logic and Computation

by

Daryna Oliynyk
Registration Number 11932433

to the Faculty of Informatics

at the TU Wien

Advisor: Ao.Univ.Prof. Dipl.-Ing. Dr.techn. Andreas Rauber
Assistance: Univ.Lektor Mag.rer.soc.oec. Dipl.-Ing. Rudolf Mayer

Vienna, 1st February, 2023

Daryna Oliynyk   Andreas Rauber

Erklärung zur Verfassung der Arbeit

Daryna Oliynyk

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 1. Februar 2023

Daryna Oliynyk

Acknowledgements

Accomplishing this degree would be impossible without my family. I am very grateful to my parents, who encouraged me to take this journey, supported me at every single step, and always believed in me. I am also thankful to my grandmother for her help and the inspiration she gave me through her example. I am grateful to my sister for cheering me up and reminding me how joyful life can be. And I am very thankful to my husband, who was always there for me and probably knows my topic better now than he has ever wished.

I sincerely appreciate the guidance and support of my supervisors. I am grateful to Andreas Rauber for all the discussions that we had and his ability to clarify things I have been struggling with for weeks in a few minutes. I am also very thankful to Rudolf Mayer for his incredible support, spending hours on "just 5-minute" talks with me, providing tons of feedback on my work and encouraging my further research.

Lastly, I would like to mention all the people in Ukraine who are fighting these days for freedom and their future. I admire their courage and am grateful that they reminded me to be brave and not give up.

Abstract

Machine learning models offered as a service are the most common targets for a model stealing attack that aims to reproduce a model's behaviour without its owner's consent. Such attacks lead to intellectual property violations and unfair competition, bringing more attention to the topic. This work analyses the most significant group of model stealing attacks against black-box image classifiers. We categorise relevant work based on the considered attacker's profile, and highlight inconsistencies in experiment design and attack evaluation that lead to comparability issues. Further, we conduct experiments against CNN image classifiers and investigate how different attacker's capabilities and attack optimisation techniques impact the attack's performance. In particular, we propose a novel data-free attack, which is significantly more efficient while having comparable performance with the state-of-the-art. Subsequently, we study three data-perturbation defences as countermeasures against model stealing attacks and investigate how they affect the utility of the target model. Finally, we re-visit the related work issues and propose solutions for each to ensure comparability in future work.

Contents

CHAPTER 1

Introduction

Building a well-performing machine learning solution is a complicated process that usually requires a significant amount of high-quality data, computational power and human expert knowledge. As data gathering and model training are highly time-consuming processes, finding a model configuration that will reach the desired performance could take months or even years. Considering the amount of time, resources, and expert knowledge invested in developing a machine learning model, it can be considered as intellectual property (IP) of its owner. Therefore, if the model becomes publicly available, the owner should take action to prevent intellectual property infringements. While publishing a license will define legal model exploitation, it can not prevent unauthorised usage of the model.

The problem of protecting machine learning models becomes more important with the rising popularity of Machine Learning as a Service (MLaaS), a cloud-based technology for creating and running machine learning algorithms. On the one hand, MLaaS allows users to overcome computational power limitations and train their models on a cloud. More relevant to this work, they also offer ready-to-use machine learning solutions, i.e. models that were trained to reach a certain performance score on a particular task. These models can be fully public, so anyone can download and re-use them, or they can be provided with limited access, e.g. as black boxes that allow only input-output (query) interactions. For the latter, the model owner can further reduce access to the model by setting query limits and fees. If a malicious user wants to omit those limitations and obtain an (illegitimate) copy of the model, they can launch a *model stealing attack* [TZJ+16].

Model stealing attacks aim to extract characteristics of a *target* model such as its (hyper)parameters and architecture or obtain a model that copies the behaviour of the target model. Recently, those attacks were investigated in fields like image classification [OSF19], image-to-image translation [SDGA21], natural language processing [KTP+20], and reinforcement learning [CGZ+21]. A stolen model can be used to launch a competitive service, causing unfair competition. However, model stealing can also be utilised as a preparation stage for other attacks. For instance, the majority of methods for crafting adversarial examples [SZS+14] require access to model weights, which are not available under black-box access. However, it was shown that adversarial examples crafted for a stolen model are transferable, i.e. they could also fool the target model [PMG+17]. Therefore, an adversary can launch an evasion attack against the target model by exploiting a stolen model. Consequently, the model can be compromised, and the integrity of a system that relies on that model can be affected.

The rise of model stealing attacks led to the development of countermeasures that aim to detect an ongoing attack [JSMA19], mitigate an attack [OSF20], or prove that a model was stolen [SAMA21]. However, nearly all defences make certain assumptions about an attacker, either about their knowledge or the strategy they apply. Moreover, they have limitations: if a defence can only detect an attack or prove that it was performed, the target model is not actually protected and can still be stolen; and defences that mitigate attacks usually affect the utility of the model, causing discomfort even for benign users.

A recent survey study [OMR23] showed that the current state of model stealing attacks and defences requires deeper investigation in both directions. On the one hand, there is no unified way of attack evaluation, which leads to the inability to compare different attacks using only metrics reported by corresponding papers. On the other hand, proposed defences usually rely on specific attackers' behaviour and are estimated only against a small subset of defences. Currently, there is an ongoing competition between defences and attacks: while earlier defences are mostly broken, newer ones have since been published and showed resistance to those attacks that breached the previous defences. Moreover, only a minority of papers consider an adaptive attacker, which is widely studied for other categories of attacks, for instance, evasion attacks [TCBM20].

1.1 Problem Statement and Research Questions

This work delves into the largest category of model stealing attacks, which concerns image classification models, namely convolutional neural networks. At first, it analyses current state-of-the-art and classifies relevant attacks based on the attacker profile, highlighting major non-comparability issues caused by inconsistency in attack design and evaluation. Those observations are further used for designing an experiment setup for this work, ensuring that effectiveness and efficiency metrics characterise the attack performance completely. Then it empirically compares six attack scenarios corresponding to different attacker profiles. Two scenarios represent a novel data-free attack, which turns out to be significantly more efficient than the current state-of-the-art. Finally, it explores three attack mitigation techniques and shows their utility against considered attacks.

An attack starts with constructing a dataset labelled by the target model (thus acting as an oracle). An adversary can either use original data samples used to train the target model or, if the original data is unavailable, any publicly available dataset. In an extreme data-free scenario, an attacker is assumed to have no datasets available, so the data is synthesised. The adversary can also use an advanced method for picking samples for labelling, e.g. active learning. Active learning dynamically selects samples out of a pool for labelling, so that only a subset of data is used for training. This can become crucial when there is a limit on the number of queries the adversary can send to the target model. The next step is to train a so-called *substitute model* using the data labelled by the target model, to achieve a similar performance as that target model. To make the training faster and more effective, the attacker can leverage transfer learning, i.e. use a model pre-trained on another dataset to speed up the convergence. After the training is finished, the attack needs to be evaluated and potentially adjusted to reach better performance.

The usability of an attack is measured by its performance in terms of effectiveness and efficiency. The following metrics correspond to **effectiveness** measures:

- **Accuracy** measures how well the stolen model performs on the original classification task. In other words, it measures the similarity between predictions of the stolen model and original labels.
- **Fidelity** measures similarity in predictions of the stolen and the target models. It shows to which extent they *agree* on (original) data samples, which means that they predict the same value and solve the corresponding classification task in the same way.
- **Transferability** measures how similar the decision boundaries of the target and the stolen models are. To evaluate that, one should first generate some *adversarial examples* for the stolen model and then measure how many of them can also fool the target model.

Depending on the attacker's intention, they could prioritise one of the metrics above. If the goal is to obtain the model that solves the original classification task, the accuracy of the stolen model should be high. If obtaining the same behaviour as the original model is the goal, fidelity or transferability may be more valuable.

The **efficiency** of a model stealing attack is measured using the following two metrics:

- **Query budget** corresponds to the number of queries an adversary used to perform an attack. It directly affects the performance of a substitute model, as it needs enough training data to reach the desired performance. However, since this metric does not consider the complexity of the stealing task, and more complex models generally require more training data, it can only be used to compare attacks against one particular model.
- **Efficiency score** combines query budget with the complexity of the target model. It is defined as the average number of queries needed to steal a single trained parameter of the target model. In contrast to the query budget, it allows, to some extent, comparing attacks performed against models of different complexity.

Since an owner of a model can limit the number of queries available to a user, an adversary is interested in optimising the usage of queries to perform more efficient attacks.

To reduce the success of behaviour-stealing attacks, a model owner can apply either a reactive or proactive defence. The former includes defence strategies that can either detect an ongoing attack, or, when given a stolen model, prove that it is an illegitimate copy of the target model. The latter refers to defences that aim to mitigate the performance of model stealing attacks by limiting or perturbing information revealed about the target model. Our primary focus is on proactive data perturbation defences that perturb either a data sample before the target model makes a prediction on it, or modify an output predicted for that sample.

The scope of the thesis can be summarised with the following research questions:

1. **To what degree are model stealing attacks effective?**

   We measure the utility of model stealing attacks under different assumptions regarding the attacker's capabilities and a stealing approach. To gauge the performance, we use accuracy, fidelity, and transferability for effectiveness evaluation, and query budget and efficiency score for efficiency evaluation.

   a) **To what extent does the effectiveness of attacks depend on its query budget?**

      This question aims to reveal how many queries it takes to create a substitute model with comparable performance to the target model and how the performance is changed depending on query limitations. We also aim to explore how the ratio of substitute training data to target training data correlates with the performance of the substitute model.

   b) **To what extent do the effectiveness and efficiency of attacks change depending on the complexity of the target model?**

      The complexity of the model corresponds to the number of trainable parameters of this model. Since more complex models could require more data and time to reach a certain performance, one may assume that they also need more queries to be stolen.

   c) **To what extent does the effectiveness of model stealing attacks change when the target model architecture is not known?**

      Black-box access implies that the target model architecture is unknown. An adversary then has to make a guess and can select substitute architectures more or less complex than the target architecture. We want to examine how (mis)matching the target architecture impacts the attack performance, while keeping the query budget constant.

   d) **How does the effectiveness of model stealing attacks change if the target model is trained with transfer learning?**

Transfer learning speeds up a training process and can be used by both parties. However, target and substitute models might be trained from weights obtained while training on the same dataset (e.g. ImageNet [DDS+09]). It gives additional similarity between target and substitute network behaviour even before an attack is performed. With this question, we want to verify if this is an advantage for an attacker that leads to a better-performing substitute model.

e) **To what extent does the effectiveness of model stealing attacks change depending on the availability of data?**

We discern four attack cases, depending on the availability of data for an adversary. The strongest assumption corresponds to the case when the original data used for target model training is available. A weaker assumption is the availability of problem-domain data, which preserves features of the original data but might come from a different distribution. The case of non-problem domain means an attacker that can only collect unrelated data of the same modality. The weakest overall assumption stands for the data-free case, when no data is available, and a substitute model has to be trained on artificially generated data. This question aims to compare (some of) the listed scenarios to examine the advantage an adversary gets with higher availability of data.

f) **To what extent does query optimisation of model stealing attacks improve their effectiveness?**

The owner of a target model can put severe limitations on the attacker's query budget. Hence, we want to explore which techniques the attacker can use to make an attack more effective, preserving the same efficiency score, and to which extent those techniques are helpful.

2. **To what degree are data perturbation defences against model stealing attacks useful?**

Various defence approaches against model stealing have been proposed. However, many of them can not mitigate an attack, but rather only state that an attack is happening or took place in the past. We aim to explore a category of defences that aim to mitigate attacks by trying to reduce the performance of a substitute model. Namely, we look at data perturbation defences, which modify inputs or outputs of the target model, with the intention to add noise to the information an adversary obtains.

a) **To what extent does the effectiveness of model stealing attacks decrease when a certain defence is applied?**

The main goal of a defence is to reduce the performance of a substitute model. We aim to explore if the effectiveness of an attack changes while the query budget remains fixed. Moreover, we want to find out if any performance metrics (accuracy, fidelity, transferability) are more sensitive than others to an applied defence.

b) **To what extent does the utility of the target model change when a data perturbation defence is applied?**

Applying a defence that adds noise to predictions can harm the performance of the target model. Hence, we want to measure how the performance changes when the inputs or outputs of the target model are modified.

c) **To what extent does the utility of defences change when attacker's knowledge about the target model is limited?**

Within this research question, we aim to examine if there is any difference in defence utility when an attacker meets the limitations considered in Question 1. We assume that some defences that are ineffective against a stronger attacker (who has more knowledge) can still be effective against a weaker one (who has less knowledge).

1.2 Methodology

1.2.1 Literature Review

The literature review is based on the restricted version of Kitchenham's guidelines [KPB+09] applied to the recent survey paper on model stealing attacks and defences [OMR23]. It summarises more than 100 related works, including recent articles published in 2022. We use the survey to get an overview of the attacks and defence approaches and select papers relevant to image classification models for further in-depth analysis. The review process is organised as follows.

1. Gather information about attack techniques applicable to image classification models;
2. Select state-of-the-art papers that exploit those techniques;
3. Analyse threat models, attack varieties and experimental setup of the selected papers;
4. Gather information about defence techniques;
5. Select and review state-of-the-art papers focusing on attack performance mitigation.

Additionally, we analyse some papers on crafting adversarial examples as they can be utilised for model stealing attacks [OMR23].

1.2.2 Threat Modelling

Threat modelling is an essential step in security analysis. It describes the attacker's incentives, goals, and capabilities [BR18]. The incentives and goals remain unchanged within the scope of this thesis and match the ones considered in behaviour-stealing papers.

However, the attacker's capabilities, e.g. knowledge about the target model, might differ significantly. We define a threat model for each scenario considered in this thesis, aligning it with previous work.

1.2.3 Experiment Design and Evaluation

Following the CRISP-DM process [WH00], the subsequent steps are selected for conducting the experimental part of this thesis and evaluating its outcomes.

Design and implementation of target model training
Designing a target model training process is necessary to imitate ownership of a model and develop countermeasures. We base our architecture and training dataset choice on the papers selected in Step 3 during the literature review to ensure comparability with related work.

Design and implementation of model stealing attacks
During threat modelling, we define the attacker's capabilities, e.g. the knowledge about the target model or its training data. At this stage, we investigate advanced techniques that could potentially improve an attack's performance. For instance, an adversary can use different query optimisation techniques to reduce the number of queries needed. Then we configure attack settings comprising a dataset used for querying the target model, an architecture used for substitute model, query budgets, etc.

Evaluation of model stealing attacks
Each implemented attack is evaluated using three effectiveness metrics: accuracy, fidelity, and transferability; and two efficiency metrics: the number of queries and the efficiency score. Based on the evaluation, one can conclude how the attacker's knowledge and applied attack strategy affect attack performance.

Design and implementation of defences
The choice of defence strategies is based on their ability to protect against behaviour-stealing attacks. For instance, if a defence was created to protect the target model architecture, it might not be applicable for behaviour protection. At the same time, some defences designed to protect tabular data classifiers can be adapted to other data domains, e.g. for image data considered in this thesis.

Evaluation of defences
The main evaluation criteria of countermeasures is the capability to decrease the attack performance. A defence can decrease the effectiveness of an attack by reducing its accuracy, fidelity, and transferability, or make the attack less efficient by forcing an attacker to use more queries. Another important aspect is that applying the defence should not decrease the utility of the target model more than acceptable.

Analysis and comparison of attacks and defences
For every considered attack and defence, their effectiveness and efficiency are measured as described in the research questions. Then, we explore the impact of attack and defence

settings on the attack performance. Finally, we compare the results of this thesis with the state-of-the-art.

1.3 Structure of the Work

The rest of the thesis is structured as follows. Chapter 2 provides general information about machine learning, which is necessary for understanding the scope of the thesis. Then, Chapter 3 describes state-of-the-art attack and defence approaches, highlighting the category of attacks focused on image classification stealing. Chapter 4 describes in detail the attack scenarios implemented in this work, followed by a description of the considered countermeasures Chapter 5. Chapter 6 details the experimental setup, while Chapter 7 summarises and analyses the main results of experiments. Finally, Chapter 8 concludes the work by summarising the contributions, answering the research questions, and providing directions for future work.

CHAPTER 2

Background

This chapter provides the necessary background for understanding the rest of the thesis. We explain the basics of Machine Learning, focusing on its application for image classification tasks. At first, we focus on simpler (shallow) models to illustrate relevant learning concepts. Then we switch to Deep Learning, covering architectures used in this thesis. We conclude the chapter with an overview of adversarial machine learning.

2.1 Machine Learning for Image Classification

Machine Learning is a powerful tool that has been applied to solve a vast diversity of problems, from spam email detection to autonomous driving. As this thesis focuses on one particular problem, namely image classification, most of the terminology and explanations are narrowed down to this topic.

2.1.1 Dataset

A *dataset* consists of a set of pairs (x, y), where $x \in \mathbb{R}^n$ is an image and $y \in C = \{c_1, c_2, \dots, c_k\}$ is a corresponding label. Figure 2.1 depicts samples from MNIST [Den12], one of the most common image datasets for digit classification problem. In this dataset, x is a grayscale image of size 28×28, and y can be any of ten digits.

A dataset defines the task that a machine learning algorithm should solve. Usually, the data is split into three parts which are training, validation, and test set. The *training set* is used for creation of a machine learning solution. The *validation set* is used to evaluate different hyperparameter configurations and select the most feasible solution. The *test set* is used for final evaluation, and, in particular, for comparing results with the dataset benchmarks.

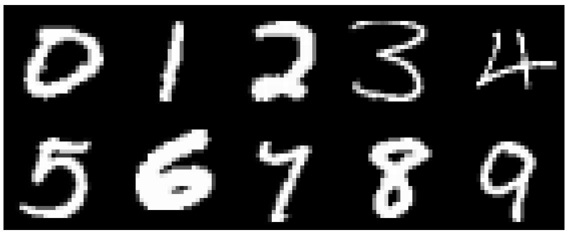


Figure 2.1: MNIST dataset [Den12].

2.1.2 Data Preprocessing

Data preprocessing includes diverse techniques that aim to prepare data for future use. This work only focuses on image data, so we describe typical image preprocessing methods below.

1. *Normalisation*, also known as *min-max scaling*, re-scales each input feature $x^{(f)}$ such that it lays within the range $[0, 1]$ by applying the following transformation

$$x^{(f)}_{normalised} = \frac{x^{(f)} - x^{(f)}_{min}}{x^{(f)}_{max} - x^{(f)}_{min}} \tag{2.1}$$

   Here, $x^{(f)}_{min}$ stands for the minimum value of feature $x^{(f)}$ in the considered (training) dataset, and $x^{(f)}_{max}$ for the maximum value.

   Since pixel minimum and maximum values for each RGB channel are 0 and 255 correspondingly, image normalisation is done by dividing each pixel value by 255.

2. *Standardisation* is another data transformation technique that makes mean values of features to be 0 and standard deviation values to be 1. For a feature $x^{(f)}$ its mean $x^{(f)}_{mean}$ and standard deviation $x^{(f)}_{std}$ are calculated using the training data. Then, the following transformation is applied:

$$x^{(f)}_{standardised} = \frac{x^{(f)} - x^{(f)}_{mean}}{x^{(f)}_{std}} \tag{2.2}$$

   For image data, standardisation is performed channel-wise for each pixel. We use image standardisation based on the recommendation of LeCun et al. [LBOM12], who suggest to use scaled data samples with zero mean to improve the convergence of a model.

3. *Resizing* is an image-specific data preprocessing step, which consists in changing the width and height of an image. It is mainly used to ensure (by downsampling or upscaling) that all samples in a dataset have the same size and can fit into a model. In particular, resizing may lead to a changed aspect ratio of some images.

2.1.3 Data Augmentation

To increase the variety of available data without gathering new data samples, *data augmentation* is used. It includes diverse data modification techniques, which preserve the content (and hence the associated label) of a data sample, but generate new samples with different input features. We list some image augmentation techniques below.

1. *Rotation* of an input image by a specified angle. This transformation should be done carefully for images with distinctive bottom and top parts not to ruin spatial features.
2. *Flipping* of an image horizontally or vertically. Similarly to rotation, in some cases should be avoided. For instance, a vertically flipped face is much harder to recognise than the original unflipped version.
3. *Zooming* an image in or out while preserving its actual size.
4. *Shifting* of an image on axes x and y, while preserving its actual size.

2.1.4 Machine Learning Model

A *classification machine learning model* is a function $f : \mathbb{R}^n \to C = \{c_1, c_2, \ldots, c_k\}$ that maps each data sample $x \in \mathbb{R}^n$ into a class $c \in C$. A parameter-based model $f = f_w$ is dependent on some parameters w that are optimised in order to fit the task better. The process of finding optimal values for w from a training set $(x_i, y_i), i \in \{1, \ldots, n\}$ is called *training* of the model. The process of making a prediction $f_w(x) = \hat{y}$ on an unseen sample x is called *inference*.

2.1.5 Shallow Machine Learning

Shallow machine learning includes models that learn to predict labels for data represented through pre-defined features. However, for non-tabular data, a *feature extraction*, which aims to extract numerical meaningful characteristics from data, is needed. Below we briefly describe some shallow machine learning approaches for classification problems.

- *Logistic Regression* (LG) is a binary-classification model that predicts the probability of a sample belonging to a (positive) class. For a sigmoid function σ:

$$\sigma(z) = \frac{1}{1 + e^{-z}} \tag{2.3}$$

  and an input sample $x = (x_1, x_2, \ldots, x_n)$, logistic regression is defined as

$$f_w(x) = \sigma(w_1 x_1 + w_2 x_2 + \ldots + w_n x_x + b) = \sigma(w^T x + b) \tag{2.4}$$

  Here $w = (w_1, w_2, \ldots, w_n)$ and b are trainable parameters of the model, also called weights (w) and bias (b). Generalised for multiclass classification problems, the algorithm is called Multinomial (or Multiclass) Logistic Regression.

- *Decision Tree* (DT) is a non-parametric tree-based model that consists of internal (decision) nodes and leaves. Each internal node corresponds to a specific rule which defines a data split. Each split corresponds to tree branches that lead to another internal node or leaf. Leaf nodes correspond to predictions, i.e., all samples in the same leaf belong to the same class.
- An ensemble of decision trees is called *Random Forest* (RF).
- *Support Vector Machine* (SVM) is a maximum-margin-based approach for binary classification. It aims to create a hyperplane that separates positive and negative classes with the largest possible distance to both classes. Like Logistic Regression, Support Vector Machines can be generalised for multiclass classification problems.

2.1.6 Fully-connected Feedforward Neural Networks

One of the most widespread machine learning models is a *neural network*. It consists of *nodes* organised into *layers*, such that nodes from one layer are connected to nodes from other layers. The first layer is called *input layer*. Each node of the input layer corresponds to one feature of a sample x. In Figure 2.2, the input layer consists of two nodes, which correspond to features x_1 and x_2. The last layer is called *output layer*. The number of nodes in the output layer corresponds to the number of classes. In the example, we have three outputs y_1, y_2, and y_3. All layers between input and output are called *hidden layers*. A network with more than one hidden layer is called a *deep neural network*. In a *feedforward* neural network, nodes are only connected to nodes from successive layers. Each node in a *fully-connected* feedforward neural network is connected to all nodes in the following layer. A fully-connected feedforward neural network is also called Multilayer Perceptron (MLP) [Ros58].

We now explain how an MLP makes predictions using the example from Figure 2.2. The network has two inputs: x_1 and x_2. Each of them is connected to each node in the first hidden layer. Connections from the inputs to the first node correspond to a pair of weights (w_1, w_2) and a bias b. With this, each node computes a value $z = w_1x_1 + w_2x_2 + b$. Then a so-called *activation function* is applied to z, for instance, a Rectified Linear Unit (ReLU) function (Equation (2.5)).

$$ReLU(z) = \max(z, 0) \tag{2.5}$$

The primary purpose of an activation function is to bring non-linearity into a neural network, making an approximation of complex non-linear functions possible. ReLU also decides if a current node should impact the final decision (if the value of z is smaller than 0, the node outputs 0, making no contribution to the further inference). The sigmoid function, mentioned as the function used in logistic regression, (Equation (2.3)) is another example of an activation function used in neural networks. Hence, a computation within one node with a sigmoid activation function looks just like Logistic Regression (Equation (2.4)).

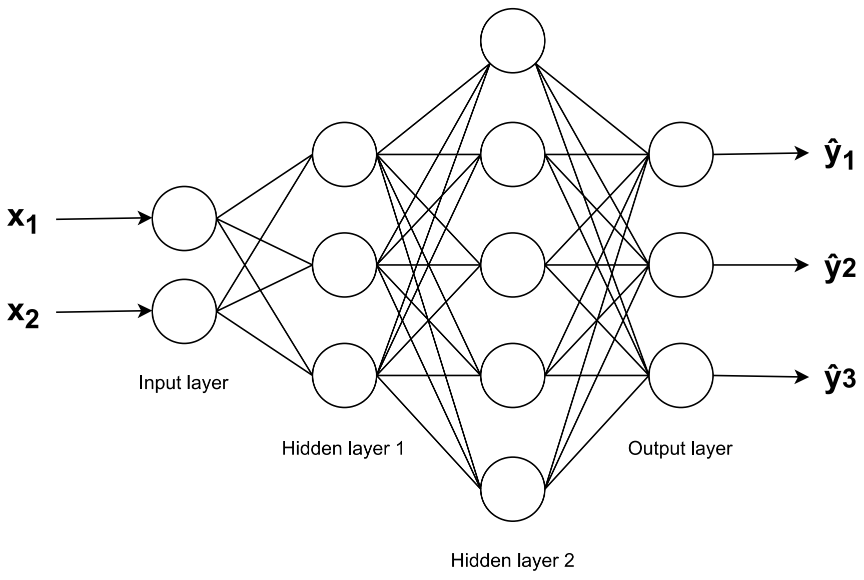


Figure 2.2: A feed-forward fully-connected neural network.

Above, we described the computation that happens in a single node, and now we generalise it for an arbitrary MLP that solves a multiclass classification problem. We consider an n_0-dimensional input $x = (x_1, \ldots, x_{n_0})^T$ that is passed through the MLP to obtain a k-dimensional output $\hat{y} = (\hat{y}_1, \ldots, \hat{y}_k)^T$, where $\hat{y}_i$ corresponds to a probability of x belonging to a class i. This process is called *forward propagation*. Each hidden layer $l \in \{1, \ldots, d\}$, where d stands for the total number of hidden layers, consists of n_l neurons. For the i^{th} neuron of the l^{th} layer, we denote as $z_i^{[l]}$ a weighted sum that comes into the neuron and as $a_i^{[l]}$ an activation which the neuron outputs. We define $z_i^{[l]}$ in Equation (2.6) and $a_i^{[l]}$ in Equation (2.7). For convenience, we set $a_i^{[0]} = x_i$, $i \in \{1, \ldots, n_0\}$ and $a_i^{[d+1]} = \hat{y}_i$, $i \in \{1, \ldots, k\}$.

$$z_i^{[l]} = (w_i^{[l]T} a^{[l-1]} + b_i^{[l]}), \tag{2.6}$$

where $l \in \{1, \ldots, d+1\}$, $w_i^{[l]} = \begin{pmatrix} w_{i1}^{[l]} \\ \vdots \\ w_{in_{l-1}}^{[l]} \end{pmatrix}$ and $b_i^{[l]}$ are weights and biases that correspond to network connections leading from layer $l-1$ to node i,

$a^{[l-1]} = \begin{pmatrix} a_1^{[l-1]} \\ \vdots \\ a_{n_{l-1}}^{[l-1]} \end{pmatrix}$ is the vector of neuron outputs of the $(l-1)^{\text{th}}$ layer.

The activation of a neuron is then calculated as follows:

$$a_i^{[l]} = g(z_i^{[l]}), \tag{2.7}$$

where g is an activation function, for instance, *ReLU* (see Equation (2.5)). For the output layer, a typical activation function for multiclass classification is a softmax function, defined as:

$$\text{Softmax}(z^{[d+1]})_i = \frac{e^{z_i^{[d+1]}}}{\sum\limits_{j=1}^{n_l} e^{z_j^{[d+1]}}},\ i \in \{1, \ldots, k\} \tag{2.8}$$

The outputs of softmax activation $\hat{y}_i = \text{Softmax}(z^{[d+1]})_i,\ i = 1, \ldots, k$ correspond to probabilities that a given input sample belongs to corresponding classes $1, \ldots, k$.

2.1.7 Loss and Cost Functions

A *loss function* L defines an error between the ground-truth label y of a sample x and the corresponding prediction of a model $\hat{y} = f_w(x)$. A *cost function* C measures accumulated error over the whole dataset. During the training process, parameters w are optimised such that the value of C is minimised, and, in the optimal scenario, is equal to 0.

One of the most common loss functions for classification problems is *cross-entropy*, which measures the similarity between two probability distributions. For each sample x with its associated ground-truth label y, we can consider its ground-truth label distribution defined by probability function $p(c_i)$, $i \in \{1, \ldots, k\}$ as follows:

$$p(c_i) = \begin{cases} 1, \text{if } c_i = y \\ 0, \text{otherwise.} \end{cases} \tag{2.9}$$

Then the predicted probability function $\hat{p}(\cdot)$ is defined by outputs of the model, for instance, by softmax outputs of the last layer of a neural network, as was discussed above:

$$\hat{p}(c_i) = \hat{y}_i \tag{2.10}$$

The cross-entropy loss for sample x and its ground-truth label y is calculated as

$$L(x, y) = -\sum_{i=1}^{k} (p(c_i) log(\hat{y}_i) \tag{2.11}$$

If a model predicts a correct class with probability 1, the loss equals 0. The corresponding cost function calculated on the whole dataset $X = (x_1, \ldots, x_n)$ with labels $Y = (y_1, \ldots, y_n)$ is then defined as the average loss value of the dataset (Equation (2.12)).

$$C(X, Y) = \frac{1}{n} \sum_{i=1}^{n} L(x_i, y_i) \tag{2.12}$$

2.1.8 Gradient Descent

The process of optimising parameters of a machine learning model, also referred to as model training, is performed using an optimisation algorithm called *gradient descent*. It aims to gradually minimise a cost function until its value reaches the global minimum. After computing the cost function (Equation (2.12)), its gradient is calculated for each parameter of the model. The gradient consists of partial derivatives of the cost function, calculated for each of the learned parameters of the model. The parameters of the model are changed towards opposite to the gradient direction, by a predefined magnitude called the *learning rate*, denoted as α. The calculation is done gradually from the last layer towards the input layer, which is why this process is called *backward propagation*. For the cross-entropy cost function, we have the following dependency from the model's parameters:

$$C(w) = \frac{1}{m}\sum_{i=1}^{m} L(x_i, y_i) = \frac{1}{m}\sum_{i=1}^{m}(-\sum_{j=1}^{k}(p(c_j)log(f_w(x_i)_j)) \tag{2.13}$$

Then for each parameter w of model f, its value is changed as shown in Equation (2.14). Such an update happens for each step of gradient descent. One training cycle through the whole dataset is called an *epoch*.

$$w = w - \alpha\frac{\partial}{\partial w}C(w) \tag{2.14}$$

Depending on the number of samples used for updating the model's parameters, there are three variations of gradient descent.

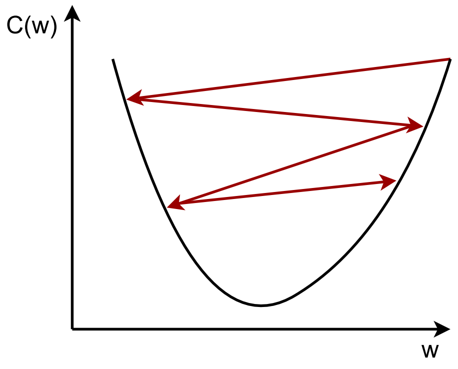
1. *(Batch) Gradient Descent* ((B)GD) uses the whole dataset for a single parameters update. In other words, one epoch corresponds to one gradient descent step. This approach gives the most precise direction of cost function optimisation, since it considers all prediction errors on each step. At the same time, it is the slowest approach and, for this reason, is usually not applicable for training image classifiers. In Equation (2.13), BGD corresponds to the case when $m = n$, i.e. the size of the training dataset.

2. *Stochastic Gradient Descent* (SGD) makes an optimisation step for each data sample. SGD is the most efficient and the least precise approach, which requires carefully selected learning rate to converge. It corresponds to $m = 1$ in Equation (2.13). Thus, during one epoch, SGD makes as many steps as there are samples in the train set.

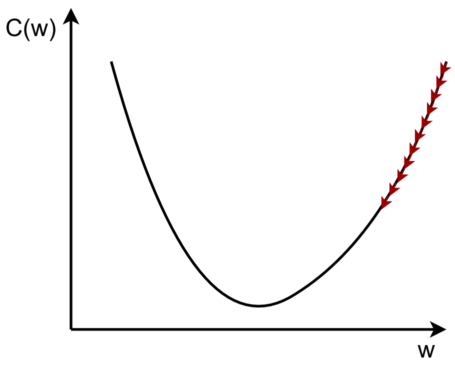
3. *Mini-batch Stochastic Gradient Descent* takes a mini-batch of samples to make a single optimisation step. Its size b is called the *(mini-)batch size* and usually equals a power of 2, e.g. 16, 32, 64, to align with memory storage on GPUs. Mini-batch SGD combines the advantages of the two approaches above: it is significantly faster

than BGD, but more precise than SGD. In Equation (2.13), it corresponds to $m = b$. In one epoch, mini-batch SGD makes $\lfloor \frac{n}{b} \rfloor$ steps, where n is the size of the train set. If n is not divisible by b, the last batch can either be smaller, resulting in one additional step, or it can be skipped.

In this work, we utilise mini-batch SGD as the most prominent approach for image classifiers. However, even being more stable than regular SGD, the mini-batch size is usually relatively small compared to the dataset size, which leads to a certain degree of stochasticity. Another challenge is picking a proper learning rate α. Large values can result in stepping over and non-ability to land into a global (local) minimum as shown in Figure 2.3a, whereas a small learning rate can make a convergence process unreasonably slow (Figure 2.3b). Besides that, different parameters of a model might require different magnitudes of change, which can not be achieved with a constant learning rate value.



(a) Large learning rate.



(b) Tiny learning rate.

Figure 2.3: Optimising cost function with different learning rates. With a large learning rate, gradient descent steps over the optimal value, whereas with a tiny learning rate the optimisation process is very slow.

In the rest of the work, we use the term gradient descent as synonym for mini-batch stochastic gradient descent.

2.1.9 Advanced Optimisation Algorithms

Different optimisation algorithms can be implemented to improve the convergence of gradient descent. We describe only the ones relevant to this work; more examples and explanations can be found in e.g. in the overview paper by Ruder [Rud16]. For convenience, we denote a modification done at the t^th step of gradient descent as δ_t. Hence, parameters update can be written as follows:

$$w = w - \delta_t, \tag{2.15}$$

where

$$\delta_t = \alpha \frac{\partial}{\partial w} C(w). \tag{2.16}$$

In the following part, we discuss how δ_t can be modified to improve the convergence.

1. *Momentum* modifies the parameters of a model, taking into account the update from the previous gradient descent step. It prevents model training from "forgetting" the direction it has moved previously and the magnitude of the last step taken. Momentum is implemented by adding an additional component to δ_t:

$$\delta_t = \beta\delta_{t-1} + \alpha\frac{\partial}{\partial w}C(w) \tag{2.17}$$

   Here β is the momentum term that controls the influence of the previous step δ_{t-1}. The most common value for β is 0.9.

2. *Adaptive Moment Estimation* (*Adam*) addresses the need for different model parameters to be updated with different magnitudes. It uses estimations of the first and the second moments of the gradients, m_t and υ_t respectively, at each step t:

$$m_t = \beta_1 m_{t-1} + (1-\beta_1)\frac{\partial}{\partial w}C(w) \tag{2.18}$$

$$\upsilon_t = \beta_2\upsilon_{t-1} + (1-\beta_2)(\frac{\partial}{\partial w}C(w))^2 \tag{2.19}$$

   Here β_1 and β_2 are decay parameters responsible for the computation of running averages. Usually, values 0.9 and 0.999 are used for them, respectively. It was spotted that calculated in such a way moments are biased towards 0. Hence they were corrected as shown in Equation (2.20) and Equation (2.21).

$$\hat{m}_t = \frac{m_t}{1-\beta_1^t} \tag{2.20}$$

$$\hat{\upsilon}_t = \frac{\upsilon_t}{1-\beta_2^t} \tag{2.21}$$

   The final update of weights combines both bias-corrected moments $\hat{m}_t$ and $\hat{\upsilon}_t$, learning rate α and some small parameter ε, used for numerical stability:

$$\delta_t = \frac{\alpha}{\sqrt{\hat{\upsilon}_t} + \varepsilon}\hat{m}_t \tag{2.22}$$

2.1.10 Training Hyperparameter Optimisation

The optimal training process depends on model characteristics and the training dataset. To find the optimal model parameters, such hyperparameters as e.g. learning rate, mini-batch size, and optimisation algorithm must be tuned. Besides, different data preprocessing or data augmentation techniques can be tested to find the most promising combination. In this thesis, the following procedure is used for hyperparameter tuning.

The training hyperparameters, along with their possible values, form a grid representing all possible hyperparameter combinations. Those combinations are then evaluated on a validation set, and the best-performing combination of hyperparameter values is selected. The process of going through the whole grid, trying every single combination, is called *(full) grid search*. Another method to perform the hyperparameter search is to try random combinations from the grid. This approach is called *random search*.

2.1.11 Evaluation

As mentioned above, evaluation can be done on two parts of a dataset: validation and test sets. The former is used for determining the most optimal hyperparameter configurations, and the latter is done for the final performance evaluation to compare the model with, e.g., a state-of-the-art solution. The performance of each system can be characterised by two criteria: effectiveness and efficiency. Effectiveness is usually represented by metrics that measure the predictive power of a model, whereas efficiency corresponds to, for example, execution time, i.e. the training time of a model and time required for inference, or other computational aspects, such as memory requirements.

We now describe in detail how the effectiveness of classification models is usually evaluated. We consider a dataset $(x, y)_i$, $i \in \{1, \ldots, n\}$ of size n as validation or test set, on which we measure the performance of model f.

- True positive TP_j shows how many samples of class c_j, $j \in \{1, \ldots, k\}$ are correctly classified by f:

$$TP_j = \sum_{i=1}^{n} \mathbb{1}_{(f(x_i)=c_j \wedge y_i=c_j)} \tag{2.23}$$

- False positive FP_j shows how many samples of the dataset are misclassified by f as class c_j:

$$FP_j = \sum_{i=1}^{n} \mathbb{1}_{(f(x_i)=c_j \wedge y_i \neq c_j)} \tag{2.24}$$

- True negative TN_j shows how many samples of the dataset that do not belong to class c_j are correctly classified by f:

$$TN_j = \sum_{i=1}^{n} \mathbb{1}_{(f(x_i)=y_i \wedge y_i \neq c_j)} \tag{2.25}$$

- False negative FN_j shows how many samples of class c_j are misclassified by f:

$$FN_j = \sum_{i=1}^{n} \mathbb{1}_{((f(x_i) \neq y_i \wedge y_i=c_j))} \tag{2.26}$$

- Precision P_j shows how many samples classified as c_j belong to that class:

$$P_j = \frac{TP_j}{TP_j + FP_j} \tag{2.27}$$

- Recall R_j shows how many samples from class c_j are correctly classified:

$$R_j = \frac{TP_j}{TP_j + FN_j} \tag{2.28}$$

- F1-Score $F1_j$ stands for the harmonic mean of precision and recall:

$$F1_j = \frac{2P_jR_j}{P_j + R_j} \tag{2.29}$$

There are two ways in which precision, recall, and f1-score can be aggregated for the whole dataset: micro- and macro-averaging. A micro-averaged metric considers contributions of true (false) positives (negatives) from each class while computing the total score. For instance, micro-averaged precision is then calculated as follows:

$$P_{micro} = \frac{\sum\limits_{j=1}^{k} TP_j}{\sum\limits_{j=1}^{k} TP_j + \sum\limits_{j=1}^{k} FP_j} \tag{2.30}$$

A macro-averaged metric is calculated as the mean value of metric scores per class. For precision, we have the following:

$$P_{macro} = \frac{1}{k}\sum_{j=1}^{k} P_j = \frac{1}{k}\sum_{j=1}^{k} \frac{TP_j}{TP_j + FP_j} \tag{2.31}$$

The above-mentioned metrics should be prioritised if a dataset is imbalanced, i.e. the amount of samples in each class is unequal. However, the datasets used in this work have the same number of samples for each category. Hence, we use accuracy as the primary metric for measuring model performance on solving a given classification task. Accuracy *Acc* shows how many samples from the dataset are correctly classified:

$$Acc = \frac{1}{n}\sum_{i=1}^{n} \mathbb{1}_{(f(x_i)=y_i)} \tag{2.32}$$

2.1.12 Overfitting and Underfitting

We distinguish the following cases depending on a model's performance on training and test (or validation) sets. We note here that "poor" and "good" performance are data-specific terms that should be defined for each task individually.

1. The model performs poorly on the train set, i.e. it did not learn the representation of the training data. In this case, we say that the model is *underfitting* the data. Usually, it means that the complexity of the model is too low for the given task.

2. The model performs well on the train set, but poorly on the test set. In this case, the model *overfits* the data, i.e. it learned well the training set but lacks generalisation abilities. More complex models tend to overfit due to larger learning capacity. To reduce overfitting, various *regularisation* techniques which control, for instance, the magnitude of the learned weight update can be applied.

3. The model performs well on the train and test set. This is the state we want to achieve while training a machine learning model.

2.2 Deep Learning

Shallow machine learning approaches can be very efficient, but are often not capable of capturing sophisticated data structures. One bottleneck of shallow classification is feature extraction, which is a difficult task on its own. Therefore, some special types of neural networks have been developed to also extract features from raw data. Depending on the data domain, different types of neural networks have been proposed: Convolutional Neural Networks (CNN) for image data, Recurrent Neural Networks (RNNs) for sequential data, Graph Neural Networks (GNNs) for graph data, etc. Below we introduce neural network architectures relevant to this work.

2.2.1 Convolutional Neural Networks

Convolutional Neural Networks (CNNs) can capture spatial data characteristics, which is an essential property for solving tasks in the image domain, where neighbouring pixels together form structures. There are three main categories of layers used in CNNs.

1. *Convolutional* layers are the main component of CNNs. They consist of *filters* (also called *kernels*), which aim to detect if a particular pattern is present in the image. The *convolution* process is shown on Figure 2.4. A filter for each channel moves like a sliding window from the top left corner to the bottom right, while the dot product of the filter and the captured input values (the so-called receptive field) is computed. The step size of the sliding window is called *stride*. Since pixels on the border are only captured once, sometimes an additional frame, a so-called *padding*, is added to increase the border pixels' impact. In Figure 2.4, a 2×2 filter is applied to a single-channel input of size 4×4 with stride 2 and no padding. As a result, the filter produces four output values, one per filter slide. After the convolution, an activation function is usually applied.

2. *Pooling* layers aim to reduce the dimensionality of an input. They also consist of a filter, which moves over the input values. However, that filter computes an

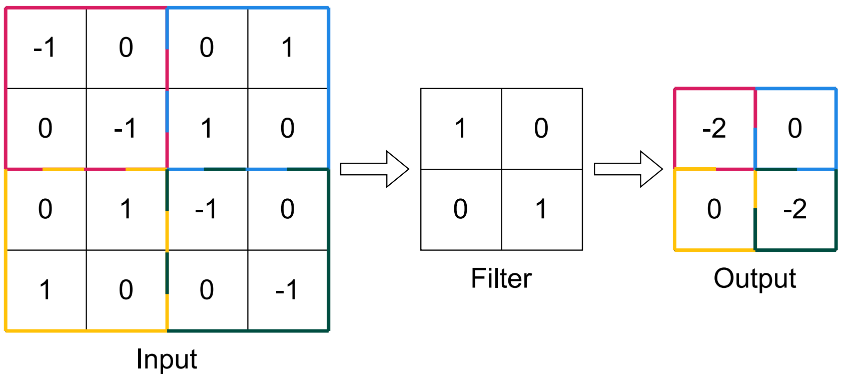


Figure 2.4: A convolution process.

aggregated value over the corresponding receptive field using a function that does not depend on trainable weights. The most common aggregation functions are maximum and averaging, and the corresponding layers are aptly called *max pooling* and *average pooling* layers. Figure 2.5 shows an application of a max pooling layer with a 2 × 2 filter to a single-channel input of size 4 × 4 with stride 2 and no padding.

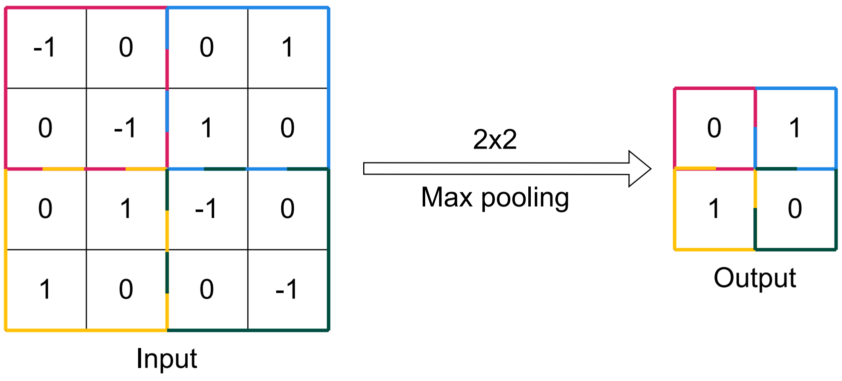


Figure 2.5: An application of a max pooling layer.

3. *Fully-connected* layers, the same as used for MLPs, are usually placed as the last layers of a CNN to perform a classification task.

We now describe several advanced techniques proposed to improve neural network training and utilised in the CNNs considered in this work.

Dropout

Dropout is a regularisation technique that randomly drops several nodes during forward and backward propagation for each mini-batch. For fully-connected layers, it means that some nodes do not contribute to a prediction, as if their weights were 0. For pooling layers, dropout means that some values of the receptive field are not taken into account while computing the aggregated value [WG15]. For convolutional layers, some of the activations calculated after a convolution are dropped [WG15].

Batch normalisation

During training, weights of a model are constantly changing, consequently changing the distribution of inputs to hidden layers and causing a covariate shift [IS15]. *Batch normalisation* [IS15] normalises the inputs to hidden layers, similarly to how data is normalised before feeding into a model. While it was initially developed to resolve the problem of covariate shift, a follow-up work [STIM18] showed that it is not the case. However, batch normalisation is still useful for speeding-up training.

Using the same notation for equations as in Section 2.1, i.e. $z^{(l)}$ denoting the output of layer l before applying the activation function, on each mini-batch, for each layer, we calculate the mean μ as

$$\mu = \frac{1}{n_l} \sum_{i=1}^{n_l} z_i^{(l)} \tag{2.33}$$

, and the standard deviation σ as

$$\sigma = \sqrt{\frac{1}{n_l} \sum_{i=1}^{n_l} (z_i^{(l)} - \mu)^2 + \varepsilon} \tag{2.34}$$

, where ε corresponds to a small number, added for numerical stability. Using these estimated mean μ and standard deviation σ values, the output of layer l is normalised as follows:

$$z_{i_{norm}}^{(l)} = \frac{z_i^{(l)} - \mu}{\sigma} \tag{2.35}$$

As the result, all layer outputs have mean 0 and standard deviation 1.

However, since having all $z_i^{(l)}$, $i \in \{1, \ldots, n_l\}$ within the same range produces limited activation values, normalised values are usually scaled using two trainable parameters $\beta^{(l)}$ and $\gamma^{(l)}$:

$$\tilde{z}_i^{(l)} = \gamma^{(l)} z_{i_{norm}}^{(l)} + \beta^{(l)} \tag{2.36}$$

Skip connections

A neural network is called *plain* if its layers are connected consecutively, i.e. the output of a layer is the input to the consecutive layer. He et al. showed that from a certain network depth on, a plain network performs worse [HZRS16]. As a solution for training deeper networks, they introduced *residual blocks*, as shown on Figure 2.6. For three consecutive layers, l, $l+1$, and $l+2$, the regular activation of layer $l+2$ is $a^{(l+2)} = g(z^{(l+2)})$. In a

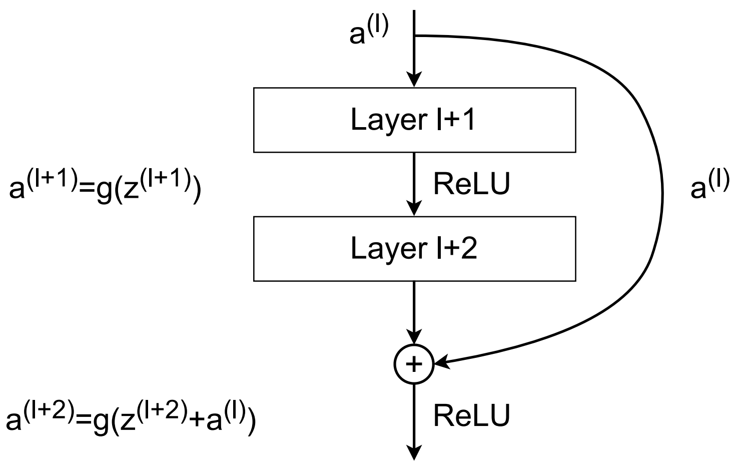


Figure 2.6: A residual block, introduced in [HZRS16].

residual block, an additional *skip connection* leading from layer l to layer $l + 2$ is added, so that the activation of layer $l + 2$ is calculated as $a^{(l+2)} = g(z^{(l+2)} + a^{(l)})$.

Residual blocks are building components of Residual Networks, also known as ResNets, as well as other architectures. Compared to plain networks, the performance of ResNets is improving with adding more hidden layers [HZRS16].

2.2.2 Transfer Learning

Features extracted by a CNN become more complex and task-specific towards the last layer. In contrast, earlier layers extract more generic features that can be useful for different image tasks [YCBL14]. Hence, learned weights can be reused for another classification task, even if data distribution and categories significantly differ from the original data used to obtain those weights [YCBL14]. This approach is called *transfer learning*, as knowledge acquired from one dataset is transferred to another. We can distinguish three patterns for transfer learning layer training.

1. Layers trained *from scratch* are initialised with random weight values and have to learn the task from zero. Usually, the final, fully-connected layer is trained from scratch, as it performs the final classification and is dataset-specific, and might often not even fit the required architecture, as the number of classes might be different.

2. *Fine-tuned* layers are initialised with learned weights and are also updated during further training, called *fine-tuning*. Sometimes, a smaller learning rate value is used to train those layers to prevent the complete erasing of previously learned patterns and control the deviation from original weights. It was also shown that fine-tuning a network initialised with pre-trained weights leads to a better generalisation of the network compared to random initialisation [YCBL14].

3. *Frozen* layers are also initialised with learned weights, but are not updated during training. It is assumed that these layers have learned to extract features required for a new task and do not need to be changed. Besides making training more effective, freezing layers also makes training more efficient, as these layers do not require training.

2.2.3 Diffusion Models

Diffusion models were first introduced by Sohl-Dickstein et al. in 2015 [SDWMG15]. Their main idea was to gradually apply a diffusion process to destroy the data structure and then learn a reversed diffusion process to generate new data instances from unstructured inputs. In the case of image data, the data structure is destroyed by gradually adding Gaussian noise until the image becomes noise itself. During the reverse diffusion process, denoising autoencoders are sequentially applied to recreate the image structure from the noise, as shown in Figure 2.7. Rombach et al. [RBL+22] showed that diffusion models reach state-of-the-art performance for image generation tasks such as text-to-image synthesis, super-resolution (i.e. increasing the resolution of an image), and image inpainting (i.e. reconstructing missing regions in an image).

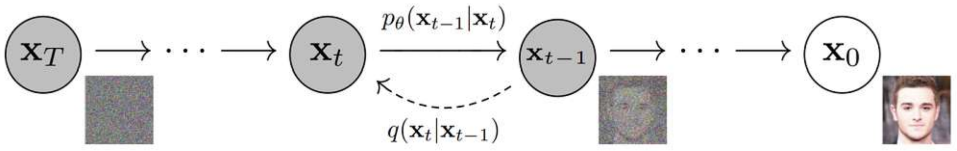

$\mathbf{x}_T$
$\mathbf{x}_t$
$p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)$
$\mathbf{x}_{t-1}$
$q(\mathbf{x}_t|\mathbf{x}_{t-1})$
$\mathbf{x}_0$

Figure 2.7: Denoising process. The image is taken from [HJA20].

This work uses diffusion models to create data for performing a data-free attack, which is discussed later in Section 4.2.

2.3 Adversarial Machine Learning

Adversarial machine learning considers potential security threats occurring during development and usage of machine learning models. A *malicious party*, also called *attacker* or *adversary*, can attack a machine learning model to target any of the aspects subsumed in the so-called *CIA triad*: confidentiality, integrity, or availability. Table 2.1 shows a categorisation of attacks against machine learning models from [BR18], structured along the goal and capabilities of the attackers. Below, we describe attack categories relevant to this work.

2.3.1 Adversarial Examples

Adversarial examples belong to *evasion* attacks (Table 2.1). They aim to compromise the integrity of a model, intervening in the prediction phase, and forcing the model to make mistakes. Szegedy et al. [SZS+14] observed that neural networks, while being able

Table 2.1: Categorisation of attacks against machine learning (by Biggio and Roli [BR18]).

Attacker's capability	**Attacker's goal**		
	Integrity	**Availability**	**Confidentiality**
Test data	Evasion (e.g., adversarial examples)	-	Model stealing, model inversion, membership inference, ...
Train data	Poisoning for subsequent intrusions (e.g., backdoors)	Poisoning to maximise error	-

to classify test samples correctly, can be easily fooled if images are slightly modified. They showed that imperceptible non-random perturbations can force a neural network to predict arbitrary labels for the same image. Such perturbed samples are called *adversarial examples*.

For dataset sample x and model f, we denote as x^{adv} an adversarial example, such that $f(x) \neq f(x^{adv})$ and $x^{adv} = x + \delta$. Here δ is a small (imperceptible) perturbation that depends on an adversarial crafting algorithm. We can distinguish two categories of adversarial examples depending on the desired model prediction on x^{adv}. If the goal is to make the model predict any different from $f(x)$ label, i.e. $f(x^{adv}) \neq f(x)$, we say that x^{adv} is an *untargeted* adversarial example. If the goal is to force the model to predict a specific label $c \neq f(x)$, i.e. $f(x^{adv}) = c$, we call x^{adv} a *targeted* adversarial example. Depending on the adversarial examples used, we can also categorise evasion attacks into targeted and untargeted.

We now describe an adversarial example crafting algorithm relevant to this work. The Deepfool algorithm, proposed by Moosavi et al. [MDFF15], was initially designed to measure quantitatively the robustness of a classification model by an approximate calculation of the minimal perturbation required to change its prediction. They showed that such perturbation can be explicitly calculated for linear classifiers, and further proposed an iterative process for linearisation for an arbitrary (non-linear) model by approximating the model's decision boundary with hyperplanes. We provide the Deepfool algorithm in Algorithm 2.1. The algorithm requires two inputs: a classifier f and a sample x, for which an adversarial example has to be found. At first, predicted probabilities $f_k(x)$ are calculated for each class k. The top-1 predicted class that corresponds to the label of x is denoted as y_0. Then, the algorithm enters an iterative process of finding the minimal perturbation required to flip the label y_0. At each iteration, the distance to the closest approximated decision boundary is estimated (lines 7-10), and a perturbation r_i required to bypass this distance is calculated (line 11). If the perturbation is large enough to change the prediction of f, the algorithm stops. Otherwise, a further perturbation is calculated. The algorithm returns the accumulated perturbation over all iterations.

Algorithm 2.1: Deepfool [MDFF15]

```
Input: classifier $f$, data sample $x$
Output: perturbation $\hat{r}$
1  $f(x) = (f_1(x), \dots, f_d(x))$ ;                    /* $d$ = #classes */
2  $x_0 \leftarrow x$ ;
3  $y_0 \leftarrow \underset{1 \le k \le d}{\arg\max} f_k(x_0)$ ;
4  $i \leftarrow 0$ ;
5  while $y_i = y_0$ do
6      for $k \neq y_0$ do
7          $\omega'_k \leftarrow \nabla f_k(x_i) - \nabla f_{y_0}(x_i)$ ;
8          $f'_k \leftarrow f_k(x_i) - f_{y_0}(x_i)$ ;
9      end
10     $\hat{l} \leftarrow \underset{k \neq y_0}{\arg\min} \frac{|f'_k|}{\|\omega'_k\|_2}$ ;
11     $r_i \leftarrow \frac{|f'_{\hat{l}}|}{\|\omega'_{\hat{l}}\|_2^2} \omega'_{\hat{l}}$ ;
12     $x_{i+1} \leftarrow x_i + r_i$ ;
13     $y_{i+1} \leftarrow \underset{1 \le k \le d}{\arg\max} f_k(x_{i+1})$ ;
14     $i \leftarrow i + 1$ ;
15 end
16 $\hat{r} = \sum_i r_i$ ;
```

2.3.2 Model Inversion

Model inversion attacks belong to confidentiality-violating attacks (Table 2.1). They aim to reconstruct training data of a machine learning model [FJR15], raising threats of leaking sensitive information. While originally Fredrikson et al. introduced various scenarios applicable for different models and input data, we focus only on a single case introduced as a *model inversion reconstruction attack* applied to reconstruct image data [FJR15]. The authors aimed to reconstruct training data of a facial recognition model. Namely, given a unique identifier of a person from the training set, they aimed to recreate an image of that person. The reconstruction was assumed successful if it was possible to identify the target person from a set of images, knowing the reconstructed image. During the attack, an adversary starts with a random input of the size of an input image and uses gradient descent to optimise this input to maximise the probability that the input belongs to the desired class. Additionally, image manipulation techniques such as denoising and sharpening can be applied. Model inversion attacks can be used as an approach to gather data of a higher quality (e.g. [GCY+21]).

2.3.3 Model Stealing

Model stealing (*model extraction*) attacks, first introduced by [TZJ$^+$16], violate the confidentiality of machine learning models. They are applied to reveal different properties of a model, or copy its entire behaviour [OMR23]. Behaviour-stealing attacks are the target of this thesis, and are described in detail in the following chapters.

2.4 Active Learning

Active learning was investigated for supervised learning tasks with a large amount of unlabelled data, which can be labelled by querying an oracle (in most cases, a human expert). The most straightforward approach is to label the whole dataset and train a model on it. However, as labelling can be expensive, an optimisation technique is needed. Active learning reduces the number of requests to the oracle by sending only samples, which can improve the performance of a model. For instance, those can be samples with the least confident predictions, or those close to the decision boundary, etc.

We now define the terminology commonly used in active-learning-related papers. In active learning, two instances can label data: a model f that is learning and an oracle o that returns the ground-truth labels. Unlabelled data from which samples are selected is called *pool*. Labelled data that corresponds to the training set of f is called *seed*. Further, most algorithms require some *initial seed*, on which f is trained before active learning is applied. An active learning selection algorithm is then repeated for one or several *rounds*, during which a predefined number of samples is selected from the pool, labelled by the oracle, and added to the seed.

Two active learning algorithms are relevant for this thesis: DFAL (DeepFool Active Learning) and κ-center. Below, we describe each of them.

- Deepfool Active Learning (DFAL) [DP18] is a sample selection strategy based on adversarial examples created by the Deepfool algorithm (Algorithm 2.1). We describe the approach of DFAL in Algorithm 2.2. For each sample x_i in a data pool D, it crafts an adversarial example x_i^{adv} for the model f using Deepfool. Then it calculates the perturbation $\|x_i - x_i^{adv}\|_2^2$ and selects the k samples with the smallest perturbation value. Afterwards, those samples (without corresponding adversarial examples) are labelled by the oracle o. Hence, DFAL selects samples that are the closest to the decision boundary of the model f.

- The κ-center algorithm, proposed by Sener and Savarese [SS18], is shown in Algorithm 2.3. The algorithm requires the existence of an initial seed S_0, used as a *core-set*. For each sample x in a pool D, distances between probabilities predicted by the model f for the core-set samples and x are calculated. Then, the sample with the largest distance to all core-set samples is selected, labelled by the oracle o, and added to the core-set. The procedure continues until k samples are selected.

Algorithm 2.2: DFAL [DP18]

```
    Input: model $f$, data pool $D$, number of samples to select $k$
    Output: $x'_1, \dots, x'_k$
1   $D = \{x_1, \dots, x_n\}$ ;
2   for $i = 1$ to $n$ do
3   |   $x_i^{adv} \leftarrow \texttt{Deepfool}(f, x_i)$ ;
4   |   $\alpha_i \leftarrow \|x_i - x_i^{adv}\|_2^2$ ;
5   end
6   $x'_1, \dots, x'_n \leftarrow$ sort $D$ in ascending order by $\alpha_i$ value ;
7   $x'_1, \dots, x'_k \leftarrow$ select first $k$ samples from $x'_1, \dots, x'_n$ ;
```

This way, the κ-center picks the most sparse samples, making the training dataset of f more diverse.

Algorithm 2.3: κ-center [SS18]

```
    Input: substitute model $\hat{f}$, initial sample seed $S_0$, data pool $D$, number of
           samples to select $k$
    Output: $x'_1, \dots, x'_k$
1   for $i = 1$ to $k$ do
2   |   $x'_i \leftarrow \arg\max_{x_j \in D} \min_{x_r \in S_{i-1}} \|\hat{f}(x_j) - \hat{f}(x_r)\|_2^2$ ;
3   |   $D \leftarrow D \setminus x'_i$ ;
4   |   $S_i \leftarrow S_{i-1} \cup x'_i$ ;
5   end
```

CHAPTER 3

Model Stealing

This chapter describes the terminology relevant to model stealing attacks and depicts a profile of an attacker that performs an attack, like their goal, actions and knowledge. Then, different attack methods are described, including current state-of-the-art substitute model training attacks. For this group of attacks, we introduce a new categorisation based on the attacker's knowledge. We conclude the chapter with a description of countermeasures against model stealing.

3.1 Definition

Model stealing attacks (sometimes called also *model extraction* attacks) represent a subcategory of adversarial machine learning that targets the confidentiality of machine learning applications and aims to create an illegitimate copy of a model or some of its assets. Usually, these attacks endanger Machine-Learning-as-a-Service (MLaaS) applications, which allow their clients to utilise prepared machine learning solutions. For instance, let there be a high-performing API that classifies given images into predefined categories, as shown on Figure 3.1.

Any API user can send an image for classification and obtain the corresponding output, which is, in this case, the type of animal depicted in the image. Sometimes, there might be a limit for the number of images a client can send to the API daily without any payments. For further access, APIs may charge fees for usage above the daily limit, making a profit that way; note that there might also be no free daily limit, and all queries are paid from the onset. However, malicious users might want to avoid additional expenses, or monetise the underlying machine learning solution on their own. They can copy the behaviour of the API by training their own model using inputs and corresponding outputs as a training dataset. Figure 3.2 shows how a model stealing attack can be performed against such API. Below we introduce the terminology and describe each stage of the attack.

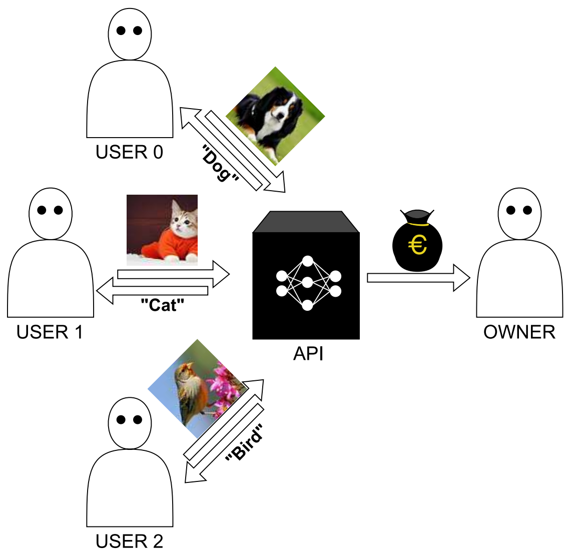


Figure 3.1: Example of an image classification API; the owner monetises the service e.g. by charging the users per query.

We call the model under attack the *target model*, and denote it as f. We further assume that the we only have *black box* access to the target model, which means that the only information an adversary can retrieve are the predictions (*outputs*) for given input samples (*inputs*). Moreover, in the context of image classifiers, we consider only target models that output top-1 predictions (*labels*) for input images. In other words, the only available action for the adversary is to send an image x to the target model and obtain a label $f(x) \in \{c_1, \ldots, c_k\}$. We call such a single request to the model a *query*, where x is the *query input* and $f(x)$ is the *query output* (or response). Using queries, the adversary can create an *attacker's dataset* (see Stage I in Figure 3.2) and then use it to train a so-called *substitute model* (Stage II in Figure 3.2). The substitute model, denoted in this work as $\hat{f}$, can e.g. further be used to launch an alternative API with lower fees, which could lead to lower demand for the original API and profit loss for the original owner (Stage III in Figure 3.2).

A successfully performed attack should satisfy two criteria: efficiency and effectiveness. In terms of efficiency, an attacker should spend a reasonable amount of resources to collect and label the data through the target model. The most crucial part of this process is the number of queries required to perform the attack. For instance, if an API only allows 10 queries per day, and an adversary has no budget to pay fees, performing an attack that requires 10,000 queries will become unreasonable, as the data collection process itself would take more than 3 years. Therefore, defining the number of queries, a so-called *query budget* of an attack, is an important step in attack designing. As suggested in [OMR23], we will also report an *efficiency score*. This metric shows how many queries per parameter (weight) of the target model it takes to perform an attack. This metric allows comparing different attacks better, as attack results could be reported on the

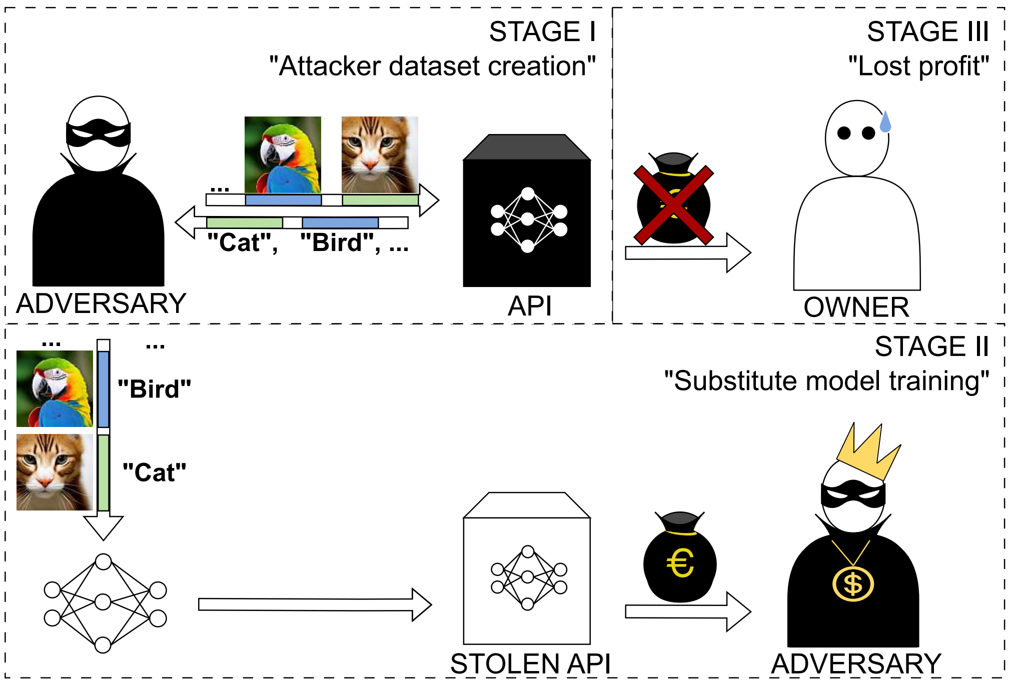


Figure 3.2: Model stealing attack on image classification API. At Stage I, an adversary labels data using the target model as an oracle. At Stage II, the adversary trains a substitute model, subsequently monetising it. At Stage III, the owner of the target model meets the consequences of stealing, e.g. loses the profit.

different target model architectures, targeting different complexity of a stealing task. Let's consider attack A, which steals a simple neural network with 100 trainable weights using 5,000 queries. Attack B steals a more complex convolutional neural network with 5 millions of trainable weights using 50,000 queries. A simple comparison of query budgets tells us that attack A is more efficient – but it targets a simpler target model. If we compare the efficiency scores, it follows that attack A with the score 50 is way less efficient than attack B with a score of 0.01. Only if the target model is the same, we suffice to report the query budget, as it will represent the same information as the efficiency score.

For the effectiveness evaluation, the performance of a substitute model is compared with the performance of the target model. Three metrics are mainly used for that purpose: accuracy, fidelity, and transferability [OMR23].

- **Accuracy** shows how a substitute model performs on the classification task that the target model was trained to solve. It compares the outputs of the substitute model with the original labels of a dataset. The accuracy of the substitute model $\hat{f}$ on dataset X^{test} is measured as:

$$\text{Accuracy} = \frac{1}{|X^{test}|} \sum_{i=1}^{|X^{test}|} \mathbb{1}_{(\hat{f}(x_i^{test})=y_i^{test})} \tag{3.1}$$

The main drawback of this metric is that the data on which the performance is evaluated should be labelled. Another issue is that Equation (3.1) does not include the target model; hence, the metric does not really show how successful stealing is. However, one can compare the accuracy of the target and substitute models, e.g. by computing *relative accuracy*, which is the relation of substitute accuracy to the target accuracy. It shows to some extent how much of the performance of the target model is covered by the substitute model. However, none of these approaches can tell how similar are predictions of the models. If both models achieve, for instance, 90% of accuracy on a test set, the relative accuracy is 1.0, but these models can still make different mistakes and output the same labels only on 80% of the test data. For that reason, we introduce in this work a metric we call *joint accuracy*, which shows how many samples are classified correctly by both the target model f and the substitute model $\hat{f}$, as follows:

$$\text{Joint Accuracy} = \frac{1}{|X^{test}|} \sum_{i=1}^{|X^{test}|} \mathbb{1}_{(f(x_i^{test})=\hat{f}(x_i^{test})=y_i^{test})} \tag{3.2}$$

- **Fidelity** measures the similarity of target and substitute prediction. It compares the labels that the target model f and the substitute model $\hat{f}$ output on test set X^{test}. Whereas joint accuracy only shows on which samples both models output the *same correct* prediction, fidelity counts *both correct and incorrect equal predictions*, as

$$\text{Fidelity} = \frac{1}{|X^{test}|} \sum_{i=1}^{|X^{test}|} \mathbb{1}_{(\hat{f}(x_i^{test})=f(x_i^{test}))} \tag{3.3}$$

Contrary to accuracy, fidelity does not require data labelled with the ground truth. Instead, it requires labels from the target model. In a real-world scenario, that would mean that an adversary has to spend a part of its query budget for obtaining labels for the attack evaluation. We relax this limitation and do not count queries needed for evaluation as a part of the query budget.

We prioritise fidelity in our attack scenarios, as it captures the behavioural similarity of two models. If we again consider models that reach 90% on a test set, their fidelity can vary from 80% to 100%, revealing more information than accuracy could potentially do. Compared to joint accuracy, fidelity better represents the similarity between two models, as it considers all equal predictions.

- **Transferability** is a special case of fidelity that focuses on the behavioural similarity of two models *near their decision boundaries*. It uses adversarial examples, crafted to fool the substitute model, as a test set on which the target and substitute model outputs are compared. Papernot et al. [PMG16] were the first to show that adversarial examples, crafted for one model, can also fool another model, i.e. those adversarial examples are *transferable*. To measure the transferability, one first needs to create an adversarial test set X^{adv} from a regular test set X^{test}. For each sample $x_i^{test} \in X^{test}$, an adversarial example generation algorithm is used to craft perturbed sample $x_i^{adv} = x_i^{test} + \delta$, such that $\hat{f}(x_i^{test}) \neq \hat{f}(x_i^{adv})$. We use Deepfool Section 2.3.1 as the adversarial algorithm in this work.

We further distinguish two cases of transferability: targeted and untargeted. These notions are closely related to the ones we introduced in Section 2.3.1 for targeted and untargeted evasion attacks. *Untargeted transferability* measures how many adversarial examples x_i^{adv} are also adversarial for target model f in that sense that they force the target model to change its prediction comparing to unperturbed sample x_i^{test}; it is computed as follows:

$$\text{Untargeted Transferability} = \frac{1}{|X^{adv}|} \sum_{i=1}^{|X^{adv}|} \mathbb{1}_{(f(x_i^{adv}) \neq f(x_i^{test}))} \tag{3.4}$$

Targeted transferability, similarly to targeted evasion attacks, requires that an adversarial sample x_i^{adv} flips the label of the target model $f(x_i^{adv}) \neq f(x_i^{test})$ in the same way it flips the label of the substitute model $\hat{f}(x_i^{adv}) \neq \hat{f}(x_i^{test})$, i.e. the perturbed labels are equal $f(x_i^{adv}) = \hat{f}(x_i^{adv})$; this measure is computed as

$$\text{Targeted Transferability} = \frac{1}{|X^{adv}|} \sum_{i=1}^{|X^{adv}|} \mathbb{1}_{(f(x_i^{adv}) \neq f(x_i^{test}) \wedge f(x_i^{adv}) = \hat{f}(x_i^{adv}))} \tag{3.5}$$

3.2 Threat Model

Above, we described the threat model considered in this work. However, in general, goals, available actions, and the attacker's knowledge might differ significantly. Following the taxonomy presented in the survey on model stealing attacks and defences [OMR23], we now briefly discuss other possible scenarios.

3.2.1 Attacker's Goals

In this work, an adversary aims to train a substitute model $\hat{f}$, which imitates the behaviour of the target model. However, in general, an adversary might also want to steal some other assets of the model, like its architecture, training hyperparameters, or learned parameters.

- An **architecture stealing** attack tries to reveal hyperparameters that characterise the architecture of the target model. For instance, for a neural network, that could be the number of layers, their types, kernel size in a convolution layer, etc. There are two reasons why an adversary might want to steal the architecture. Firstly, having information about the architecture makes training a substitute model easier. Typically, the adversary has to make assumptions about the type and complexity of the target model. If the selected architecture is not complex enough, the substitute model might not converge. Secondly, if the target model has a unique architecture, which can be beneficial in solving other tasks, the adversary can try to reuse it for their own goals.

- **Training hyperparameters** stealing aims to extract values of hyperparameters like the batch size, optimisation algorithm, regularisation hyperparameter, etc. Similarly to architecture stealing, the revealed information will not yield a full substitute model, but can support its training process. An adversary can combine those two attacks for training a substitute model with the same architecture and training hyperparameters as the target model was trained.

- Attacks that target **learned parameters** aim to steal, for instance, the weights of a neural network. To perform such an attack, an adversary needs to know the architecture of the target model, which limits the applicability of this type of attack. As mentioned before, the architecture can also be stolen, so this attack can be launched against a black-box model after an architecture stealing attack. However, the stolen architecture needs to be precise — otherwise, the weights will likely be recovered incorrectly. A successful parameter-stealing attack leads to an identical copy of the target model, which also means that it will behave identically to the target model.

- Finally, as we defined at the beginning of this section, some attacks aim to copy the **behaviour** of the target model. Most model stealing attacks target behaviour, although they can be further divided into two categories, as done in [OMR23]. The first category aims to reach the **same level of effectiveness** as the target model. Achieving this goal would mean that the accuracy of the stolen model is high. For the second category, **prediction consistency** with the target model is the primary goal. In this case, fidelity and transferability are the most important metrics, and depending on the attacker incentives, one of them can be prioritised. If an attacker wants to attack the target model with an evasion attack, transferability should be high. If consistency is needed in solving the original classification task, high fidelity should is required.

3.2.2 Attacker's Actions

Previously, we considered only one type of action – queries. They are the only available interactions if a target model is hidden behind a cloud-based API, and an adversary has no physical access to a device on which the model is executing. However, stealing is also possible if an adversary has a **side-channel** access to the target model. Using either hardware [BBJP19] or software [DSRB19] side channels, one can reveal the architecture

[HDK+20] or even the weights of the target model [ZCZL21]. In this work, we only consider an API case and do not explore side-channel model stealing attacks.

3.2.3 Attacker's Knowledge

The attacker's knowledge is the basis on which any attack is built. Having more information available can make an attack more effective and more efficient. Hence, we call an adversary who knows more *stronger*, and one who knows less *weaker*. We further describe three areas of the attacker's knowledge: the target model, available data, and API.

The weakest knowledge of the **target model** corresponds to the **black-box** setting, when an attacker can only observe the inputs and outputs of the model. The strongest attacker is the one who knows the model entirely, i.e., both its architecture and weights. In this case, no model stealing is needed for most of the above defined assets. However, one can still attack a **white-box** model to reveal which hyperparameters were used for training [WG18]. Any state between the white and black box corresponds to a **grey-box** scenario. This includes, for instance, knowledge about the architecture of the target model.

According to [OMR23], an adversary may have access to one of the following categories of data: original, problem-domain, non-problem domain, and artificial data.

- **Original** data means the target model's training data, which corresponds to the strongest knowledge.
- **Problem-domain** data correspond to the same classification task as the original data, but its distribution can be different.
- **Non-problem domain** data, which constitutes weaker knowledge than problem-domain data, stands for any (syntactically) suitable input data, e.g. any image data for image classifiers.
- **Artificial** data includes any data generated or synthesised. For instance, Gaussian noise or samples produced by a generative machine learning model are subsumed under artificial data. Depending on the quality of artificial samples, they can be more or less useful than other categories. Therefore, the attacker's strength depends on the concrete method used for crafting the artificial data.

APIs can limit information revealed to end clients. For image classification models, we can distinguish several cases, listed below, from the weakest to the strongest attacker:

- The API provides only the **top-1 label** per image;
- The API provides the **top-k labels**, i.e. the ranking of first k potential classes;

- The API provides the **top-k probability scores**;
- The API provides the **probabilities** (confidence scores, logits, etc.) for **all classes**.

Most works consider either the first scenario or the last scenario. Returning probabilities for all classes can be beneficial for both sides. Benign service clients can find such a service more informative and valuable than a restricted one, bringing more profit to the API's owner. At the same time, if an API provides exact scores predicted by the target model, training a substitute model becomes easier [PGS+20, YDZ+22]. The owner of the service can then perturb the probabilities (see Section 3.4.2) so that using them can lead to a non-converged model and failed attack [KPQ21b, MLF22]. Usually, top-1 predictions remain unchanged after such perturbations, so an attacker can still use them as in case (1). Since knowing in advance if probabilities are correct is an unrealistic scenario, it might be safer to only use labels for an attack. Therefore, we only consider the top-1 label case in this work, and thus the weakest possible assumption.

3.3 Attacks

In the previous section, we described a threat model for model stealing. This section gives an overview of concrete methods applicable under specific conditions for various attack goals. As we do not consider side channel attacks in this work, we describe techniques that require only input-output access, so-called *query-based attacks*. We use names for attack categories as in [OMR23].

We summarise the attacker's goals and knowledge of each method in Table 3.1. We note here that an attack that has learned the parameters of the target model as a stealing goal can also lead to behaviour stealing. Hence, all attacks that have "Parameters" as the attacker's goal in Table 3.1 can also copy the behaviour. In the last column of the table, if an attack is applicable for black-box image classifiers stealing, we denote it with ✓ and with ✗ otherwise. As we only consider CNN-based classifiers, we mark as not feasible categories of attacks which target (1) shallow machine learning models, i.e. models that require feature extraction beforehand, and (2) deep neural networks with specific architecture crucial for attack performance.

3.3.1 Witness-finding Attack

The *witness-finding* attack is the first model stealing attack, and was published by Lowd and Week in 2005 [LM05]. It was proposed as an approach to reconstruct the weights of a linear binary model. A linear model for every input feature f has a weight (linear coefficient) w_f, which is multiplied by the feature value during inference. To perform a witness-finding attack, one needs one positive sample x^+ (classified as class 1) and one negative sample x^- (classified as class 0). Then the feature values of x^+ are changed one by one until a *sign witness* (s^+, s^-) is found. A sign witness is a couple of samples that are identical except for one feature value f', but belong to different classes (one

Table 3.1: Comparison of query-based attacks.

Attack	**Attacker's goal**	**Required attacker's knowledge**	**Black-box image classifier stealing**
Witness-finding	Parameters	Architecture	✗
Equation-solving	Parameters, training hyperparameters	Architecture	✗
Path-finding	Parameters (nodes)	Architecture (model type)	✗
Recovery	Parameters, architecture	Architecture (not necessary)	✗
Meta-model training	Architecture, training hyperparameters	Presence of hyperparameters	✗
Substitute model training	Behaviour	–	✓

positive, one negative). Then the corresponding weight $w_{f'}$ is set to 1 or -1 depending on sign witness values. Afterwards, a line search is used to reveal the relative weight of other features. The bottleneck of the attack is finding a sign witness, as it requires lots of queries and simply does not always exist. However, a successful attack leads to exact model extraction. Two other works later adapted this approach for stealing Support Vector Machines [TZJ$^+$16, RST19].

This attack is only applicable to a small number of machine learning models. It also requires knowledge about the architecture of the target model and, therefore, cannot be applied for stealing black-box CNN image classifiers, as shown in Table 3.1.

3.3.2 Equation-solving Attack

By querying a target model f, an API client obtains an output $f(x)$ for each query input x. Hence, by sending several queries $x_1, x_2, \ldots x_n$, a malicious user can create a system of equations $f(x_i) = y_i, i \in \{1, \ldots, n\}$, where unknowns of the system are learned parameters (or weights) w of the target model $f = f_w$. If the system can be solved, the user can steal the weights of the target model as a result of an *equation-solving* attack.

Equation-solving attacks are usually performed against shallow machine learning models, such as Logistic Regression, Support Vector Machines, Multi-layer Perceptron [TZJ$^+$16], and Support Vector Regression Machines [RST19]. As witness finding, they also require full knowledge of the model architecture, so that an adversary is able to create a system of equations. It was also shown that an adversary with white-box access to a model can use an equation-solving attack to reveal training hyperparameters [WG18]. However, none of the scenarios fits into stealing black-box CNN image classifiers, as shown in Table 3.1.

3.3.3 Path-finding Attack

The *path-finding* attack [TZJ$^+$16] aims to copy Decision and Regression trees, i.e. getting the same nodes and splitting criteria as the original trees comprise. The attack works under the condition that an API client can observe both the prediction and a unique identifier of the leaf node that made that prediction. Then, an adversary can vary different feature values to find out the splitting criteria that lead to that particular node. Thus, conditions for all leaves can be recovered. Decision and Regression trees can be adapted to handle missing values, the attack can also be performed using queries with only some features present. This approach can reduce the number of queries needed for stealing [TZJ$^+$16], although some APIs might not allow such behaviour. Being only suitable for trees, this attack does not fit for stealing CNN image classifiers (Table 3.1).

3.3.4 Recovering Attack

Milli et al. were the first to theoretically explore stealing attacks against two-layer neural networks with ReLU activation functions [MSDH19]. They argued that the input space is divided by hyperplanes into small cells, such that a ReLU-NN behaves as a linear function within each cell. The authors showed how to *recover* weights that correspond to those hyperplanes using binary search.

The approach was later practically tested and extended for deeper networks by several works [JCB$^+$20, RK20, CJM20]. Rolnick et al. also showed how to extract the architecture of ReLU-DNNs [RK20]. As recovery attacks only target specific DNNs, they do not fit our goal of stealing black-box CNN image classifiers (Table 3.1).

3.3.5 Meta-model Training Attack

As mentioned in Section 3.2, many works have been published on stealing architecture and weights using side channels. However, the task of revealing the architecture through query-based interactions is much more complicated. The most comprehensive work in that area was published by Oh et al. [OAFS18], whose *meta-model* attack aims to reveal the architecture and training hyperparameters of convolutional neural networks.

The authors trained CNNs with different characteristics (number of layers, layer type, kernel size, activation function, batch size, etc.). Then they crafted a specific input image for each characteristic value such that the image can reveal if this characteristic with this value is present in the model. For instance, they crafted a sample that indicates the presence of a max-pooling layer. This sample is fed into a target model, and based on the output, a *meta-model* decides if the target model has a max pooling layer. The main limitation of this attack is that only single characteristics can be revealed, and an attacker has to be sure that these characteristics make sense for the target model.

Although the authors of the meta-model attack launched it against image classifiers, the stealing goal does not match our threat model, so we do not consider it in this work.

3.3.6 Substitute Model Training Attack

As mentioned in Section 3.1, the idea of *substitute model training* attack is to use data labelled through an API as a dataset for training a local model (see Figure 3.2). This approach does not require additional information about the target model and has no restrictions on the task domain or target model type. The first substitute model training attack was proposed by Tramer et al. [TZJ+16]. Along with other attacks presented in that paper, the authors launched their attack against shallow machine learning models. Since then, more than 50 papers have been published [OMR23], targeting Convolutional Neural Networks [PMG+17, OSF19], Recurrent Neural Networks [TYF20], BERT language models [KTP+20], Graph Neural Networks [DR20], Encoders [LJLG22], Generative Adversarial Networks [SDGA21], and Deep Reinforcement Learning agents [BH19].

As this thesis focuses on image classification, we discuss aspects and approaches relevant to that area in detail. Table 3.2 shows aggregated information from 24 relevant papers, which were selected based on the following criteria:

- The paper introduces a new substitute training attack or extends a previous work;
- Both target and substitute models are trained on image data;
- The goal of stealing is a substitute model that reaches a high score in at least one of the following metrics: accuracy, fidelity, or transferability.

We now explain the meaning of each column, possible options, and how it impacts the performance of an attack.

- **Attacker's data** corresponds to data categories used for querying the target model described in Section 3.2: original, problem-domain, non-problem domain, and artificial data. However, we look at artificial data from a different perspective now. In previous work, some generators of artificial data are trained using, e.g. non-problem domain data. Consequently, an attack that uses artificial data produced by such a generator should be classified as requiring non-problem domain data. In contrast, there are cases when artificially crafted data is the only data used for performing an attack, and no data from any other source was used. We call such an attack *data-free*, as it assumes that no data is available for the attacker. Subsequently, in terms of knowledge and availability of data, we can group papers into four categories, from the weakest to the strongest assumption:
  - Data-free attacks;
  - Attacks that utilise non-problem domain (NPD) data;
  - Attacks that utilise problem-domain (PD) data;
  - Attacks that utilise original data.

Table 3.2: Comparison of substitute training approaches.

Reference	**Attacker's data**	**Data crafting technique**	**Substitute has the same architecture**	**Query optimization**	**Target model outputs**		**Metrics**		
					Probabilities	**Labels**	**Acc**	**Fid**	**Tr**
[PMG+17]	Original or PD	Adversarial augmentation	✗			✓	✓		✓
[CSBB+18] (CopyCat)	NPD or/and PD		N/A			✓	✓		
[PGS+20] (Activethief)	NPD		✓✗	AL	✓	✓		✓	
[JSMA19]	Original	Adversarial augmentation	✓✗		✓	✓		✓	✓
[OSF19] (Knockoff)	NPD		✓✗	RL	✓	✓	✓		
[ASJ+20]	NPD		✓	RL	✓	✓	✓		
[PYZ18]	Original	Adversarial augmentation	✗	AL	N/A	N/A	✓	✓	
[PGS+19]	NPD		✓✗	AL	✓	✓		✓	
[MDN19]	NPD	Data composition	✗			✓	✓		
[YDZ+22]	Data-free (Artificial)	Generator	✓✗		✓	✓	✓		✓
[MSDH19]*	Original		✓✗				✓		
[KPQ21a]	Data-free (Artificial)	Generator	✗		✓		✓		
[RPM19]	Data-free (Artificial)	Noise	✓		✓		✓		
[BCIP20]	NPD	Generator	✓✗∼	EA	✓		✓		
[YYZ+20] (FeatureFool)	PD	Adversarial augmentation	✓✗∼		✓		✓		
[GCY+21] (InverseNet)	NPD	Model inversion	✗			✓	✓	✓	
[TMWP21] (DFME)	Data-free (Artificial)	Generator	✗		✓		✓		
[MHS21] (MEGEX)	Data-free (Artificial)	Generator	✗		✓*		✓		
[SAB22]	Data-free (Artificial)	Generator	✗			✓	✓		
[ZFS21]	Original	Adversarial augmentation	✓✗	RL	✓		✓		
[WL22]	NPD		N/A	AL		✓	✓		
[WLL+22] (Black-box Dissector)	NPD		✓✗			✓	✓	✓	✓
[YHL+22] (DTMEA)	Original		✗		✓*		✓		
[XHZ+22] (GAME)	(N)PD	Generator	✗	AL	✓		✓	✓	
Our work	Original, PD or Data-free (Artificial)	Adversarial augmentation, Generator	✓✗	AL		✓	✓	✓	✓

Technically, one can easily perform an attack designed for original data using, e.g. only PD data. However, (1) this attack may not be effective, and (2) we categorised papers based on the data they used for attacks without considering theoretical perspectives of an attack using different data sources. Some of the works considered several approaches, like trying both NPD and PD data. If a paper assumed that a small amount of original or PD data is available (even if less than 5%), we still considered those attacks as ones requiring a corresponding type of data.

Attacks implemented within this work include three different levels of data availability, from the strongest to the weakest assumption: original (for estimating the best possible extraction rate), PD (to have reasonable scores without original data) and data-free, for which a new approach is proposed. We describe our methodology in detail in Chapter 4.

- **Data crafting** is applicable for two categories of attacks: data-free attacks, and attacks that aim to create more (high-quality) data from the one available.

  – *Adversarial augmentation* is the most common approach for improving the quality of the data [PYZ18, YYZ+20, ZFS21]. The idea is to query a target model with adversarial examples crafted for a substitute model to "correct" predictions of the substitute model near its decision boundary. This approach can also be helpful for high-transferability stealing, since it specifically aims to make the decision boundaries of the target and substitute models similar [PMG+17, JSMA19].

  – *Data composition* is an approach proposed for increasing the quality of NPD data by merging two images into one [MDN19].

  – *Model inversion* is inspired by the model inversion attack [FJR15], which aims to reconstruct training data from a model. In the context of model stealing, it can result in more meaningful data when only NPD data is available [GCY+21].

  – *Generative models* are common for both data-free attacks and attacks that use data-crafting techniques to improve the quality or increase the quantity of the attacker's data samples. In the first case, an attack is launched assuming that only random noise is available [TMWP21], whereas in the second case, an attacker starts with some data, e.g. NPD [BCIP20].

  – *Noise* can also be used as a direct input for querying a model when no data or generative model is available [RPM19].

  This characteristic of an attack does not impact the strength of the attacker's knowledge just by itself, and is based on the attacker's decision to improve the data they have. We suggest considering the information in this column together with the previous one, as the choice of data crafting method depends on the initially available data.

  This work uses a generative model to create data for a data-free attack and adversarial augmentation as an optimisation technique for all attacks (see Chapter 4).

- The **target architecture** is another aspect of attacker's knowledge. If the architecture is known, an adversary can use it as the architecture of a substitute model, simplifying the whole stealing process. In general, there are two options:

  – Weaker assumption: the substitute architecture differs from the target, as the latest is assumed to be unknown (marked as ✗ in Table 3.2).

  – Stronger assumption: the substitute and target architectures are the same (marked as ✓ in Table 3.2).

  We mark papers with ✓✗ if they reported both cases. Mostly those works used the same architecture for the main batch of experiments but explored in an ablation study the impact of different architectures on their stealing approach. If target and substitute architectures were different in a paper only for some datasets, we mark that as ✓✗∼. If it was unclear which strategy the authors chose, we put N/A in the corresponding cell.

  As exploring the impact of architecture choice gives a more comprehensive view of an attack, we consider both approaches in this work.

- **Query optimisation** includes techniques that aim to increase the efficiency of an attack by reducing the number of queries. The main idea is that nearly the same attack result can be achieved by using a significantly smaller amount of data. The most common technique is using *Active Learning* (AL) [PGS+20, PYZ18, PGS+19, WL22, XHZ+22]. As mentioned in Section 2.4, active learning was initially investigated as an optimisation for labelling data in supervised learning scenarios with a significant amount of unlabelled data. Since for model stealing, the situation is the similar [CCG+20] (with the API being the oracle), active learning is widely used for query optimisation. A few other works trained *Reinforcement Learning* (RL) agents to pick samples with the highest impact on substitute model training [OSF19, ZFS21]. Barbalau et al. applied *Evolutionary Algorithm* (EA) to optimise data samples crafted by their generating model [BCIP20].

- **Target model outputs** can be more or less detailed, revealing different amount of information to an adversary. In Section 3.2, we discussed two output options: *labels* and *probabilities* (confidence scores). We mark cells corresponding to the output type(s) used in a paper as ✓. All papers that compared labels with probabilities concluded that the substitute performance is better when probabilities are used [PGS+20, JSMA19, OSF19, ASJ+20, PGS+19, YDZ+22, GCY+21]. However, some works assumed that an API returns even more revealing information. Milli et al. performed their attack assuming that gradients of the target model are available [MSDH19]. As gradients were the only source of information in their attack, both corresponding cells in Table 3.2 are empty. Miura et al. and Yan et al. considered APIs that return explanations in addition to the confidence scores [MHS21, YHL+22]. We mark those cases as ✓*. It was not possible from the description to figure out which type of output was used in one of the papers [PYZ18] and marked as N/A the two corresponding cells. Overall, for outputs of the target model, we have the following scale for the attacker's strength:

  – API outputs labels;

  – API outputs confidence scores;

  – API outputs explanations, gradients, etc.

- For each paper, we checked which **metrics** from Section 3.1 were reported. Most of the papers measured *accuracy* of a substitute model. However, as was mentioned before, solely accuracy is not the most explanatory metric for model stealing. One needs to know what is the accuracy of the target model to be able to estimate how good the performance of the substitute is in terms of accuracy. We can highlight another issue: how well-performing the target model is. If the target accuracy is far from state-of-the-art scores, reaching its performance level is likely much easier. Hence, attacks that steal low-performing models with high relative accuracy might be less effective against well-performing models. Measuring *fidelity* would fix that issue, although only seven of the 24 papers have reported those scores. *Transferability* was barely reported in previous works, even though it allows us to better understands the similarity of two models around their decision boundaries.

  We note that there is no single metric that is reported for every paper, which raises a comparability issue. Moreover, even if the same metric is used for evaluation, papers often measure it on different datasets, making the results less comparable (see statistics on datasets in Section 6.1).

We classified the papers listed in Table 3.2 based on the attacker's capabilities in Figure 3.3. The figure shows the level of strength along three axes: knowledge of the target model architecture, data used for queries, and target model outputs. The dashed vertical line divides the figure into two parts: the left corresponds to the unknown target architecture, i.e. the architecture used for the substitute model is not the same as the target model's architecture, and the right corresponds to the known target architecture, i.e. using the same architecture as the target model. Sectors correspond to particular knowledge about the data: the weakest assumption (data-free) is on the top, and the strongest (original) is on the bottom. Each sector is divided into segments by three concentric circles, which represent knowledge about target model outputs. The inner circle corresponds to labels, the middle covers probabilities, and the outer represents explainable AI (XAI) and gradients. We put references of papers into segments that correspond to the knowledge considered in the paper. If a paper covers different scenarios that fit into several segments, we put the corresponding reference in each. If some information about the knowledge is not given, we put its reference into each potential segment with an additional question mark. Within this work, we conduct experiments for six scenarios, covering most of the segments of the inner circle (highlighted in green in Figure 3.3), except for non-problem domain data.

We can make several observations based on Figure 3.3:

- Since for fair comparison, one should consider attacks within the same segment (i.e. same attacker's capabilities), **only a small fraction of works are actually comparable with each other**.

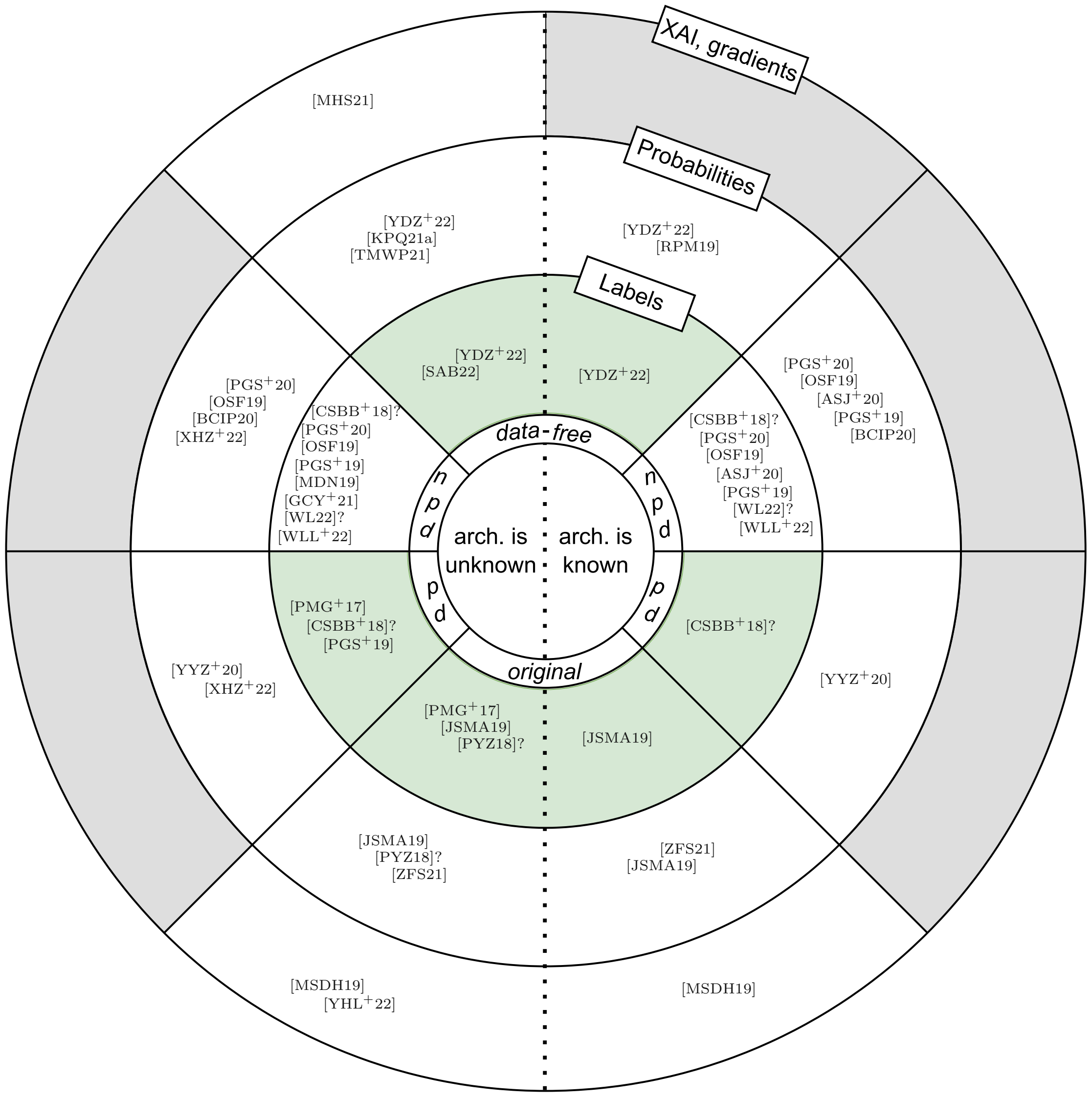


Figure 3.3: Model stealing attacks against image classifiers categorised accordingly to the attacker's capabilities.

- **Lots of configurations** with additionally provided explanations from the target model **are not explored at all**. We mark corresponding segments grey in Figure 3.3.

- **Almost half of the segments contain only one or two references**, meaning those scenarios need to be studied better.

3.4 Defences

As stated in the survey [OMR23], there are two categories of defences against model stealing: reactive and proactive. *Reactive* defences can not prevent an attack, but can detect an ongoing attack or verify the ownership of an already stolen model, i.e. prove that an attack happened in the past. Attack detection approaches are based on monitoring clients' behaviour, i.e. which queries they send to an API [JSMA19]. If an attack is suspected, the API owner can shut down the service or ban malicious clients from using the API. To verify whether an attack was performed earlier, one needs to take the following two steps:

- Before publishing the model, embed certain model-specific information into a model [SAMA21], or find some unique property [LZK21] that will be transferred into a substitute model in case of stealing.
- When an attack is suspected, check if the suspicious model contains the same information (or property) as the original target model. If the result is positive, the owner can claim that their model was stolen.

However, ownership verification alone can not help the owner, and some additional legal regulations are needed.

Proactive defences aim to mitigate or completely prevent an attack by modifying or limiting the information observable by malicious clients. These approaches include:

- Data perturbation, when either the input is modified before feeding into the target model [Gra20], or specific noise is added to the model output [OSF20].
- Model modification, which modifies the architecture of the model [LFS20] or make the weight less robust [SAAH20].
- Using task-specific architectures and avoiding transfer learning, as using a well-known architecture with publicly available pre-trained weights gives an adversary a good starting point for training a substitute model.

Any proactive defence usually decreases the utility of the target model, and hence requires finding a trade-off to not harm benign users.

Below we describe in detail different reactive and proactive techniques and their limitations.

3.4.1 Reactive Defences

In this section, we consider three reactive defences: two for ownership verification (unique model identifier and model watermarking) and one for attack detection (monitors).

Inherent Unique Model Identifier

The concept of *unique model identifiers* is similar to device or browser fingerprinting. The idea is to find some specific property or behaviour unique and inherent to the target model. Maini et al. suggested using the distances from training samples to the decision boundary of the target model [MYP21]. They argued that training data is usually well-learned by a model and located far away from its decision boundary. By training a substitute model, this knowledge is transferred and hence can be used for ownership verification. The main drawback of this approach is that the training data needs to be secret — otherwise, any model independently trained on this dataset can be falsely classified as stolen. This approach also does not fit into the scope of this work, as we use publicly available data to ensure comparability with other works.

Watermarking

Watermarking is another ownership verification approach that uses intentionally embedded information into a target model. In general, there are two categories of watermarking, depending on the level of access to a model during verification: black-box and white-box watermarking. In the first case, only model outputs are needed to verify the ownership. In the second case, one needs access to the internal structure, e.g. weights values, to check if a watermark is present. Since we consider a case when an adversary steals a model and launches a competitive API, only black-box techniques are applicable.

Black-box watermarking is based on training a model additionally, besides the original task, on a specific dataset (called trigger set) with specific labels. The idea is that either data or labels of the trigger set should be unusual enough, and thus a not watermarked model will behave differently. For instance, a trigger set can consist of images that are out of the distribution of the original training data, and a non-watermarked model would not be able to classify those as expected. Other approaches include mislabelling some specific training samples or creating a trigger set by applying a particular pattern to in-distribution data.

Initially, model watermarking was proposed as a defence against unauthorised re-distribution of an first legally obtained model [LMR23]. In this scenario, a "stolen" model is usually identical and contains the same watermark. With model stealing, the situation is different, since the model is not copied, and the model owner has to rely that the watermark is transferred during substitute model training. There have been several approaches proposed explicitly against model stealing. Szyller et al. suggested flipping some output labels returned by an API [SAMA21]. If a substitute model is later trained on queried data, flipped samples can be used as a trigger set for ownership verification; this can be seen as a "dynamic" watermarking, and implicit watermarking, as the substitute model is not directly influenced. Chakraborty et al. proposed a similar but less utility-harmful approach – instead of flipping labels, they perturbed confidence scores [CXLS22]. The idea is that a specific confidence score distribution is embedded into a substitute model and can be used for ownership verification.

We do not consider watermarking in this work, as we focus on attack mitigation techniques, although watermarking is a promising direction for future work.

Monitors

On-time detection of a model stealing attack can prevent an adversary from collecting (further) data necessary for substitute model training. It can be achieved through *monitors*, which analyse the queries each client sends to API. A monitor recognises a client as malicious if the distribution, order, or other characteristics of queries look suspicious.

The first monitor was proposed by Juuti et al. [JSMA19] to detect attacks that use adversarial examples for querying [PMG+17]. They showed that the distribution of distances between samples sent during such an attack differs significantly compared to the distribution obtained using natural data. Several other techniques were proposed to detect adversarial queries:

- Yu et al. analysed outputs of the target model after each hidden layer to distinguish adversarial and benign samples [YYZ+20].
- Liu et al. used a similar approach, but analysed outputs only of the penultimate fully-connected layer of a target model [LML+22].
- Zhang et al. checked the distances between each pair of queries, and labelled ones too close to each other as adversarial [ZCW21]. The corresponding user is blocked if the number of adversarial queries exceeds a certain threshold.
- Pal et al. used an Autoencoder to spot the difference between benign and adversarial data [PGKS21].

Sadeghzadeh et al. assumed that an adversary is more likely to send samples which are harder to learn and hence designed a monitor that can spot those samples [SSDJ22]. Two other works, by Kesarwani et al. [KMAM18] and Dziedzic et al. [DKLP22], designed monitors that measure the information leakage of submitted queries and apply some countermeasures if the leakage is too high.

The majority of monitors make some assumptions about the data a malicious user would use. If an adversary uses non-adversarial problem domain data, their behaviour is indistinguishable from benign users. Moreover, adaptive attackers, which adapt their attack to a defence applied, are barely considered in the works mentioned above and can likely overcome those monitoring systems. We will analyse this scenario in future work.

3.4.2 Proactive defences

This section briefly examines three attack mitigation techniques: data perturbation, model modification, and task-specific architecture.

Data perturbation

The goal of data perturbation defences is to hide the real behaviour of a target model through modifying queries or target model's outputs. We distinguish two categories: *input perturbation* [Gra20] and *output perturbation* [OSF20]. If an API returns confidence scores, both defences will perturb these scores using different mechanisms. However, to keep the utility of the model high, a defence should preserve the ranking of the top-1 response of the target model.

Figure 3.4 shows an example of an API that classifies images into three categories: horse, deer, or cloud. For each input image, the API returns probabilities of belonging to each class.

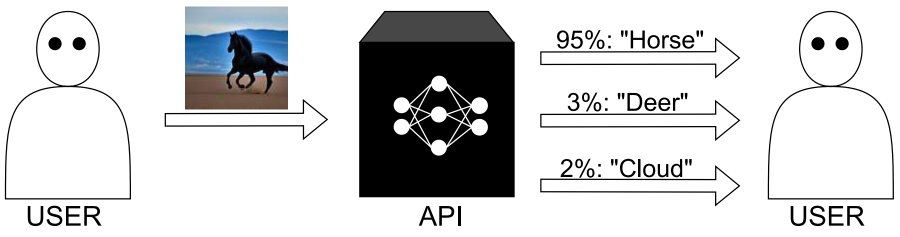


Figure 3.4: An unprotected API.

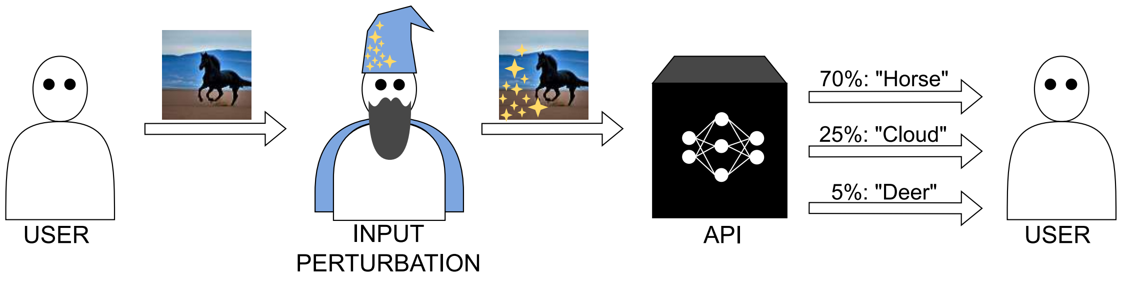


Figure 3.5: An API protected by an input perturbation defence.

When applying input perturbation, an image is changed before it is fed into the target model, as shown at Figure 3.5. As an indirect result, probability scores and the order of classes (most likely) differ from the original. In the literature, only one approach for protecting image classifiers by input perturbation was proposed [Gra20]. We describe it in detail in Section 5.1.

Output perturbation directly manipulates the outputs of the target model, as shown in Figure 3.6. It provides more control over what an end-user receives, compared to input perturbation. Together with monitors, output perturbation is the most developed area of defence against model stealing [OMR23]. Most works assume that a target model returns probability scores, which an adversary uses for training a substitute. Those scores can be perturbed without largely affecting benign clients by, for instance, preserving the order of the top classes. However, such perturbation can be enough to mislead substitute model training. Below we briefly discuss works that address confidence scores perturbation:

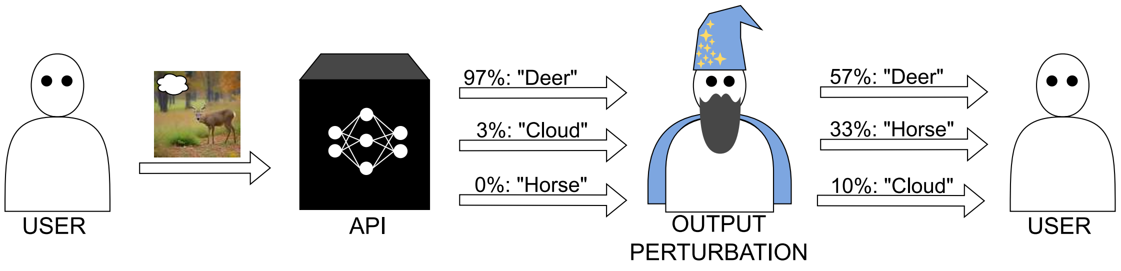


Figure 3.6: An API protected by an output perturbation defence.

- Orekondy et al. proposed to maximise angular deviation, such that gradients obtained from perturbed confidence scores are as far as possible from the original ones [OSF20];
- Lee et al. used an activation function that has collisions, i.e. it outputs the same value for different inputs, and hence makes backward propagation difficult [LEMS19];
- Chen et al. modified confidence scores such that the distribution of output samples is different [CWS$^+$20];
- Mazeika et al. introduced modifications that aim to mislead a substitute training process while keeping the overall amount of perturbations small [MLF22]. To do that, they solved an optimisation problem that minimises the perturbation while maximising the gradient update of the substitute model towards a predefined direction.

As discussed in Section 3.3.6, there could be more detailed and additional outputs, such as the gradients, or explanations of predictions. Lee et al. [LHL22] proposed a defence against attacks on models that return both confidence scores and gradient explanations. They kept the order of top-k classes the same while modifying outputs to make the obtained gradients perpendicular to the original. Protecting models that output labels only is the most challenging case for output perturbation, as any modification also affects benign clients. However, this scenario can be the most realistic as an adversary, aware that confidence scores are modified, can use only top-1 outputs to train a substitute. While this might lead to a worse effectiveness of the stolen model than if confidence scores that were not tampered with were used, it might be more beneficial than using perturbed outputs. We address the protection of label-outputting models in detail in Section 5.2.

Model modification

Another approach to hide the real outputs of a target model is to modify the model itself. One can use, for instance, knowledge distillation [HVD14] to transfer the knowledge of the target (teacher) model into a simpler (student) model. That simpler model can then be used instead of the original target model [XSZ$^+$18]. We note here that knowledge distillation differs from model stealing, as it assumes full white-box access to the teacher model. Moreover, the outputs of the student network should imitate the behaviour of the teacher also in intermediate layers and not only in the final output. Model modification approaches are especially applicable when the architecture of the target model is a stealing goal. One can add some dummy layers [CDG20] or split the architecture into two parts [LFS20] to hide how the original target model is composed.

Task-specific architecture

Training deep learning models became much easier with transfer learning, both in terms of efficiency, but to some extent also effectiveness. However, if a target model was obtained as a result of fine-tuning a publicly available model with pre-trained weights, stealing it can be much easier as well. An adversary can make an educated guess about the model architecture based on its domain and classification task, ending up using the same architecture as the target model. In this case, an adversary has an advantageous starting point, as they need to (and often quite closely actually do) repeat the training (fine-tuning) process of the target model. A few works showed that using transfer learning and well-known architectures makes a target model more vulnerable. Alti et al. showed that the relative accuracy of a Knockoff attack [OSF19] decreases substantially if a domain-specific architecture is used for a target model. Krishna et al. showed that stealing a language model trained from scratch is more challenging than stealing a pre-trained model [KTP$^+$20].

CHAPTER 4

Attack Design and Implementation

In this chapter, we describe the substitute model training attacks investigated in this work. We start with defining the attacker's knowledge and capabilities, covering the choice of the architecture of a substitute model and the attacker's data. In particular, we introduce our novel data-free attack in Section 4.2. Then we discuss query optimisation strategies, namely active learning and adversarial augmentation.

4.1 Attacker's Knowledge and Capabilities

We categorised previous works based on attacker's strength in Section 3.3.6 (Figure 3.3). In this work, we make the following assumptions about the adversary.

Attacker's data.
An adversary can use original, problem-domain, or no data at all. We use the first two options as baselines to evaluate the performance of our data-free attack. We do not consider non-problem domain data, as we launch our data-free attack as an alternative to NPD attacks proposed in the literature.

Substitute model architecture.
We consider two available options here: either an adversary knows the architecture of the target model and uses it for training a substitute, or they make a domain-based guess and use a different architecture.

Target model outputs.
We limit available outputs to the weakest scenario, assuming that an adversary can only obtain output labels.

4.2 Data-free Attack

Our novel data-free attack is based on a text-to-image diffusion model: a pre-trained stable diffusion model[1]. The main idea is to approximate problem domain data with artificial images. For each dataset category, we create text prompts to generate images. Each prompt consists of two parts: positive and negative. The positive prompt corresponds to inclusion criteria, and the negative prompt corresponds to exclusion criteria for generated images. Positive prompts usually include either a class label ("bird") or its subcategory ("swan"). Using subcategories makes the artificial data more diverse to approximate the original data better. However, if no information is provided on the subcategories appearing in the original dataset, using them can be even misleading for a substitute model. Negative prompts aim to correct mistakes a diffusion model may make. For instance, they can prevent occurrences of bad anatomy or visual artefacts. We describe prompts used in this work in Section 6.1.3. In addition, we provide results from the user study on the similarity of the generated and original data samples.

4.3 Query Optimisation

In Section 3.3.6, we described state-of-the-art query optimisation techniques. In this section, we explain specific algorithms implemented in this thesis. We carry out two query optimisation techniques: active learning and adversarial augmentation. We also describe the way we combined these strategies.

4.3.1 Active Learning

In Section 2.4, we introduced the terminology related to active learning algorithms. Now, we bring it into the context of model stealing. In active learning, there is a model that has to be trained on data labelled by an oracle. For model stealing attacks, a substitute model $\hat{f}$ corresponds to that trainable model, and the target model f corresponds to the oracle. Further, both seed (labelled data) and pool (unlabelled) data belong to the attacker's data. Hence, at each active learning round, the target model labels a certain amount of the attacker's data.

The active learning optimisation strategy used in this work is a slightly modified version of an approach from previous work by Pal et al. [PGS$^+$20]. The authors compared several active learning strategies for image and text classification. We selected the strategy with the best performance rate on image classification tasks. It combines two active learning algorithms described earlier in Section 2.4: DFAL (Algorithm 2.2) and κ-center (Algorithm 2.3). Pal et al. [PGS$^+$20] combined the algorithms in the following way. In each active learning round, they applied DFAL to select q samples, where q corresponds to the total query budget. Then they applied κ-center to select k samples out of q. In this work, in order to decrease the computation time and make the attack more efficient,

[1]`https://huggingface.co/stabilityai/stable-diffusion-2-1`

we made the following changes: (1) reducing the number of samples selected by DFAL to $2k$, and (2) splitting the pool into several sub-pools, so that at each round data is selected from a single sub-pool. The latter is applied to all optimisation techniques.

We summarise the stealing process with active learning in Algorithm 4.1. As for any model stealing attack, we need target and substitute models, the attacker's data (pool), and the query budget. Additionally, we have two parameters specific to attacks with query optimisation: seed size and the number of rounds. We assume that the seed is randomly selected from the pool, and the substitute model is trained on it. The value of k (number of samples to select per round) is calculated based on query budget, seed size, and the number of rounds. At each round, we consequently apply DFAL and κ-center, add selected samples to the seed, and, as suggested in previous work, train the substitute model from scratch on the augmented dataset.

Algorithm 4.1: Active learning attack

```
   Input: target model $f$, substitute model $\hat{f}$, data pool $D$, query budget $q$, seed
          size $s$, number of rounds $r$
   Output: $\hat{f}$
1  $S_0 \leftarrow$ select randomly $s$ samples from $D$ ;
2  $y_0 \leftarrow f(S_0)$ ;
3  $\hat{f} \leftarrow \texttt{Train}(\hat{f}, S_0, y_0)$ ;
4  $k \leftarrow \frac{q - |S_0|}{r}$ ;
5  for $i = 1$ to $r$ do
6      $x_1 \dots x_{2k} \leftarrow \texttt{DFAL}(\hat{f}, D, 2k)$ ;
7      $x'_1 \dots x'_k \leftarrow \kappa\texttt{-center}(\hat{f}, S_{i-1}, \{x_1 \dots x_{2k}\}, k)$ ;
8      $D \leftarrow D \setminus \{x'_1, \dots, x'_k\}$ ;
9      $S_i \leftarrow S_{i-1} \cup \{x'_1, \dots, x'_k\}$ ;
10     $y_i \leftarrow y_{i-1} \cup \{f(x'_1), \dots, f(x'_k)\}$ ;
11     $\hat{f} \leftarrow \texttt{Train}(\hat{f}, S_i, y_i)$ ;
12 end
```

We also provide a simplified attack scheme in Figure 4.1. Compared to Figure 3.2, we now have an additional *Stage 0*, during which an adversary selects data for querying. As a result, the attack should become more efficient in terms of querying: requiring fewer queries than a non-optimised attack to reach the same performance. The scheme is simplified, as in general, Stages 0-II can be looped, hence representing active learning rounds.

4.3.2 Adversarial Augmentation

The second query optimisation technique used in this work is adversarial augmentation. Instead of selecting the most promising samples as active learning, adversarial augmen-

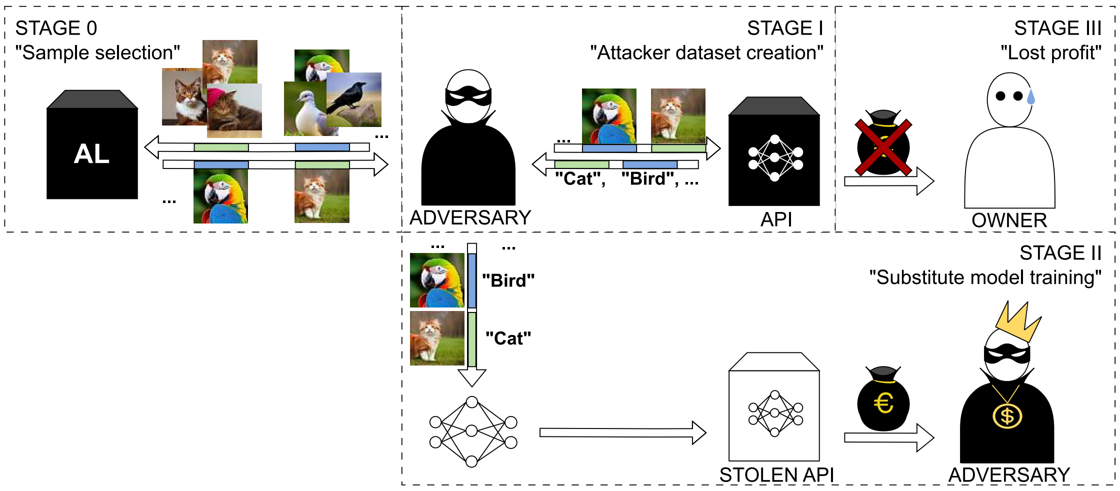


Figure 4.1: Model stealing attack with active learning query optimisation.

tation modifies samples to make them more information-revealing. Since adversarial examples lay close to the decision boundary, it is assumed that they should help better approximate the target model's decision boundary. Similarly to active learning, we picked the adversarial example crafting strategy based on previous work. Pengcheng et al. [PYZ18] compared different adversarial crafting techniques for model stealing targeting image classifiers. As Deepfool performance was the most promising, we selected it for our adversarial augmentation attack.

We summarise our implemented attack in Algorithm 4.2. The input parameters are the same as for the active learning attack. At each augmentation round, we randomly select samples from the pool, which are then augmented with their Deepfool adversarial examples. Both clean and adversarial samples are then labelled by the target model and added to the seed. The substitute model is trained from scratch after each round.

Figure 4.2 shows a simplified scheme of a model stealing attack with adversarial augmentation. As with active learning, we now have a new Stage 0, during which an adversary augments their data using adversarial examples. However, using adversarial examples can lead to new consequences: at some point, some adversarial examples can also become adversarial for the target model and hence will be mislabelled compared to their unperturbed originals. That can lead to a better approximation of the decision boundary of the target model. As adversarial augmentation can also be used in rounds, the output of Stage II can be used as input to Stage 0. Hence, as we have seen before, the scheme can be looped.

4.3.3 Active Adversarial Augmentation

Active learning aims to select the most meaningful samples for a substitute model to learn. Adversarial augmentation aims to make an approximation of the decision boundary of

Algorithm 4.2: Adversarial augmentation attack

```
Input: target model $f$, substitute model $\hat{f}$, data pool $D$, query budget $q$, seed
       size $s$, number of rounds $r$
Output: $\hat{f}$
1  $S_0 \leftarrow$ select randomly $s$ samples from $D$ ;
2  $y_0 \leftarrow f(S_0)$ ;
3  $\hat{f} \leftarrow \texttt{Train}(\hat{f}, S_0, y_0)$ ;
4  $k \leftarrow \frac{q - |S_0|}{r}$ ;
5  for $i = 1$ to $r$ do
6      $x'_1 \dots x'_{\frac{k}{2}} \leftarrow$ select randomly $\frac{k}{2}$ samples from $D$ ;
7      $x'_{\frac{k}{2}+1} \dots x'_k \leftarrow \texttt{Deepfool}(\hat{f}, x'_1), \dots, \texttt{Deepfool}(\hat{f}, x'_{\frac{k}{2}})$ ;
8      $D \leftarrow D \setminus \{x'_1, \dots, x'_k\}$ ;
9      $S_i \leftarrow S_{i-1} \cup \{x'_1, \dots, x'_k\}$ ;
10     $y_i \leftarrow y_{i-1} \cup \{f(x'_1), \dots, f(x'_k)\}$ ;
11     $\hat{f} \leftarrow \texttt{Train}(\hat{f}, S_i, y_i)$ ;
12 end
```

the target model better. We combined these two approaches to see if together they could improve the performance of the attack, leveraging the advantages of both strategies.

The combined attack is shown in Algorithm 4.3. The algorithm repeats the behaviour of the adversarial augmentation attack (Algorithm 4.2) with one difference: line 6 in Algorithm 4.2 is replaced with lines 6 and 7 in Algorithm 4.3. Instead of randomly selecting $\frac{k}{2}$ samples out of the pool, they are now selected using active learning algorithms.

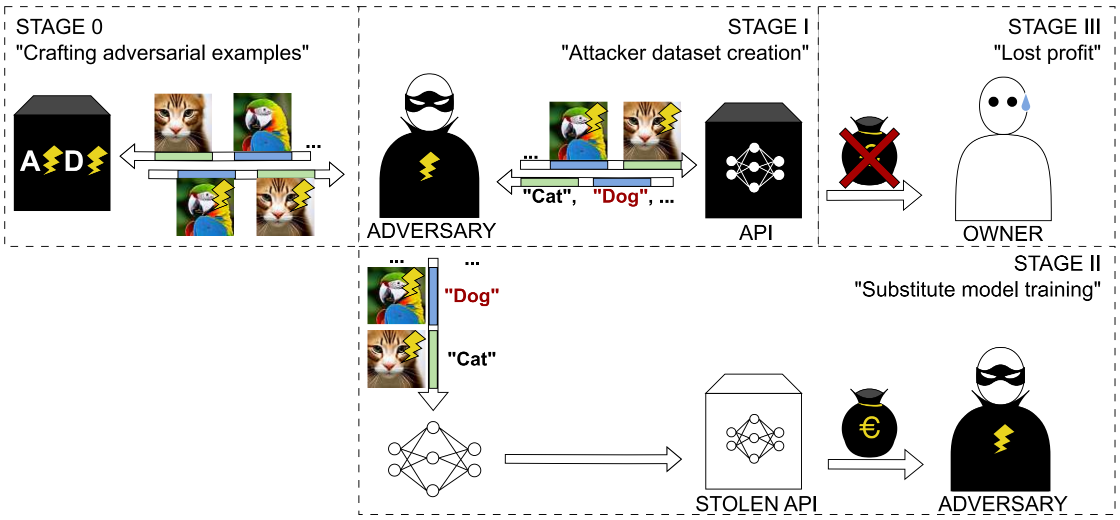


Figure 4.2: Model stealing attack with adversarial augmentation. At Stage 0, an attacker crafts adversarial examples, followed by Stage I, during which the attacker's data is labelled by a target model. At Stage II, the attacker trains a substitute model, which leads to a loss of profit at the model owner's site at Stage III.

Algorithm 4.3: Active adversarial augmentation attack

```
Input: target model $f$, substitute model $\hat{f}$, data pool $D$, query budget $q$, seed
       size $s$, number of rounds $r$
Output: $\hat{f}$
1  $S_0 \leftarrow$ select randomly $s$ samples from $D$ ;
2  $y_0 \leftarrow f(S_0)$ ;
3  $\hat{f} \leftarrow \texttt{Train}(\hat{f}, S_0, y_0)$ ;
4  $k \leftarrow \frac{q - |S_0|}{r}$ ;
5  for $i = 1$ to $r$ do
6      $x_1 \dots x_k \leftarrow \texttt{DFAL}(\hat{f}, D, k)$ ;
7      $x'_1 \dots x'_{\frac{k}{2}} \leftarrow \kappa\texttt{-center}(\hat{f}, S_{i-1}, \{x_1 \dots x_k\}, \frac{k}{2})$ ;
8      $x'_{\frac{k}{2}+1} \dots x'_k \leftarrow \texttt{Deepfool}(\hat{f}, x'_1), \dots, \texttt{Deepfool}(\hat{f}, x'_{\frac{k}{2}})$ ;
9      $D \leftarrow D \setminus \{x'_1, \dots, x'_k\}$ ;
10     $S_i \leftarrow S_{i-1} \cup \{x'_1, \dots, x'_k\}$ ;
11     $y_i \leftarrow y_{i-1} \cup \{f(x'_1), \dots, f(x'_k)\}$ ;
12     $\hat{f} \leftarrow \texttt{Train}(\hat{f}, S_i, y_i)$ ;
13 end
```

CHAPTER 5

Defence Design and Implementation

In this chapter, we describe the countermeasures implemented in this thesis against model stealing attacks. We aim to verify if one can mitigate a black-box substitute training attack that uses only top-1 labels without significant utility sacrifice. For this purpose, we focus on proactive defences that modify queries, namely input and output perturbation defences.

5.1 Input Perturbation

Input perturbation defence aims to decrease the performance of a substitute model by modifying an input image before it is sent to the target model. For a query sample x and an input perturbation ip, the target model f outputs $f(ip(x))$, instead of $f(x)$.

As an input-perturbation defence for image classifier, we consider an approach proposed by Guiga and Roscoe [GR20]. They utilised a Guided Grad-CAM [SCD$^+$17] algorithm to define how much each pixel influenced the model prediction. Afterwards, they selected the least important pixels and added noise to them. Such perturbations aim to keep the top-1 prediction of a model the same, but significantly reduce its confidence.

Algorithm 5.1 describes the image perturbation process, the key steps of which are depicted in Figure 5.1. We start with an input image x (Figure 5.1a) and pass it, together with the corresponding label y, to the Guided Grad-CAM algorithm, which returns *attributions* that correspond to the pixel importance (Figure 5.1b). In the next step, the perturbation to be added to the image is created. We denote an attribution of a single pixel as p and the maximum attribution value of the image as m. If p satisfies the condition $0 \leq p < m * t$ for a threshold parameter t, noise drawn from the normal distribution $N(\mu, \sigma)$ is added to the corresponding pixel. Figure 5.1c shows noise that is

Algorithm 5.1: Input perturbation

```
    Input: image $x$, label $y$, threshold $t$, noise mean $\mu$, noise standard deviation $\sigma$
    Output: perturbed image $x' = x + n$
1   attributions $\leftarrow$ GuidedGradCAM$(x, y)$ ;                /* Figure 5.1b */
2   $m \leftarrow$ max(attributions) ;
3   mask $\leftarrow (0 \leq$ attributions$) \times ($attributions $\leq m * t)$ ;
4   $n \sim N(\mu, \sigma^2)$ ;
5   $n \leftarrow n *$ mask ;                                           /* Figure 5.1c */
6   $x' \leftarrow x + n$ ;                                             /* Figure 5.1d */
```

added to the image, and Figure 5.1d shows the final result. Parameters t, μ, and σ are picked so that the perturbation does not affect the top-1 prediction of the model. In fact, in the original paper, the authors focused on finding feasible values for t while fixing μ and σ. In this work, we also considered different noise distributions to find the optimal one.

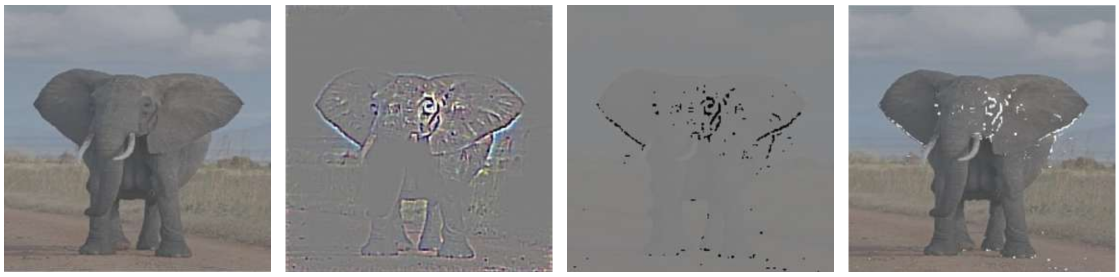

(a) Original image (b) Guided Grad-CAM (c) Noise added (d) Noisy image

Figure 5.1: Image perturbation. The images are taken from [GR20].

Originally, the authors tested their defence against an equation-solving attack [TZJ+16] that aims to reproduce the weights of the last fully-connected layer of a neural network, assuming that the attacker knows the architecture, the rest of the model weights and confidence scores predicted by the target model. In contrast, we consider this defence against a substitute training attack, assuming that the target model is a black box that only returns top-1 labels. As this defence mainly perturbs confidence scores and we are only using labels, applying exactly the same strategy the authors described in their paper will barely affect the attack. So instead of keeping all labels the same, we will add noise such that approximately 1% of the labels are flipped on the validation set. We describe in detail how the hyperparameters were picked in Section 6.3.1.

5.2 Output Perturbation

Output perturbation modifies the predictions of a target model, which are subsequently sent to clients. The target model f returns an output $f(x)$ for an input x. Then an output perturbation *op* is applied to the output $op(f(x))$ and returned to the client. As with an input-perturbation defence, the client receives a modified response; the difference is that modifications happen at different stages, and is more directed when output perturbation is applied.

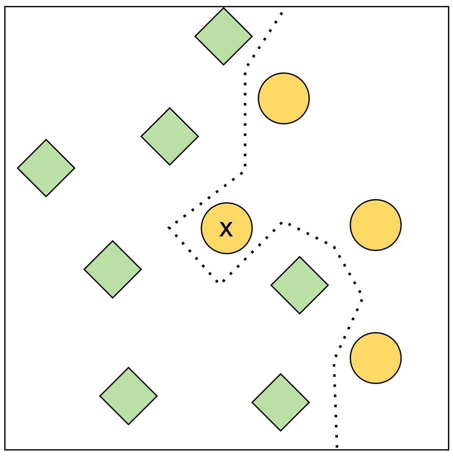
The majority of output-perturbation defences assume that an adversary uses confidence scores returned by the target model. Consequently, these approaches aim to modify the probabilities while preserving the order of top-k classifications. Since this work assumes the availability of only the top-1 labels, such perturbation techniques will not affect outputs that the adversary will obtain. Two works have presented strategies that use label flipping as a defence. Shi et al. [SS17] proposed to flip labels to protect the target model against an evasion attack. However, they assumed that an attacker would send strongly imbalanced (in terms of classes) queries, targeting the same class to find an adversarial example. In the defence presented by Kariyappa et al. [KQ20], labels are flipped for samples that are classified as out-of-distribution. We do not assume any out-of-distribution or strongly imbalanced data in our attack scenarios, although we assume that an adversary can use adversarial examples for attacking the model. Hence, we consider a defence originally invented by Cao and Gong [CG17] for mitigating evasion attacks.

A regular classifier returns one prediction for one data point, i.e. performs point-based classification. Instead, Cao and Gong suggested using a so-called region-based classifier, which predicts a label for a sample considering outputs of the model on neighbouring samples [CG17]. For each data sample x, they consider a hypercube $B(x, r)$ with the centre x and edge length $2r$ (Figure 5.2b). In this hypercube, d uniformly distributed samples $x_1, \ldots, x_d$ are randomly selected and labelled by the model $f(x_1), \ldots, f(x_d)$. Then they calculate how many predictions correspond to each possible label $\{c_1, \ldots, c_k\}$, and set the most frequent value as the prediction for x, i.e. perform majority voting:

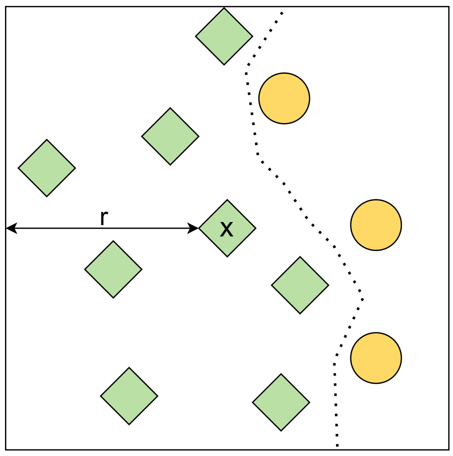
$$f(x) = \arg\max_j \sum_{i=0}^{d} \mathbb{1}_{(f(x_i)=c_j)}$$

In Figure 5.2, we have $d = 10$ and a binary classification problem, where every sample is classified as square or circle. Figure 5.2a shows a point-based classification of (potentially adversarial) data sample x. Contrary, Figure 5.2b depicts a region-based classifier that "fixes" such suspicious classification, therefore smoothing the decision boundary. Since most samples are squares in the highlighted region, the region-based classifier changes the prediction from circle to square.

The only hyperparameter one needs to define for this defence is the distance r, which characterises the size of the hypercube. In the original paper, the authors increased the value of r until the accuracy of the corresponding region-based classifier was not smaller than the accuracy of the point-based classifier. However, since we aim to keep the outputs of the target model as similar to the original ones as possible, we used different criteria. Similarly to the input perturbation, we require that the fidelity of region-based to point-based classifiers is at least 99% (see Section 6.3.2 for more details).



(a) Point-based classification.



(b) Region-based classification.

Figure 5.2: Comparison of point-based and region-based classifications.

The defence was invented to resist adversarial examples, which are crafted by applying minimal perturbations, and hence lay close to the original samples. In this work, we use Deepfool [MDFF15] as adversarial-examples-crafting technique for an advanced substitute training attack (see Section 4.3.2). We want to verify if region-based classification can mitigate any advantage that an attacker might obtain from using adversarial examples for querying the target model.

5.3 Input-output Perturbation

We combine the aforementioned input and output perturbation defences to investigate if their combination can provide a stronger defence against substitute training attacks. The combination is done in the following way. At first, we add noise to an input image x, as described in Section 5.1, thus obtaining a perturbed image $x' = x + n$. Then we pass the perturbed image to the corresponding region-based classifier f_r, as described in Section 5.2, to get a label $y' = f_r(x')$. We note here that the region-based classifier might output different labels for the same image with and without input perturbation, i.e. it could happen that $f_r(x) \neq f_r(x')$. The reason is that since x' differs from x, the regions considered for region-based classification are also different. Consequently, it may lead to further utility decrease compared to applying solely either input or output perturbation. Moreover, we do not fine-tune hyperparameters in the case of the combined defence. Instead, we use the optimal values obtained for input (Section 6.3.1) and output (Section 6.3.2) perturbations and measure the utility of the protected target model.

CHAPTER 6

Experiment Design

In this chapter, we describe settings and hyperparameters used for all experiments conducted within this thesis. As we highlighted in Section 3.3.6, experiment settings of related work are inconsistent, leading to non-comparability issues. Hence, we carefully designed our experiments to ensure a fair comparison with related work. We begin this chapter with an overview of the datasets used for training target and substitute models, where we in detail describe an artificial dataset generated for a data-free attack (Section 4.2). Next, we describe the setup for the target model training, followed by the defence parameter optimisation procedure. Then, we cover the substitute training process together with hyperparameter setting for attacks with query optimisation. We conclude the chapter by summarising the experiments conducted in this thesis to answer the formulated research questions. Most experiments in this thesis were organised and tracked using Weights and Biases[1].

6.1 Datasets

We begin with a description of the datasets used for training target and substitute models and a justification of our choice of datasets. As stated in Chapter 4, we need three categories of datasets: original, problem-domain, and artificial (for the data-free attack).

6.1.1 Original Dataset

As we aim to compare the results of this work with as many works as possible, we gathered statistics over model stealing papers, targeting image classifiers, shown in Figure 6.1. We reviewed 24 papers mentioned in Section 3.3.6. As we can see, most of them use CIFAR10 to train the target model. Following this observation, we selected CIFAR10 for our work.

[1] `https://wandb.ai/site`

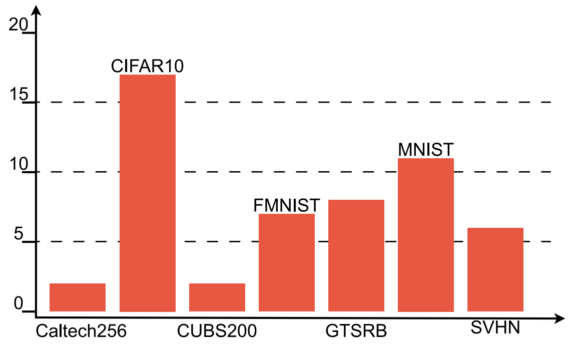


Figure 6.1: Number of papers using certain datasets for target model training.

The CIFAR10 dataset [KH09] represents objects of ten classes: airplane, automobile, bird, cat, deer, dog, frog, horse, ship, and truck (Figure 6.2). Each class contains 5,000 train and 1,000 test images of size 32 × 32 pixels. As we need a validation set for hyperparameters tuning, we split the train set into a smaller train (45,000) and a validation (5,000) set.

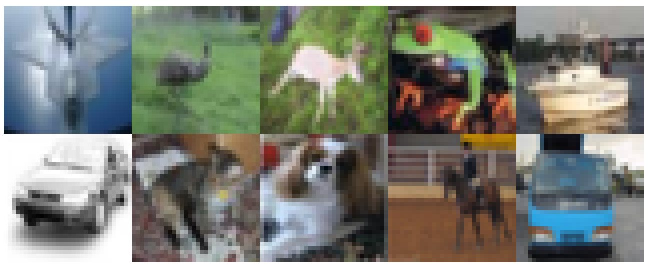


Figure 6.2: CIFAR10 dataset [KH09].

6.1.2 Problem-domain Dataset

We did not find a suitable problem-domain dataset for CIFAR10 used in the related work. The only option proposed in previous work is to use the STL-10 dataset. However, it has nine classes in common with CIFAR10, and the tenth class differs. The authors of the work that used STL-10 as PD data [CSBB+18] removed the tenth class from both datasets. However, it leads to non-comparability issues as the test set of CIFAR10 is also modified. Hence, we decided to search for an alternative dataset that contains the same ten classes as CIFAR10.

CINIC10 [DCAS18] was introduced as an extended alternative for CIFAR10. It consists of the same ten classes, but the total number of samples is 270,000. Each image has the same size as CIFAR10 images, namely 32 × 32 pixels. The dataset has train, validation, and test subsets, each containing 90,000 data samples. Since CIFAR10 is a part of CINIC10, we had to filter it out to guarantee that our problem-domain dataset is indeed problem-domain and not original. Filtering reduced the number of samples to 70,000 for

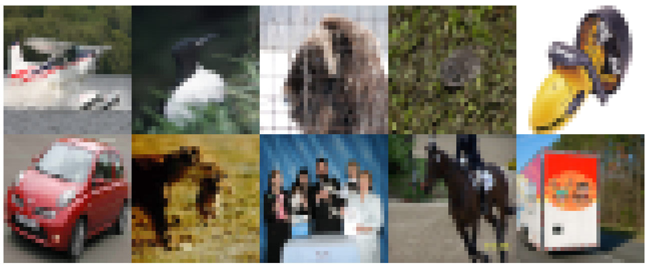


Figure 6.3: CINIC10 dataset [DCAS18].

each split. We show an example of each class from the filtered dataset in Figure 6.3. The presented images demonstrate that CINIC10 is more challenging than CIFAR10. For instance, an image from the "dog" class also contains people holding a dog. Contrary, in CIFAR10, the object to classify generally covers most of the image (Figure 6.2).

6.1.3 Artificial Dataset

We described our approach to generate the artificial dataset in Section 4.2. In this section, we describe concrete parameters used to produce data. In Table 6.1, we summarise prompts we used to imitate each of the CIFAR10 classes. Sometimes the generative model was biased towards generating very similar images for the same prompt. For instance, it constantly rendered birds of the same size and colour. Hence, we selected and used as prompts several sub-categories for these classes, to make the dataset more diverse. We also replaced "airplane" with "plane" in the positive prompt because the model always generated the same type of aircraft, up in the air, for the "airplane" prompt. In contrast, "plane" images were more diverse and contained both flying and still vehicles. In total, we generated 5,000 images per class, resulting in a dataset of size 50,000. As with CIFAR10, we split the data into train (45,000) and validation (5,000) sets. Since the generative model produced all images of size 512×512 pixels, they all were re-scaled to CIFAR10 image size 32×32 pixels.

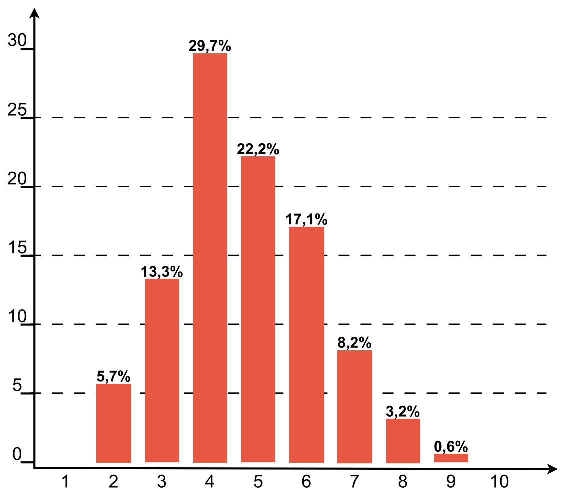
As we do not want to have any original data leaking into our data-free attack, we have to verify that CIFAR10 was not part of the training dataset of our generative model. The stable diffusion model we used was trained on LAION-5B [SBV+22], a dataset designed explicitly for text-to-image models. CIFAR10 was not a part of it; hence the generative model has never seen the original dataset.

User study

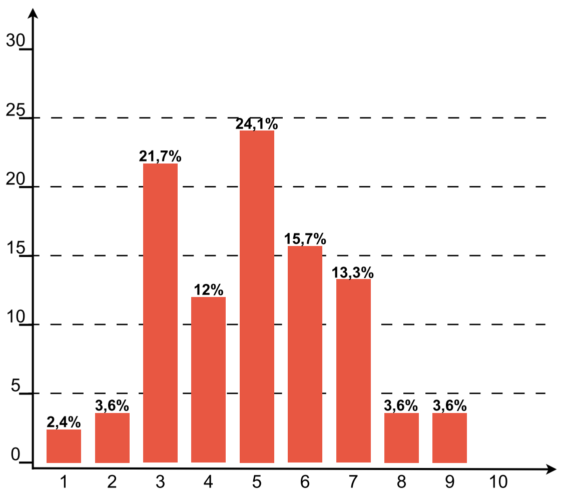
In order to estimate to which extent the artificial dataset is visually similar to CIFAR10, we conducted a user study. We launched a survey asking respondents which of two images they find to be real and which artificial. The survey contained ten image pairs, one per CIFAR10 class, each with one image coming from the original dataset and one from artificial. We also asked participants at the beginning if they were familiar with the CIFAR10 dataset, to test whether that influences the results. For each correctly

Table 6.1: Prompts used to generate artificial dataset.

Class	Positive prompt (number of generated samples)	Negative prompt
airplane	plane photo (5000)	3d, grid, deformed, ugly, mutation, mutated, blurry background, bokeh, multiple images, illustration, cropped, partial view, jpeg artifacts, grayscale
automobile	car photo (2500), automobile photo (2500)	3d, grid, deformed, ugly, mutation, mutated, blurry background, bokeh, multiple images, illustration, cropped, partial view, jpeg artifacts, grayscale
bird	cassowary photo (250), ostrich photo (250), emu photo (250), kiwi bird photo (100), owl photo (100), hawk photo (100), grebe photo (50), loon photo (50), duck photo (50), pheasant photo (100), tern photo (100), hummingbird photo (100), hen photo (50), rooster photo (50), swan photo (50), goose photo (50), parrot photo (100), bustard photo (100), tit photo (200), sparrow photo (200), woodpecker photo (200), pigeon photo (200), cuckoo photo (200), raven photo (200), oriole photo (200), warbler photo (200), chickadee photo (200), starling photo (200), dove photo (200), finch photo (200), nuthatch photo (200), bird photo (500)	3d, bad anatomy, duplicated eyes, no eyes, extra eyes, grid, extra limbs, close up, deformed, ugly, mutation, mutated, blurry background, bokeh, multiple birds, multiple images, illustration, cropped, partial view, duplicated limbs, jpeg artifacts, missing limb, floating limbs, disconnected limbs, black and white, two heads
cat	cat photo (5000)	3d, bad anatomy, duplicated eyes, no eyes, extra eyes, grid, extra limbs, close up, deformed, ugly, mutation, mutated, blurry background, bokeh, multiple cats, multiple images, illustration, cropped, partial view, duplicated limbs, jpeg artifacts, missing limb, floating limbs, disconnected limbs, black and white
deer	deer photo (5000)	3d, bad anatomy, duplicated head, missing head, extra head, grid, extra limbs, close up, deformed, ugly, mutation, mutated, blurry background, bokeh, multiple deers, multiple images, illustration, cropped, partial view, duplicated limbs, jpeg artifacts, missing limb, floating limbs, disconnected limbs, black and white, grayscale, painting, watermark, signature, two heads
dog	dog photo (5000)	3d, bad anatomy, duplicated eyes, no eyes, extra eyes, grid, extra limbs, close up, deformed, ugly, mutation, mutated, blurry background, bokeh, multiple dogs, multiple images, illustration, cropped, partial view, duplicated limbs, jpeg artifacts, missing limb, floating limbs, disconnected limbs, black and white
frog	brown frog photo (2500), green frog photo (2500)	3d, bad anatomy, duplicated eyes, no eyes, extra eyes, grid, extra limbs, close up, deformed, ugly, mutation, mutated, blurry background, bokeh, multiple frogs, multiple images, illustration, cropped, partial view, duplicated limbs, jpeg artifacts, missing limb, floating limbs, disconnected limbs
horse	black horse photo (1000), gray horse photo (1000), chestnut horse photo (1000), bay horse photo (1000), dun horse photo (1000)	3d, bad anatomy, duplicated head, missing head, extra head, grid, extra limbs, close up, deformed, ugly, mutation, mutated, blurry background, bokeh, multiple horses, multiple images, illustration, cropped, partial view, duplicated limbs, jpeg artifacts, missing limb, floating limbs, disconnected limbs, black and white, grayscale, painting, watermark, signature, two heads
ship	watercraft photo (1000), ship photo (3000), sailboat photo (1000)	3d, grid, deformed, ugly, mutation, mutated, blurry background, bokeh, multiple images, illustration, cropped, partial view, jpeg artifacts, grayscale
truck	truck photo (5000)	3d, grid, deformed, ugly, mutation, mutated, blurry background, bokeh, multiple images, illustration, cropped, partial view, jpeg artifacts, grayscale



(a) Not familiar with CIFAR10.



(b) Familiar with CIFAR10.

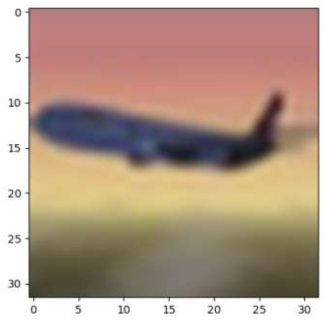
Figure 6.4: Percentage of respondents scored 1-10 points. None of the respondents from both groups got 0 points.

classified real image, we awarded one point. Hence, each respondent could get between 0 and 10 points. We present the distribution of scores in Figure 6.4. In total, we received 241 responses. Figure 6.4a shows scores of 158 respondents who are not familiar with CIFAR10. The distribution of scores is skewed towards the left side, and the average score is 4.72. Figure 6.4b presents responses of 83 respondents familiar with CIFAR10. Although there is a peak at 3 points, most respondents scored 5 or more points with an average of 4.92 points. Knowing CIFAR10 dataset could potentially contribute to recognising the real samples, although more studies are likely necessary. Another point to consider is that the images for the survey were not picked randomly and hence may not completely represent the datasets. Our generative model made many easy-to-spot mistakes, like adding redundant limbs or heads. Therefore, a random selection could lead to obvious tasks in the survey questions, whereas our goal was to investigate how visually similar artificial and original images can get.

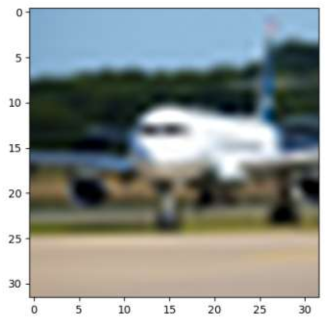
We now demonstrate two example questions from the survey. Figure 6.5 shows the most frequently missed question. Only 47 out of 241 respondents (19.5%) correctly identified the real plane. Figure 6.6 illustrates the question with the highest correct response rate: 165 or 68.5% of respondents correctly identified the real horse. We provide the rest of the survey question in Appendix A. Although the samples for the survey were not selected randomly, our generative model is capable of generating images that are similar for the human eye to the original data.

Failures of generative model

The artificial samples selected for the survey are rather "successful" examples of image generation. Figure 6.7 demonstrates several types of failures the generative model made.

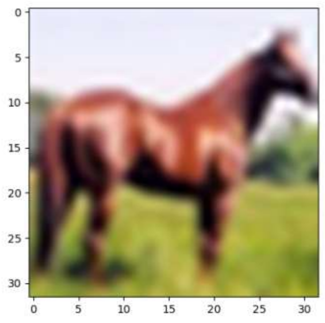


(a) Real plane.

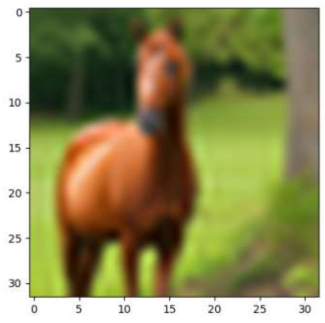


(b) Fake plane.

Figure 6.5: Survey question for the "airplane" class. This question has the highest miss-classification rate among respondents. Only 19.5% of respondents identified the real airplane correctly.



(a) Real horse.



(b) Fake horse.

Figure 6.6: Survey question for the "horse" class. This question is the most often correctly-answered question. The real horse was correctly identified by 68.5% of respondents.

N.b.: the figures provide images in their original size, so one can easily spot poorly generated image parts.

The first category of mistakes mainly concerns animals. Despite keywords such as "bad anatomy" (recommended by prompt creators[2]) and "two heads" in negative prompts, the model sometimes generated mutated animal photos. Figure 6.7a shows a sample from our artificial dataset of a deer with bad anatomy (two heads). Other common failures in this category include generating redundant limbs, missing limbs, or a missing head.

The second category includes images that miss the class object or contain only its small

[2]https://huggingface.co/spaces/stabilityai/stable-diffusion/discussions/7857

unrecognisable part. Figure 6.7b shows a generated image from the "airplane" class. Whereas a human can potentially recognise an aircraft cabin, for a model trained on plane images, that could be a very difficult task. Similarly, sometimes the model generated house pictures for the "cat" class, seemingly assuming that the cat should be somewhere around.

The last category, probably the most harmful, contains images of a wrong object. A dog image in Figure 6.7c is an output of the generative model to the "cat" prompt. Images belonging to the previous category might be seen as just noise in the dataset. In contrast, this category contains wrongly labelled images, which can harm the performance of a model trained on this artificial dataset.

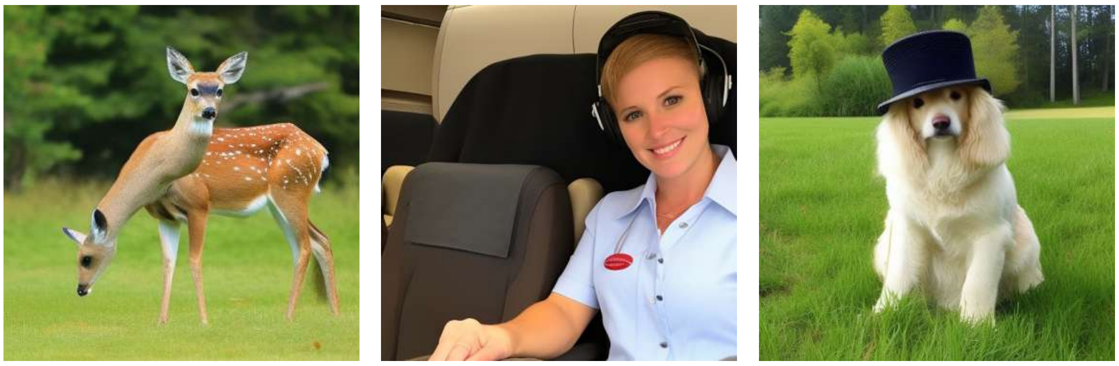

(a) "Deer". Bad anatomy. (b) "Plane". Part of object. (c) "Cat". Wrong object.

Figure 6.7: Failures of generative model.

All three categories of mistakes can negatively impact the quality of the artificial dataset. However, generating a high-quality dataset was not the primary task of this thesis. Therefore, we did not invest additional resources into improving the data quality, which might be an interesting strand for future work.

6.2 Target Model Training

In this section, we introduce the architectures chosen for the target model training. Then, we describe how each model was trained and which hyperparameter configurations were used. Finally, we present the comparison of the target model accuracy to the state-of-the-art results.

6.2.1 Architecture

Similarly to the original dataset selection, we conducted a literature study to gather statistics on related work. We analysed the same 24 papers as mentioned above (Section 3.3.6), and show the most commonly used architectures in Figure 6.8. ResNet-34 [HZRS16] turned out to be the most common choice. Moreover, its complexity is suitable for the CIFAR10 classification task. One of the goals of this thesis is to verify if using transfer learning for training the target model makes it easier to steal. Therefore, we trained two target models with ResNet-34 architecture: one was trained from scratch, and another was trained using transfer learning. In addition, we decided to test one simpler and less typical architecture to imitate the situation when the target model has some secret data-specific architecture. We selected the SimpleNet [HRFS16] architecture for that purpose. It has significantly fewer parameters than ResNet-34 and has not been studied in the model stealing context.

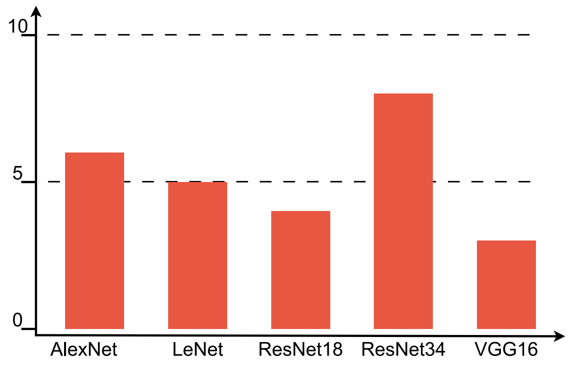


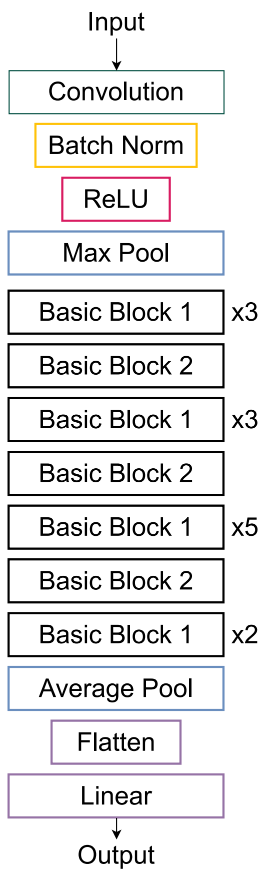
Figure 6.8: Statistics of target model architectures.

ResNet-34

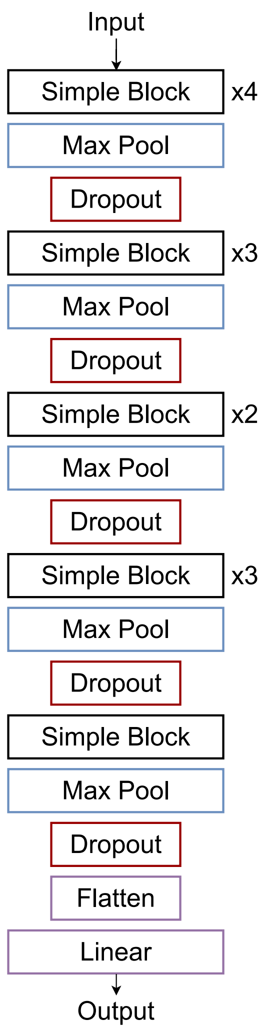
We schematically illustrate the architecture of ResNet-34 in Figure 6.9a. The network uses three types of layers (convolutional, pooling, and linear, i.e. fully connected layer), the ReLU activation function, and batch normalisation as an optimisation technique. Repeated network parts are grouped into blocks: Basic Block 1 and 2. The structure of each block is shown in Figure 6.10. As we can see, both blocks contain residual (skip) connections, convolutional layers, batch normalisation and ReLU activation function. For this work, the last layer of the network is replaced to fit a ten-class classification task. The total number of trainable parameters in our variant of ResNet-34 is 21,289,802.

SimpleNet

The architecture of SimpleNet is shown in Figure 6.9b. As ResNet-34, SimpleNet consists of convolutional, pooling, and fully connected layers. It also contains a repeated part grouped into what we call a *Simple Block* (Figure 6.10). The block consists of a single convolutional layer, on top of which batch normalisation and ReLU activation function are applied. In contrast to ResNet-34, SimpleNet has no skip connections. However, it uses dropout as a regularisation technique after each pooling layer. The total number of trainable parameters is 5,498,378. The network was initially designed as a lightweight alternative to CNNs with significantly more trainable parameters, such as ResNet models.



(a) ResNet-34 architecture [HZRS16].



(b) SimpleNet architecture [HRFS16].

Figure 6.9: Target model architectures.

6.2.2 Training Strategy

There might be two points of view on the importance of the target model performance scores. On the one hand, conceptually, there is no difference between stealing a model with random weights and stealing a model with the same architecture but state-of-the-art performance. In both cases, there is a model with weights, and the goal of stealing is to approximate the behaviour those weights imply. On the other hand, if the target model performs poorly on the original dataset, it is much easier to reach (or even surpass) its accuracy performance. One solution could be to use fidelity as the primary metric. However, as we (1) aim to compare the results of this thesis with related work that often reports accuracy, and (2) use a well-known dataset, we aim to reach accuracy scores

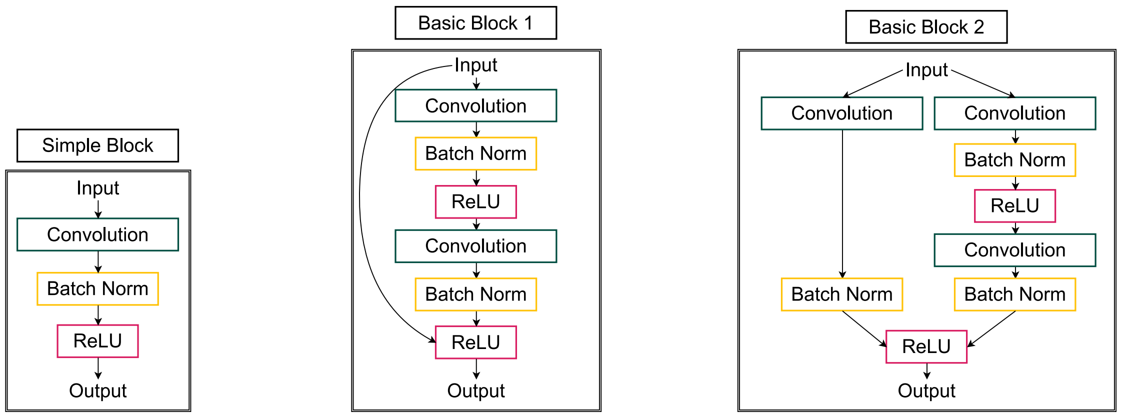


Figure 6.10: CNN building blocks used in Figure 6.9b and Figure 6.9a.

Table 6.2: Hyperparameter grid for target models trained from scratch.

Hyperparameter	Values
Learning rate	0.05, 0.01, 0.005, 0.001, 0.0005, 0.0001
Batch size	32, 64, 128
Optimiser	Adam, SGD with Momentum
Data augmentation	True, False

close to the state-of-the-art. Therefore, we optimise the training hyperparameters of our models. Of course, one has to consider the model complexity and training strategy (transfer learning or training from scratch) while comparing the performance scores.

Training from Scratch

While training from scratch, all weights of a neural network were initialised randomly and trained afterwards simultaneously. We trained both target models (ResNet-34 and SimpleNet) from scratch. In Table 6.2, we list the hyperparameters we optimised, namely learning rate, batch size, optimiser, and whether data augmentation is used, together with candidate values. We performed a full grid search to find the best combination, thus running a total of 72 experiments for each architecture. We trained a model for 25 epochs for each combination of hyperparameters and then selected hyperparameters that resulted in the best-performing model on the validation set. Subsequently, we trained a model with those hyperparameters for 100 epochs, with early stopping determined by the performance on the validation set. We also reduced the learning rate by 0.1 if the performance was not improving for 10 consecutive epochs. Table 6.3 shows the optimal hyperparameters for each architecture, and the number of epochs the final model was trained for.

Table 6.3: Best hyperparameter setting for target models trained from scratch.

Model	Learning Rate	Batch size	Optimiser	Data augmentation	Epochs
SimpleNet	0.001	128	Adam	True	96
ResNet-34	0.05	32	SGD with Momentum	True	65

Table 6.4: Hyperparameter grid for fine-tuning target models trained using transfer learning.

Hyperparameter	Values
Learning rate	0.0001, 0.00005, 0.00001
Batch size	32, 64, 128
Optimiser	Adam, SGD with Momentum
Data augmentation	True, False

Table 6.5: Best hyperparameter setting for ResNet-34 trained using transfer learning.

Training instance	Learning Rate	Batch size	Optimiser	Data augmentation	Epochs
Last layer	0.0005	128	Adam	False	15
All layers	0.00005	32	Adam	True	89

Transfer Learning

For the transfer learning setup, for ResNet-34, we used weights pre-trained on the ImageNet dataset. All layers, except for the last fully-connected layer, which has to be adapted to the number of classes in a dataset, contain those pre-trained weights. Therefore, we trained our network in two stages: at first, only the last layer was trained, and then the weights were all fine-tuned together. Hence, we had to optimise the training hyperparameters for both stages. The grid used for training the last layer of ResNet-34 consists of the same 72 hyperparameter combinations as for the models trained from scratch (Table 6.2). The hyperparameter selection procedure and further training is the same as for models trained from scratch. We compare the performance of different models after 25 epochs and train a model with the selected hyperparameters for 100 epochs with early stopping and learning rate reduction. Then we unfreeze the rest of the network and run another grid search of size 36 with the hyperparameters listed in Table 6.4. The grid for fine-tuning is smaller, as fine-tuning requires, in general, smaller learning rates, which lead to fewer combinations.

We again trained all models for 25 epochs to pick the best hyperparameter combination. Finally, we fine-tune the model with the best-performing hyperparameters for 100 epochs, using early stopping and reducing the learning rate on a plateau, i.e. when no improvement is happening for 10 consecutive epochs. The best hyperparameters for last layer training and final fine-tuning are shown in Table 6.5.

Table 6.6: Performance of benchmarks (1-3) and our target models (4-6).

	Model	Test accuracy	Number of parameters
1	SimpleNet [HRFS16]	95.51%	∼5M
2	ResNet-34[3]	95.4%	∼21M
3	ViT-H/14[DBK+21]	99.5%	∼632M
4	SimpleNet	91.76%	∼5M
5	ResNet-34 (from scratch)	93.61%	∼21M
6	ResNet-34 (transfer learning)	97.14%	∼21M

6.2.3 Performance

We provide the final test scores for each target model in Table 6.6. We also include three benchmark scores: (1) the best score obtained with a SimpleNet [HRFS16] model, (2) the benchmark score of a ResNet-34 model[3] from Hugging Face[4], and (3) the best overall score obtained for CIFAR10 [DBK+21].

We did not manage to reproduce the results of (1) [HRFS16]. Although the authors updated the model implementation and evaluation in 2023 (the target model training of this thesis was finished by then), the evaluation of their model was done in 2016 using a Caffee[5] implementation[6] [HRFS16]. Caffee is not actively developed anymore since 2017, and the code for this thesis is written in Python using PyTorch. Although there was an implementation of SimpleNet in PyTorch published in 2018, the code contained mistakes[7], suggesting that the PyTorch model was not tested by the authors back then. Hence, it is unclear if those results can be reproduced using a current technology stack.

Compared to (2)[3], we obtain a higher score with our pre-trained target model an a lower score with the model trained from scratch.

We did not compare the results of our target models with (3) [DBK+21], as it has 30 times more parameters than our largest model (Table 6.6). We included it to show the best score ever obtained for CIFAR10.

6.3 Defence Configuration

We implemented two proactive defences: input and output perturbation. We had to find defence hyperparameters for each defence and target model so that the added noise modifies no more than 1% of target model top-1 responses. Too little perturbation was also not desirable, as it could not affect an attack as expected. Hence, we selected the hyperparameters that were perturbing (close to) 1% of outputs for both defences. All configurations were evaluated on the CIFAR10 validation set, by measuring the fidelity

[3]https://huggingface.co/edadaltocg/resnet34_cifar10
[4]https://huggingface.co/
[5]https://caffe.berkeleyvision.org/
[6]https://github.com/Coderx7/SimpleNet
[7]https://github.com/Coderx7/SimpleNet_pytorch/issues/5

Table 6.7: Hyperparameter grid for input-perturbation defence.

Hyperparameter	Values
Noise mean μ	0.01, 0.03, 0.05
Noise standard deviation σ	0.001, 0.005, 0.01
Threshold t	0.1, 0.15, 0.2, 0.25, 0.3, 0.35, 0.4, 0.45

Table 6.8: Final hyperparameter configuration for input-perturbation defence and corresponding fidelity scores.

Model	Threshold t	Noise mean μ	Noise std σ	Fidelity
SimpleNet	0.4	0.03	0.01	99%
ResNet-34 (from scratch)	0.4	0.05	0.01	99%
ResNet-34 (transfer learning)	0.25	0.01	0.01	99.12%

of the non-protected to protected target model. If several hyperparameter configurations led to the same optimal fidelity score, we selected the configuration with a larger number of modified pixels.

6.3.1 Input Perturbation

Input perturbation relies on three hyperparameters: a threshold t that defines how many pixels are modified, and the mean μ and standard deviation (std) σ of the noise distribution controlling the intensity of modifications. We run a full grid search with hyperparameter values reported in Table 6.7, trying 72 combinations per target model.

Table 6.8 shows the optimal hyperparameters and a fidelity score for each target model. For ResNet-34 trained using transfer learning, we did not obtain the desired fidelity value. We plot the fidelity scores for this model in Figure 6.11. The highlighted runs are the ones closest to 99% fidelity: the one above the threshold reaches 99.12%, and the two below each 89.68%. We selected the first run, as it is the closest to our desired score.

6.3.2 Output Perturbation

The output perturbation defence requires setting two parameters: a hypercube half-edge length r and a number of samples d. The size of the hypercube impacts the amount of noise added to the target model outputs: the larger it is, the more noise we add. In contrast, increasing the number of samples taken from the hypercube can make noise smoother. However, taking more samples negatively affects the running time of the defence. Hence, we tested different values to determine if the number of samples can be reduced without significantly impacting the defence performance. We also tested the extreme case when all samples lay on the edges of the hypercube in order to get the worst fidelity estimation. We call this scenario "discrete" and the uniformly distributed sampling "uniform". We performed a full grid search with hyperparameter values reported in Table 6.9, running 120 experiments per target model.

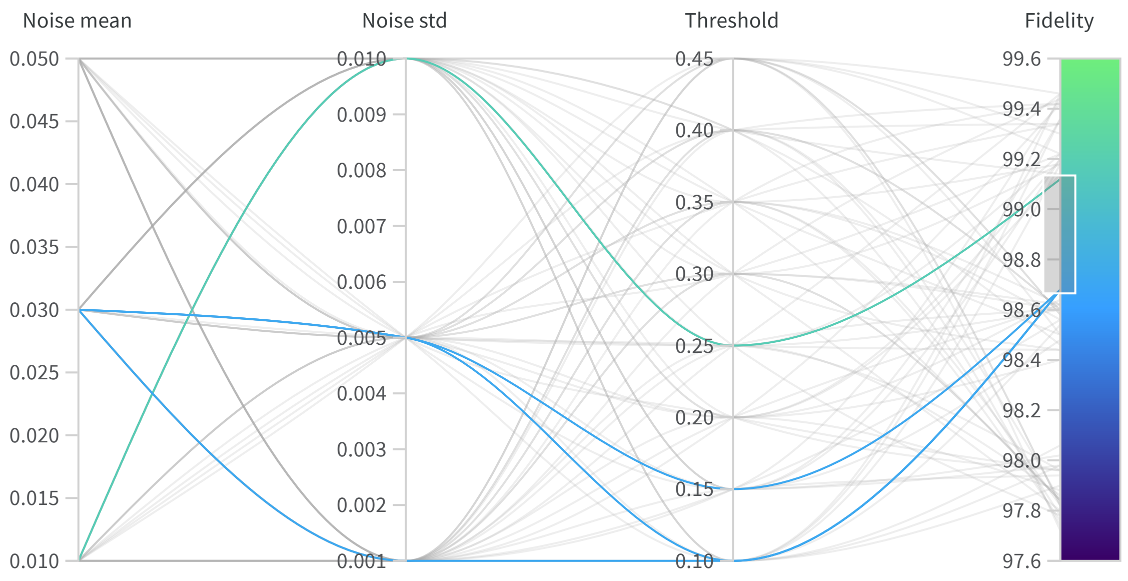


Figure 6.11: Input perturbation fidelity scores for ResNet-34 trained using transfer learning. The highlighted runs are neighbouring to the desired fidelity score of 99%.

Table 6.9: Hyperparameter grid for output-perturbation defence.

Hyperparameter	Values
Sampling	uniform, discrete
Half-edge length r	0.025, 0.05, 0.075, 0.1, 0.125, 0.15, 0.175, 0.2, 0.225, 0.25, 0.275, 0.3, 0.325, 0.35, 0.375, 0.4, 0.425, 0.45, 0.475, 0.5
Number of samples d	10, 25, 100

Table 6.10: Final hyperparameter configuration for output perturbation defence and corresponding fidelity scores.

Model	Length r	Fidelity					
		Uniform sampling			Discrete sampling		
		$d = 10$	$d = 25$	$d = 100$	$d = 10$	$d = 25$	$d = 100$
SimpleNet	0.025	99.26%	99.36%	99.3%	98.9%	98.7%	98.58%
ResNet-34 (from scratch)	0.125	99.1%	99.02%	99%	98.34%	98.38%	98.16%
ResNet-34 (transfer learning)	0.025	98.9%	98.9%	98.96%	98.48%	98.58%	98.52%

We report the optimal r value and fidelity scores for different numbers of samples and both sampling strategies in Table 6.10. For SimpleNet and ResNet-34 trained using transfer learning, the smallest r resulted in the performance closest to the desired one. We can also see that the number of samples selected from the hypercube did not particularly affect the fidelity scores. Therefore, we decided to use 10 samples for further experiments. Using the discrete sampling strategy decreased the fidelity score by 0.84%. We only use the uniform sampling strategy for the rest of the experiments.

6.4 Substitute Model Training

We now describe the configurations used to train the substitute models. We begin by introducing the architectures used for substitute model training, followed by a description of the training hyperparameter optimisation. Then, we report how attack parameters were adapted for the substitute models, and for the attacker's datasets. In particular, we describe optimal configurations for active learning, adversarial augmentation, and the combined active adversarial augmentation attack.

6.4.1 Architecture

As with the target model architecture, we gathered information from related work. Figure 6.12 shows the four most commonly used substitute architectures in papers performing image classifier stealing. The most popular architecture, ResNet-18, was also usually used in the literature to steal ResNet-34, which will be reproduced in this thesis. It is also a suitable option from the complexity point of view: it is simpler than ResNet-34, but more complex than SimpleNet. In addition, it fits one of the scenarios carried out in this thesis: different target and substitute architectures. Besides ResNet-18, we also used SimpleNet and ResNet-34 as substitute models. Transfer learning from ImageNet was used for both residual networks.

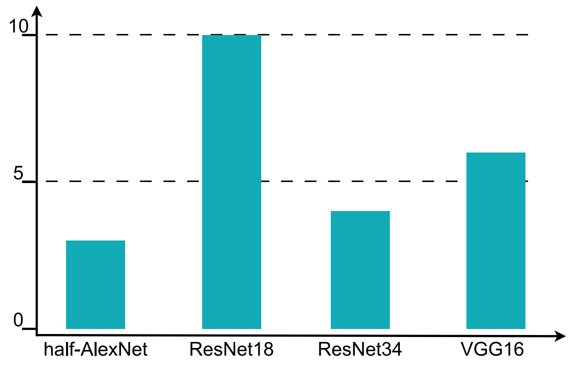


Figure 6.12: Statistics of substitute model architectures.

6.4.2 Training Hyperparameters

Three substitute architectures (SimpleNet, ResNet-18, and ResNet-34) are trained on three attacker datasets (CIFAR10, CINIC10, and artificial). Hence, nine scenarios require training hyperparameter tuning. We narrow the grid compared to the target model training, to keep the number of experiments reasonable for our available computing resources. Firstly, we fix the batch size to 64, and run all experiments without data augmentation. Secondly, we try fewer values for the learning rate. Lastly, we run all experiments using the target model with the SimpleNet architecture, which is the most efficient target model. Table 6.11 summarises the configurations we use for each model.

Table 6.11: Hyperparameter grid for substitute model training.

Model	Hyperparameter	Values
SimpleNet	Learning rate	0.01, 0.001, 0.0001
	Optimiser	Adam, SGD with Momentum
ResNet-18, ResNet-34	Last-layer learning rate	0.01, 0.001, 0.0001
	Last-layer optimiser	Adam, SGD with Momentum
	Fine-tune learning rate	0.0001, 0.00005, 0.00001
	Fine-tune optimiser	Adam, SGD with Momentum

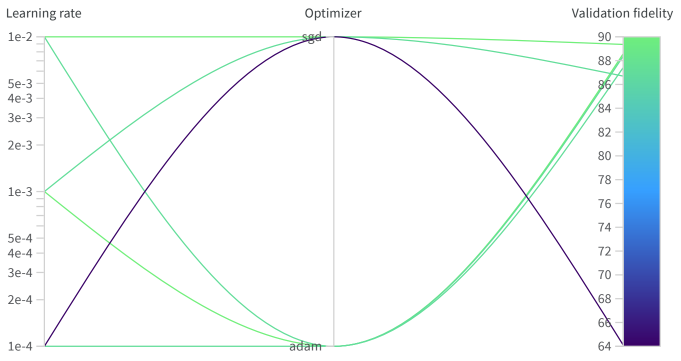
Overall, we run 6 experiments for SimpleNet, and 36 experiments for each of ResNet-18 and ResNet-34.

Each model from the grid is trained on 45,000 train samples for 25 epochs and subsequently evaluated on a validation set. The validation set is always defined by the adversary's training data. Thus, depending on the scenario, it comes from CIFAR10, CINIC10, or the artificial dataset. In general, one needs to add the validation set size into the query budget, as without querying it to the target model, the adversary can not perform hyperparameter tuning.

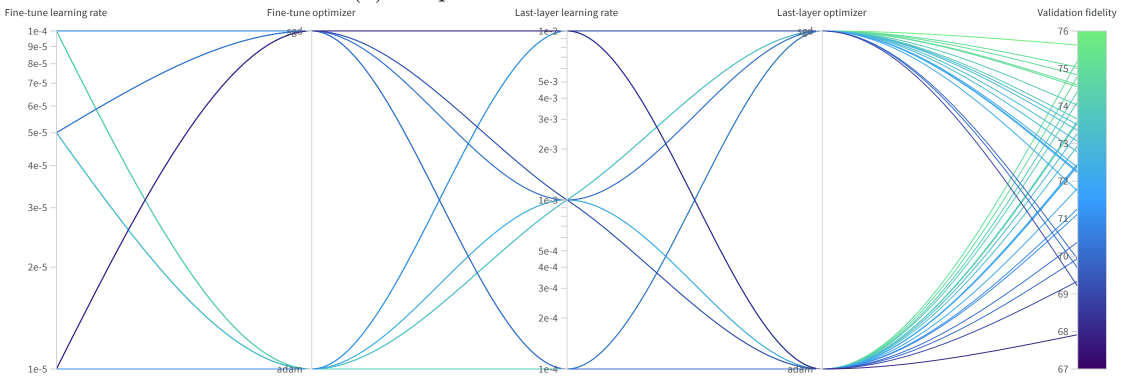
Figure 6.13 demonstrates fidelity scores of models from three different grids. Figure 6.13a shows performance of Simplenet models trained on CIFAR10. As we can see, one hyperparameter setting, namely with sgd optimiser and a learning rate of 0.0001, performs significantly worse than others. Notably, this setting performed poorly also on the other two datasets. The best-performing model reached 89.36% fidelity on the CIFAR10 validation set. Figure 6.13b shows the fidelity scores for ResNet-18 models trained on CINIC10. This dataset is the most complicated for learning, so none of the models reached a fidelity score of 76% or higher. The highest obtained score on the validation set is 75.62%. Finally, Figure 6.13c represents the performance of ResNet-34 models trained on the artificial dataset. For this dataset, we see the smallest difference between the worst- to the best-performing hyperparameters and, in general, the highest validation fidelity scores. The best-performing model reached a score of 90.78%. The artificial dataset is thus likely the easiest of the three attacker's datasets to learn.

Measuring fidelity on the attacker's data is likely the most feasible scenario for substitute hyperparameter tuning. However, if the attacker uses non-original data, the highest-scoring hyperparameters might differ from the ones actually optimal for the stealing task. That's due to the fact that the attacker's data likely comes from a different distribution than the original data, requiring different hyperparameters for optimal learning. To have a fair comparison setup, we evaluate all attacks also on the original CIFAR10 test set in Chapter 7. We also compare test and validation scores to investigate how the attacker's observations differ from the actual performance of the attack.

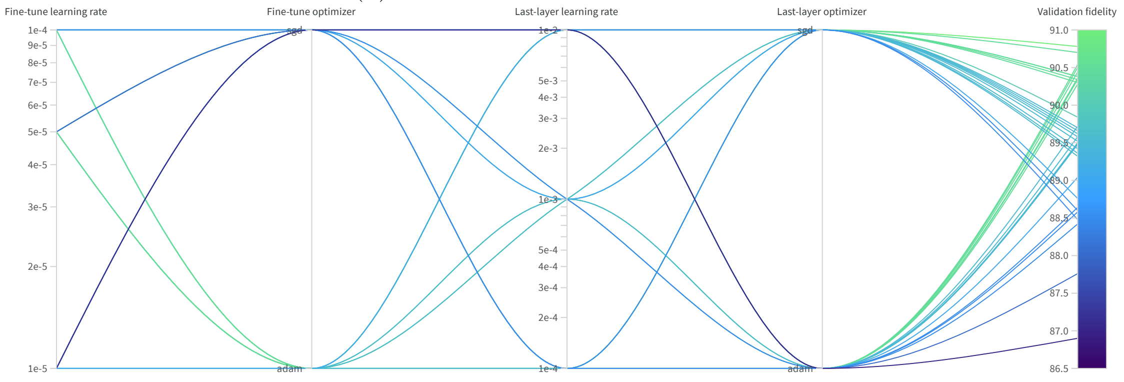
Tables 6.12 and 6.13 provide the best-performing hyperparameters for SimpleNet and ResNet models, respectively. Those hyperparameters are further used for training



(a) SimpleNet trained on CIFAR10.



(b) ResNet-18 trained on CINIC10.



(c) ResNet-34 trained on artificial dataset.

Figure 6.13: Validation fidelity of different training hyperparameter configurations.

substitute models with different query budgets for 100 epochs with early stopping and learning rate reduction on a plateau. In the following, the hyperparameters are tuned for the largest query budget we investigate (45,000 training samples). However, we assume that they are (nearly) optimal also for smaller attacker's training sets, drawn randomly from the full ones used in this section. We report the final scores later in Chapter 7.

Table 6.12: Best training hyperparameter setting for SimpleNet substitute models.

Dataset	Learning rate	Optimiser	Validation fidelity
CIFAR10	0.01	SGD with Momentum	89.36%
CINIC10	0.01	SGD with Momentum	74.60%
Artificial	0.01	SGD with Momentum	89.84%

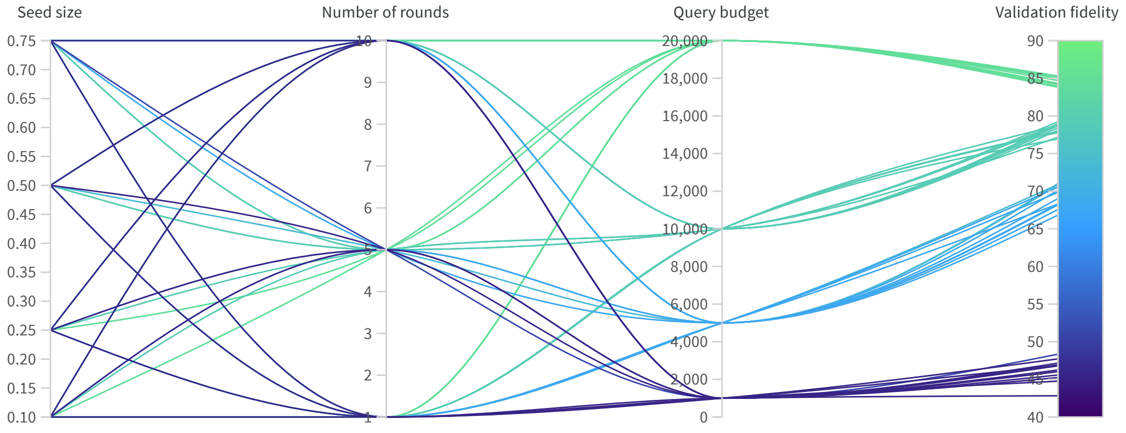
Table 6.13: Best training hyperparameter setting for ResNet-18 and ResNet-34 substitute models.

Model	Dataset	Last-layer learning rate	Last-layer optimiser	Fine-tune learning rate	Fine-tune optimiser	Validation fidelity
ResNet-18	CIFAR10	0.001	Adam	0.0001	Adam	92.34%
	CINIC10	0.0001	SGD with Momentum	0.0001	Adam	75.62%
	Artificial	0.0001	Adam	0.00005	Adam	90.8%
ResNet-34	CIFAR10	0.01	SGD with Momentum	0.00005	Adam	92.2%
	CINIC10	0.0001	SGD with Momentum	0.0001	Adam	74.8%
	Artificial	0.001	SGD with Momentum	0.0001	Adam	90.78%

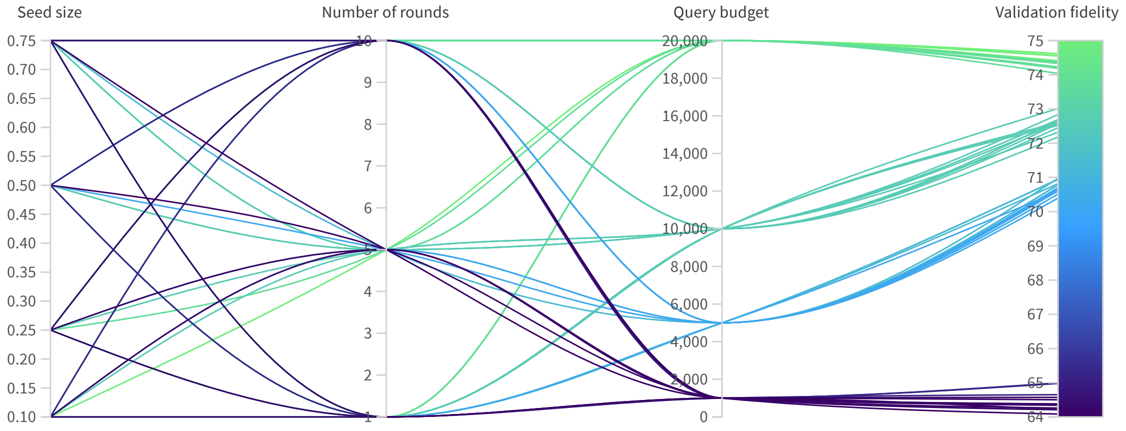
6.4.3 Active Learning

The active learning attack implemented in this thesis (see Algorithm 4.1) requires setting two hyperparameters: the seed size s and the number of rounds r. Previously, only Pal et al. [PGS+20] explored the impact of those hyperparameters on the performance of a substitute model. However, their study was quite restricted, as the authors set the seed size to 10% of the total query budget and only compared two configurations, with 10 and 20 rounds correspondingly. They obtained an increase in performance of 0.03% on the CIFAR10 dataset when using 20 rounds. As this is rather a minor improvement and using more rounds leads to an increase in the attacker's running time, we limited the number of rounds in our experiments by 10. Overall, we experiment with four seed sizes (10%, 25%, 50%, and 75%) and three round counts (1, 5, 10). As active learning selects samples non-randomly, we assumed that there might be different optimal hyperparameter values for different query budgets. Hence, we run a full grid search of 12 active learning hyperparameter combinations for four query budgets (1,000, 5,000, 10,000, and 20,000) for each substitute architecture (SimpleNet, ResNet-18, ResNet-34) and each dataset (CIFAR10, CINIC10, Artificial). We only considered one target model in these experiments, SimpleNet, to make the tuning process more efficient.

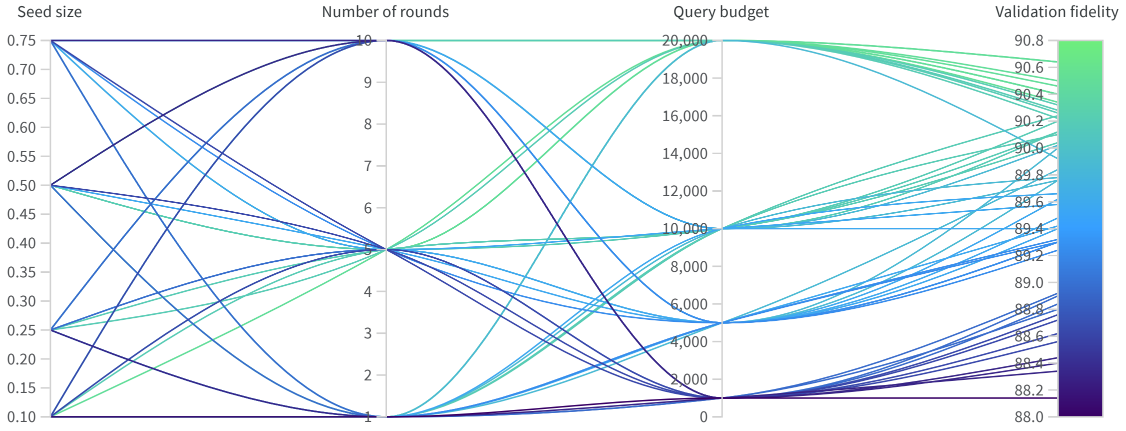
Figure 6.14 shows the impact active learning hyperparameters have on the validation fidelity in three different attack configurations. For CIFAR10 and SimpleNet (Figure 6.14a), the tuning has more effect when the number of queries is smaller. In particular, all validation fidelity scores for 20,000 queries lie between 83.8% and 85.3%, with the largest difference between runs of 1.5%. However, for 1,000 queries, this difference is 5.52%,



(a) SimpleNet trained on CIFAR10.



(b) ResNet-18 trained on CINIC10.



(c) ResNet-34 trained on artificial dataset.

Figure 6.14: Impact of different active learning configurations on validation fidelity.

Table 6.14: Best-performing active learning hyperparameter configurations and corresponding validation fidelity scores.

Model	Dataset	Query budget	Seed size	Number of rounds	Validation fidelity
SimpleNet	CIFAR10	1,000	750	5	48.34%
		5,000	2,500	5	71%
		10,000	2,500	5	79.22%
		20,0000	10,000	1	85.3%
	CINIC10	1,000	100	5	40.15%
		5,000	3,750	5	56.45%
		10,000	2,500	1	62.9%
		20,0000	15,000	5	69.47%
	Artificial	1,000	250	5	70.58%
		5,000	1,250	10	83.66%
		10,000	5,000	5	87.2%
		20,0000	10,000	10	88.9%
ResNet-18	CIFAR10	1,000	500	1	84.5%
		5,000	1,250	5	89.7%
		10,000	1,000	5	91.86%
		20,0000	10,000	5	91.62%
	CINIC10	1,000	100	10	64.99%
		5,000	1,250	5	70.97%
		10,000	2,500	1	73.01%
		20,0000	2,000	10	74.62%
	Artificial	1,000	100	10	89.12%
		5,000	3,750	5	90.08%
		10,000	1,000	5	90.36%
		20,0000	10,000	10	90.84%
ResNet-34	CIFAR10	1,000	100	10	81.38%
		5,000	500	1	89.26%
		10,000	1,000	10	91.24%
		20,0000	2,000	5	91.82%
	CINIC10	1,000	750	1	65.43%
		5,000	3,750	1	69.75%
		10,000	1,000	1	71.16%
		20,0000	2,000	5	73.12%
	Artificial	1,000	500	1	88.92%
		5,000	500	5	90%
		10,000	1,000	1	90.24%
		20,0000	5,000	10	90.64%

and scores range between 42.82% and 48.34%. Figure 6.14b shows fidelity scores for ResNet-18 models trained on CINIC10. For this dataset, the difference in performance for the same query budget remains similar for all cases, and below 1%. In both figures, fidelity scores are always better when the number of queries is higher. Figure 6.14c illustrates a completely different behaviour of ResNet-34 trained on the artificial dataset. For instance, one run with a budget of 5,000 queries outperforms a run with 20,000 queries. Another observation is that the worst-performing combination resulted in 88.14% fidelity, which is notably higher than in CIFAR10 and CINIC10 experiments. That can be another indication for the simplicity of the artificial dataset.

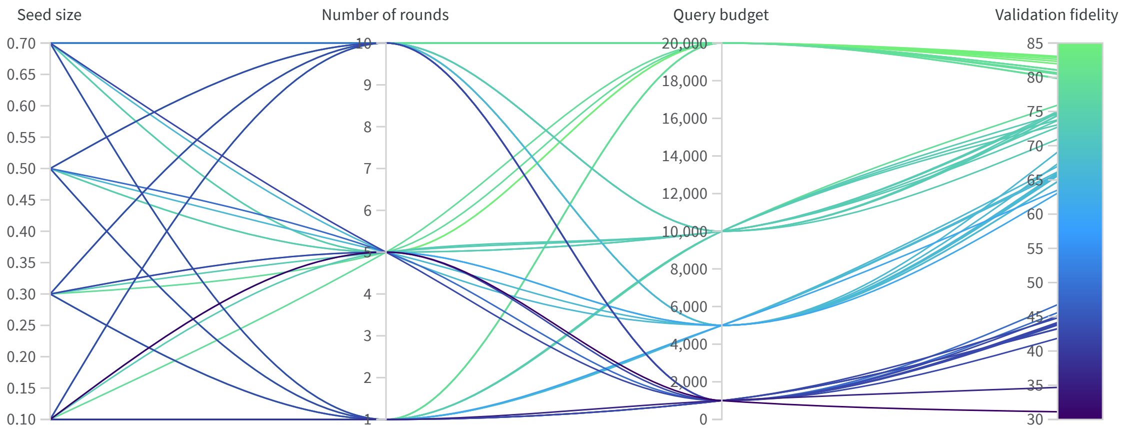
We completed a total of 412 experiments, and selected the 36 best-performing configurations for each substitute architecture, dataset, and query budget. We show the corresponding hyperparameter values and validation fidelity scores in Table 6.14. In this table, we provide the seed size as the absolute number of queries, instead of the percentage of the total query count. As we can see, for the SimpleNet model, increasing the number of queries leads to a notable performance increase. ResNet models gain less than 1% fidelity when the number of queries is increased from 5,000 to 20,000, demonstrating their ability to learn the data using fewer samples than SimpleNet. As we already saw in Figure 6.14c, scores for the artificial dataset are remarkably higher for smaller query budgets, compared to the other datasets.

6.4.4 Adversarial Augmentation

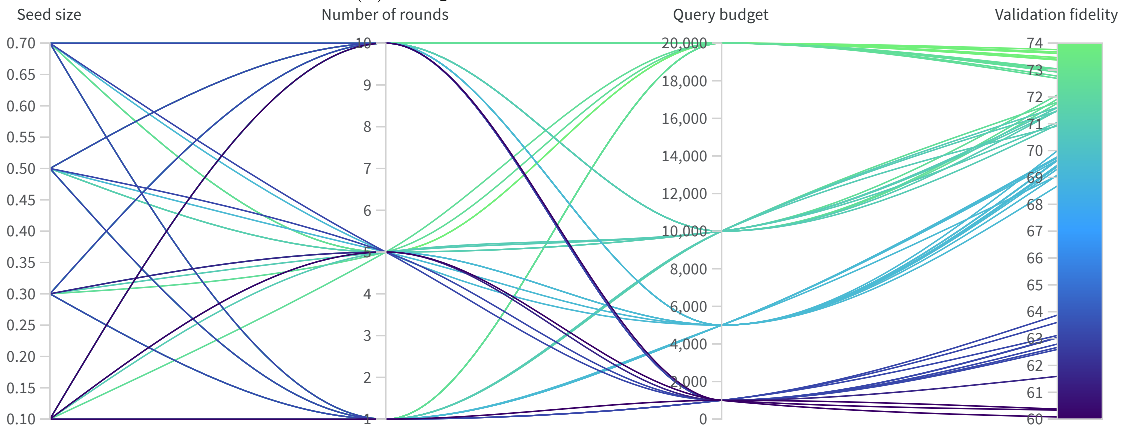
The adversarial augmentation attack (Algorithm 4.2) requires setting the same hyperparameters as the active learning attack, namely the seed size and the number of augmentation rounds. To the best of our knowledge, no previous work report any observations about the impact of these hyperparameters on the attack effectiveness. Initially, we aimed to test the same grid as for the active learning attack. However, for adversarial augmentation, an even number of samples should be added to the attacker's dataset at each round. The reason is that half of those samples should be randomly selected from a pool, and the second half correspond to adversarial examples crafted for those randomly selected samples. Due to this, we had to adjust the seed size values. Hence, we conducted the experiments with the same round count values (1, 5, 10), but slightly modified seed sizes (0.1, 0.3, 0.5, 0.7). As in the previous section, we had 12 experiments per model (SimpleNet, ResNet-18, ResNet-34), dataset (CIFAR10, CINIC10, Artificial), and query budget (1,000, 5,000, 10,000, and 20,000), which means 412 experiments altogether.

Figure 6.15 shows the impact of hyperparameters on the validation fidelity for three model-dataset configurations. The trends are similar to the ones we saw in Figure 6.14. In particular, the performance of different hyperparameters for a fixed number of queries gets more stable with more queries. Besides, we can observe the same difference between the behaviour of ResNet-34 on the artificial dataset (Figure 6.15c) and the behaviours of SimpleNet and RenNet-18 trained on CFAR10 and CINIC10 correspondingly (Figure 6.15a and Figure 6.15b) as we have already seen for active learning. When the query budget is 5,000 or more on the artificial dataset, models trained with less data sometimes outperform models trained with more data. However, there are also several differences from Figure 6.14. Firstly, for each query budget for CIFAR10 and CINIC10, the worst-performing configurations are those where the seed size is 10% of the total number of queries. Secondly, for CINIC10, the difference in hyperparameter performance for the same query count is more diverse, especially for 1,000 queries. Finally, the lowest validation score for all three grids is worse than for the active learning attack, suggesting that this approach might be less effective.

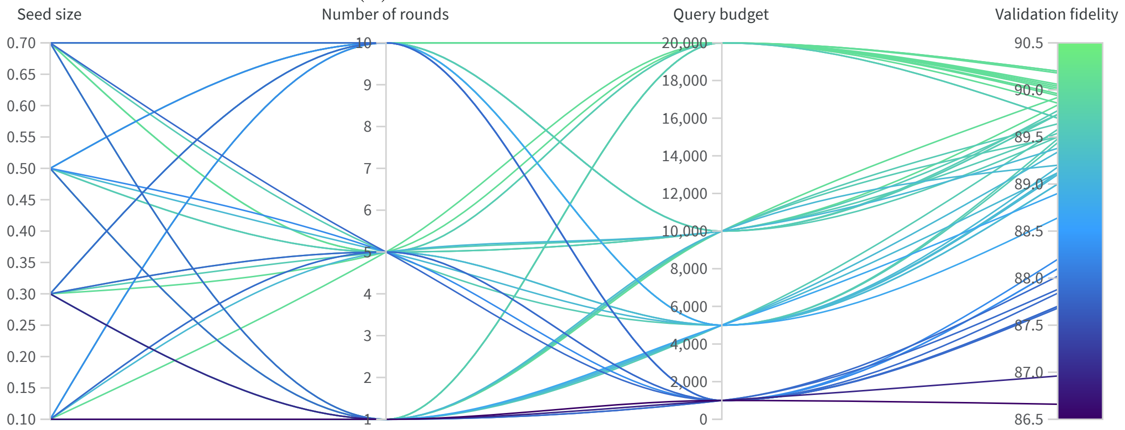
Table 6.15 shows the best-performing hyperparameter configurations and reached validation fidelity scores. Notably, in most cases, the seed size is 70% of the total number of queries. A potential explanation is that the adversarial examples added to the attacker's dataset at each round are less meaningful than the data in the pool. Hence, learning is more effective if the substitute model gets more data from the pool to train on. However,



(a) SimpleNet trained on CIFAR10.



(b) ResNet-18 trained on CINIC10.



(c) ResNet-34 trained on artificial dataset.

Figure 6.15: Impact of different adversarial augmentation configurations on validation fidelity.

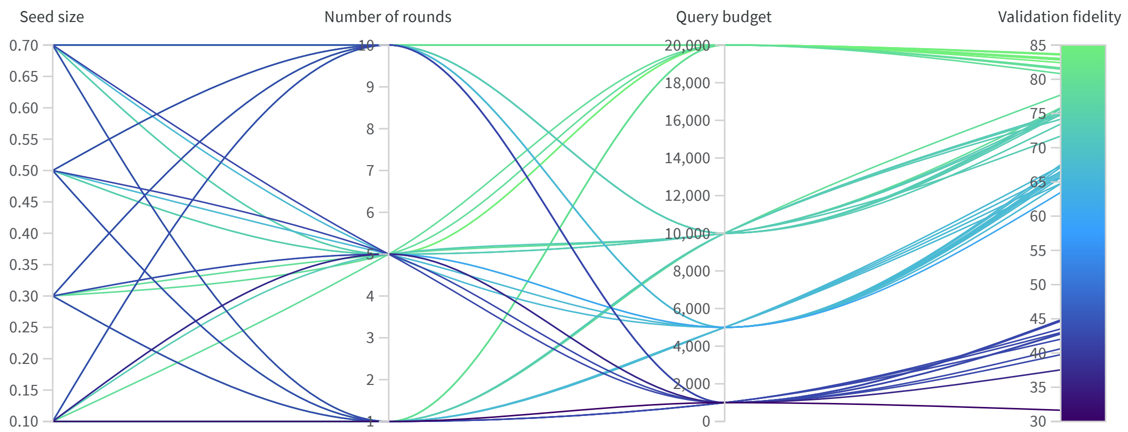
Table 6.15: Best-performing adversarial augmentation hyperparameter configurations and corresponding validation fidelity scores.

Model	Dataset	Query budget	Seed size	Number of rounds	Validation fidelity
SimpleNet	CIFAR10	1,000	700	10	46.78%
		5,000	3,500	5	69.08%
		10,000	7,000	1	75.92%
		20,0000	14,000	5	83.06%
	CINIC10	1,000	700	10	38.77%
		5,000	3,500	10	54.18%
		10,000	7,000	5	62.26%
		20,0000	14,000	5	67.52%
	Artificial	1,000	700	1	68.38%
		5,000	3,500	10	82.04%
		10,000	7,000	1	85.96%
		20,0000	6,000	1	88.24%
ResNet-18	CIFAR10	1,000	700	1	81.92%
		5,000	3,500	10	87.96%
		10,000	7,000	5	90.24%
		20,0000	6,000	10	91.18%
	CINIC10	1,000	700	1	63.87%
		5,000	3,500	5	70%
		10,000	7,000	5	72.07%
		20,0000	10,000	10	73.76%
	Artificial	1,000	500	1	87.98%
		5,000	3,500	5	89.8%
		10,000	7,000	1	90.46%
		20,0000	14,000	1	90.54%
ResNet-34	CIFAR10	1,000	700	1	79.66%
		5,000	3,500	5	87.52%
		10,000	5,000	1	89.4%
		20,0000	10,000	10	91.24%
	CINIC10	1,000	700	1	64.22%
		5,000	3,500	10	68.97%
		10,000	7,000	1	70.64%
		20,0000	14,000	5	72.4%
	Artificial	1,000	100	10	88.2%
		5,000	2,500	1	89.58%
		10,000	3,000	1	89.92%
		20,0000	10,000	1	90.2%

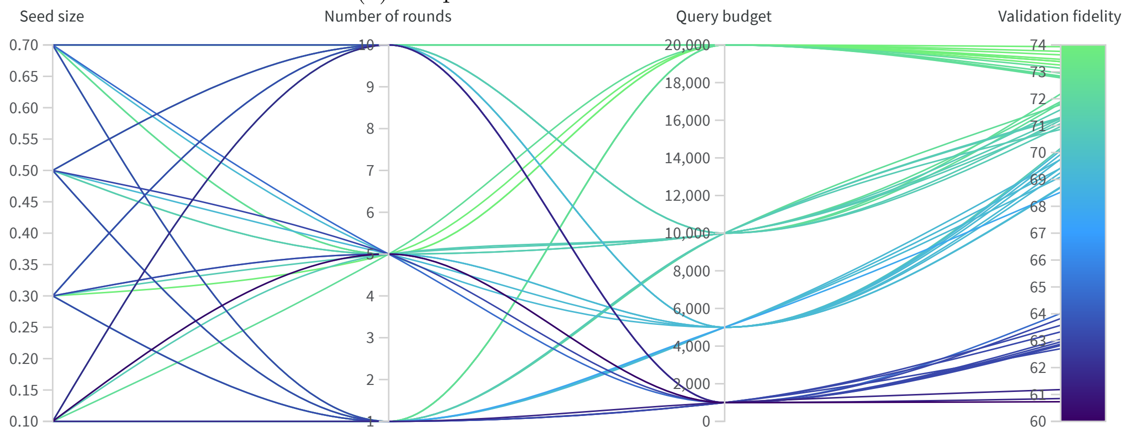
we draw this conclusion solely based on the fidelity measured on the attacker's validation set. We investigate this assumption in the following chapter, where all attacks are evaluated based on their accuracy, fidelity, and transferability on the CIFAR10 test set. Another observation from Table 6.15 is that the scores are lower than in Table 6.14. As assumed above, this attack can be less effective than the active learning attacks.

6.4.5 Active Adversarial Augmentation

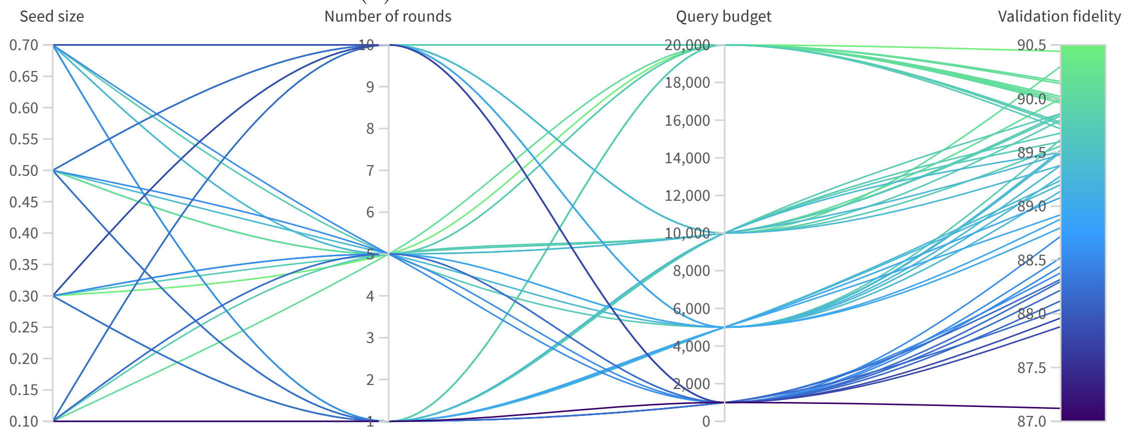
Lastly, we optimised the hyperparameters of our combined active adversarial augmentation attack (Algorithm 4.3), which requires defining the same hyperparameters as active learning and adversarial augmentation. Hence, we used the same grid as for the adversarial augmentation attack, running another 412 experiments.



(a) SimpleNet trained on CIFAR10.



(b) ResNet-18 trained on CINIC10.



(c) ResNet-34 trained on artificial dataset.

Figure 6.16: Impact of different active adversarial augmentation configurations on validation fidelity.

Figure 6.16 presents the impact of different hyperparameters on the performance of three models: SimpleNet trained on CIFAR10, ResNet-18 trained on CINIC10, and ResNet-34 trained on the artificial dataset. The trends are the same as we observed for the adversarial augmentation attack (Figure 6.15), with a slightly improved performance on CINIC10 and the artificial dataset. We show the best hyperparameter configurations and corresponding validation scores in Table 6.16. Similarly to active learning, the optimal seed size is 70% of the total query count in many cases, although for CIFAR10 and ResNet models, this is not the case anymore. The performance is less consistent, sometimes better and sometimes worse, than the active learning and adversarial augmentation attacks. For instance, ResNet-18 trained on 20,000 CIFAR10 samples has 92.22% fidelity when trained with active adversarial augmentation. This score exceeds both the active learning (91.62%) and adversarial augmentation (91.18%) attack performance. At the same time, SimpleNet trained on 5,000 CIFAR10 samples has 71% fidelity when trained with active learning, 69.08% when trained with adversarial augmentation, and only 67.52% when trained with active adversarial augmentation. Such behaviour can be caused by the size of the attacker's data, making this attack more effective when more data is available. We further analyse this behaviour in the next chapter.

Table 6.16: Best-performing active adversarial augmentation hyperparameter configurations and corresponding validation fidelity scores.

Model	Dataset	Query budget	Seed size	Number of rounds	Validation fidelity
SimpleNet	CIFAR10	1,000	700	5	44.84%
		5,000	3,500	5	67.52%
		10,000	7,000	1	77.7%
		20,0000	14,000	1	83.7%
	CINIC10	1,000	500	1	39.07%
		5,000	3,500	1	53.69%
		10,000	7,000	1	61.97%
		20,0000	10,000	1	68.11%
	Artificial	1,000	700	1	69.96%
		5,000	2,500	10	81.86%
		10,000	1,000	5	86.22%
		20,0000	6,000	10	88.76%
ResNet-18	CIFAR10	1,000	500	1	83.12%
		5,000	1,500	5	88.48%
		10,000	5,000	1	90.86%
		20,0000	6,000	10	92.22%
	CINIC10	1,000	700	5	64.02%
		5,000	3,500	10	70.16%
		10,000	7,000	10	72.18%
		20,0000	14,000	10	73.94%
	Artificial	1,000	500	10	88.64%
		5,000	2,500	5	89.96%
		10,000	1,000	5	90.12%
		20,0000	10,000	10	90.56%
ResNet-34	CIFAR10	1,000	500	5	79.74%
		5,000	3,500	1	88.14%
		10,000	7,000	10	90.3%
		20,0000	2,000	10	91.42%
	CINIC10	1,000	700	10	63.9%
		5,000	3,500	10	69.5%
		10,000	7,000	1	70.25%
		20,0000	14,000	1	71.75%
	Artificial	1,000	700	5	88.72%
		5,000	3,500	5	89.62%
		10,000	7,000	10	90.3%
		20,0000	6,000	5	90.44%

6.5 Experiment Summary

So far, we described the selection procedure of hyperparameters relevant to model training, defence and attack setups. We now summarise the experiments conducted to study the effectiveness of attacks and defences in Figure 6.17. For each target model (SimpleNet, ResNet-34 trained from scratch, and ResNet-34 trained using transfer learning), we have one unprotected model and three defence strategies: input perturbation, output perturbation, and input-output perturbation. We attack each target model by training a substitute with three different model architectures (SimpleNet, ResNet-18, and ResNet-34) on three dataset types (CIFAR10/original data, CINIC10/PD data, and artificial/synthetic data) with four query budgets (1,000, 5,000, 10,000, and 20,000), applying four querying strategies (random, active learning, adversarial augmentation, active adversarial augmentation). In total, there are 144 substitute models trained for these 12 target models, resulting in 1,728 experiments. In addition, we run experiments with the "full" query budget of 45,000 queries, which is equal to the size of the CIFAR10 training set. However, for this budget, applying active learning does not make any difference as it leads to picking all 45,000 samples from the training set. For this reason, we do not include this group of experiments in the diagram. We provide a detailed analysis of the experiments in Chapter 7.

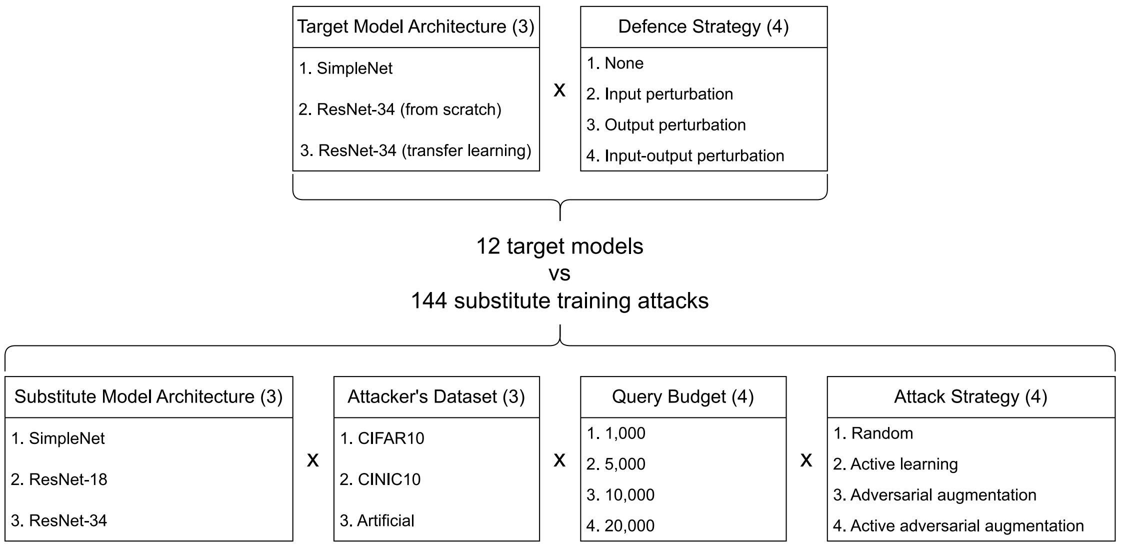


Figure 6.17: Experiment setups carried out in this chapter.

CHAPTER 7

Evaluation of Attacks and Defences

This chapter analyses the experiments shown in Figure 6.17 and is organised as follows. At first, the baseline scores for attacks assuming the strongest attacker's knowledge are provided for each target model and query budget. Afterwards, we explore how the different choices of target and substitute architectures influence the effectiveness of attacks, consequently studying the importance of transfer learning for stealing. We conclude the attack analysis by investigating the impact of the attacker's data source and query optimisation strategy on the attack results. Finally, we examine the effects of data perturbation defences on the attack performance in all attack settings.

7.1 Attack analysis

We analyse the impact of different attack settings on its performance, covering the bottom part of Figure 6.17. We used the hyperparameter settings described in Chapter 6 for all attacks. All substitute models were trained for 100 epochs, with early stopping and reducing learning rate on a plateau. The reported query budgets cover only training data size; for hyperparameter selection and early stopping, the validation set size should also be included.

7.1.1 Baselines

As baseline attacks, we consider the ones with the strongest attacker's knowledge without query optimisation. For each target model, we have 5 baselines of different query budgets, as shown in Table 7.1. We report the joint and the substitute model's accuracy, fidelity, and untargeted transferability on validation and test sets. Below, we analyse how query

Table 7.1: Baseline scores for substitute training attacks. In all experiments, substitute models are trained on the original data (CIFAR10), and have the same architectures as target models.

Target model	Query budget	Validation scores				Test Scores				Target accuracy
		Joint Acc	Acc	Fid	Tr	Joint Acc	Acc	Fid	Tr	
SimpleNet	1k	48.94%	50.86%	50.88%	1.28%	48.07%	50.26%	50.39%	1.52%	91.76%
	5k	70.58%	72.96%	73.36%	2.5%	69.84%	72.01%	73.01%	2.45%	
	10k	77.5%	79.54%	80.72%	3.32%	76.39%	78.84%	79.93%	3.27%	
	20k	83.5%	86.02%	86.84%	5.48%	82.7%	85.38%	86.33%	5.86%	
	45k	88.02%	90.72%	91.36%	8.64%	87.02%	90.27%	90.61%	8.43%	
ResNet-34 (from scratch)	1k	77.5%	79.94%	79.72%	0%	76.69%	79.35%	78.74%	0%	93.61%
	5k	85.54%	88.7%	87.42%	0%	85.32%	88.54%	87.24%	0%	
	10k	89.3%	92.64%	91.12%	0%	88.51%	91.91%	90.51%	0%	
	20k	91.36%	95.32%	92.94%	0.04%	90.73%	94.73%	92.39%	0.01%	
	45k	92.78%	97.22%	94.02%	0.12%	92.15%	96.67%	93.5%	0.05%	
ResNet-34 (transfer learning)	1k	81.44%	82.16%	82.4%	0.02%	80.58%	81.61%	81.79%	0.04%	97.14%
	5k	89.66%	90.36%	90.76%	0.06%	89.47%	90.29%	90.91%	0.1%	
	10k	91.94%	92.84%	92.88%	0.16%	91.3%	92.26%	92.73%	0.17%	
	20k	94.6%	95.38%	95.82%	0.24%	93.77%	94.79%	95.16%	0.28%	
	45k	96.26%	97.16%	97.4%	1.44%	95.37%	96.49%	96.84%	1.22%	

budget, complexity or performance of the target model, and validation scores correlate with the attack performance.

In all cases, increasing the **query budget** leads to an improvement of all reported metrics. In particular, the highest scores are obtained when substitute models are trained on the whole training set of the target model. However, the most notable improvement in accuracy and fidelity scores happens when the query budget is increased from 1,000 to 5,000. The improvement is more significant than any of the following, including the one obtained when the query budget is increased from 5,000 to 45,000. Transferability scores are improved the most when increasing the query budget from 20,000 to 45,000 samples, although even for 45,000 samples, they remain very low.

It is difficult to say which characteristic of the **target model** impacts the difference in attack results the most. The fidelity scores correlate with the performance of the target model – the higher the accuracy of the target model, the higher the fidelity score reached by a substitute model. We speculate that learning mistakes a target model makes is more difficult than learning correctly classified samples. Low transferability suggests that substitute models poorly mimic the behaviour of target models near the decision boundaries. This can indicate difficulties in learning to misclassify the same samples as target models.The learning capabilities of SimpleNet and ResNet-34 are different, and it could happen that SimpleNet can not reach any higher scores. For both variants of training the ResNet-34 target models, the substitute model is the same, and trained using transfer learning. However, the substitute model's accuracy is even higher for the target model trained from scratch, even as the target model mislabelled more samples in this case. The substitute model probably learned to generalise better and was not making as many mistakes as the target model. The transferability scores are higher for the target model trained using transfer learning. Since, in this case, the target and the substitute

Table 7.2: Performance of substitute model training attacks with different target and substitute architectures measured on the CIFAR10 test set.

	Target →	SimpleNet				ResNet-34 (from scratch)				ResNet-34 (transfer learning)			
Substitute ↓	Metric→ QB ↓	Joint Acc	Acc	Fid	Tr	Joint Acc	Acc	Fid	Tr	Joint Acc	Acc	Fid	Tr
SimpleNet	1k	48.07%	50.26%	50.39%	1.52%	48.95%	50.41%	51.01%	1.29%	47.32%	48.08%	48.16%	**0.87%**
	5k	69.84%	72.01%	73.01%	2.45%	70.82%	72.28%	73.64%	1.73%	70.5%	71.44%	71.49%	0.73%
	10k	76.39%	78.84%	79.93%	3.27%	77.88%	79.55%	81.05%	2.53%	78.14%	79.13%	79.27%	0.69%
	20k	82.7%	85.38%	86.33%	5.86%	83.35%	85.15%	86.55%	3.37%	84.32%	85.26%	85.55%	0.75%
	45k	**87.02%**	**90.27%**	**90.61%**	**8.43%**	**88.34%**	**90.45%**	**91.41%**	**4.38%**	**89.08%**	**90.12%**	**90.33%**	0.78%
ResNet-18	1k	78.54%	82.2%	81.31%	0%	79.89%	82.31%	82.5%	0.01%	81.54%	82.56%	82.68%	0.02%
	5k	84.95%	89.33%	87.37%	0.02%	86.25%	89.11%	88.54%	0.01%	87.75%	88.73%	88.94%	0.06%
	10k	87.51%	92.05%	89.95%	0.08%	88.79%	92.01%	91.01%	0.03%	91.07%	92.07%	92.31%	0.09%
	20k	88.72%	93.46%	91.15%	0.05%	90.09%	93.35%	92.4%	0%	92.8%	93.87%	94.13%	0.12%
	45k	**89.84%**	**95.28%**	**91.71%**	**0.18%**	**91.49%**	**95.29%**	**93.4%**	**0.06%**	**94.19%**	**95.39%**	**95.44%**	**0.24%**
ResNet-34	1k	77.72%	81.7%	80.01%	0%	76.69%	79.35%	78.74%	0%	80.58%	81.61%	81.79%	0.04%
	5k	83.93%	88.37%	86.26%	0%	85.32%	88.54%	87.24%	0%	89.47%	90.29%	90.91%	0.1%
	10k	87.6%	92.6%	89.69%	0.02%	88.51%	91.91%	90.51%	0%	91.3%	92.26%	92.73%	0.17%
	20k	89.36%	94.64%	91.32%	0.03%	90.73%	94.73%	92.39%	0.01%	93.77%	94.79%	95.16%	0.28%
	45k	**90.35%**	**96.29%**	**91.95%**	**0.12%**	**92.15%**	**96.67%**	**93.5%**	**0.05%**	**95.37%**	**96.49%**	**96.84%**	**1.22%**

models started training from the same weights, they likely ended up being more similar. However, transferability scores are negligible for both ResNet-34 target models. They are higher for the SimpleNet model, although remaining too low for conducting reasonably successful adversarial attacks.

As the validation set belongs to the CIFAR10 dataset, the **validation scores** of substitute models are very close to the test scores. Accuracy and fidelity are lower on the test set, but the difference is usually less than 1%. Hence, an adversary that uses the original data can quite precisely estimate the performance of the substitute model on the test set.

7.1.2 Substitute Architecture

In this section, we analyse how the choice of architecture for the substitute model impacts the attack performance. In addition, we investigate if there is any difference in stealing performance when the same transfer learning approach is used to train target and substitute models.

For this group of experiments, we use the original CIFAR10 training data as the attacker's dataset, and the random query strategy (i.e. no query optimisation). Hence, there are 3 target models, each attacked by 3 substitute models with 5 different query budgets. Thus, we end up with 45 experiments; their performance scores are reported in Table 7.2. For each experiment, the same metrics as in the previous section are reported: joint accuracy and substitute model's accuracy, fidelity, and untargeted transferability.

Below, we highlight the most notable trends caused by the **substitute architecture** choice.

- SimpleNet has the lowest accuracy and fidelity scores for all target models and query budgets. As stated earlier, this model has the lowest learning capability and probably can not reach higher scores.

- For the smallest query budges of 1,000 samples, ResNet-18 outperforms ResNet-34 in terms of accuracy and fidelity. This trend sometimes persists for up to 20,000 queries.

- ResNet-34 performs the best for 45,000 queries for all target models, suggesting that for big query budgets, picking a model with a higher complexity can be beneficial for an attack.

- SimpleNet reaches the highest transferability scores for target models trained from scratch, compared to the other substitute models. However, while attacking ResNet-34 trained using transfer learning, it has significantly lower scores and performs worse than the ResNet-34 substitute model. Most likely, that is an impact of using transfer learning, which makes the decision boundary of the target model less dataset-specific.

Overall, depending on the query budget, either ResNet-34 or ResNet-18 can be the optimal architecture choice for a model stealing attack. However, we will report all further experiments for ResNet-18 architecture. While having a reasonable (the best for some experiments) performance, ResNet-18 differs from all target architectures. Hence, it corresponds in all experiments to the weaker attacker's knowledge, compared to other substitute architectures. Based on its performance, we can also conclude that having the same architecture as the target model does not imply the best attack performance. Besides, we note that the transferability scores are too low to conduct an adversarial example attack regardless of the architecture used. For this reason, we do not consider transferability while selecting the optimal architecture. For SimpleNet or ResNet-34 substitute models, we will discuss those results that exhibit behaviour that diverges from that of ResNet-18.

Regarding the usage of **transfer learning** for training the target model, we have the following observations. Except for the SimpleNet substitute model, the fidelity scores are higher when the pre-trained ResNet-34 is attacked. It could be the case that the cause of such behaviour is not transfer learning, but the fact that the pre-trained model performs better on the CIFAR10 dataset. For the larger query budgets (20,000 and 45,000), the performance of the attack is similar for both ResNet-34 target models. For smaller budgets, stealing ResNet-34 trained from scratch is more challenging. For instance, with a query budget of 1,000 samples, the attack is even less effective than stealing the SimpleNet target model. Remarkably, using SimpleNet as substitute model, stealing the ResNet-34 trained from scratch is easier in terms of fidelity scores. Based on these observations, we can conclude that substitute models trained with the same strategy as the target model (either with transfer learning or without) have higher fidelity scores than those trained with a different strategy.

Table 7.3: Performance of substitute model training attacks with different attacker's datasets measured on the CIFAR10 test set.

	Target →	SimpleNet				ResNet-34 (from scratch)				ResNet-34 (transfer learning)			
Dataset ↓	Metric→ QB ↓	Joint Acc	Acc	Fid	Tr	Joint Acc	Acc	Fid	Tr	Joint Acc	Acc	Fid	Tr
CIFAR10	1k	78.54%	82.2%	81.31%	0%	79.89%	82.31%	82.5%	0.01%	81.54%	82.56%	82.68%	0.02%
	5k	84.95%	89.33%	87.37%	0.02%	86.25%	89.11%	88.54%	0.01%	87.75%	88.73%	88.94%	0.06%
	10k	87.51%	92.05%	89.95%	0.08%	88.79%	92.01%	91.01%	0.03%	91.07%	92.07%	92.31%	0.09%
	20k	88.72%	93.46%	91.15%	0.05%	90.09%	93.35%	92.4%	0%	92.8%	93.87%	94.13%	0.12%
	45k	**89.84%**	**95.28%**	**91.71%**	**0.18%**	**91.49%**	**95.29%**	**93.4%**	**0.06%**	**94.19%**	**95.39%**	**95.44%**	**0.24%**
CINIC10	1k	65.65%	68.60%	68.76%	0%	69.14%	71.32%	71.49%	0.01%	72.19%	73.2%	73.41%	0.05%
	5k	76.33%	79.6%	79.41%	0.01%	79.79%	82.19%	82.24%	0.01%	82.78%	83.81%	84.06%	0.03%
	10k	80.1%	83.48%	83.23%	0%	82.25%	84.83%	84.69%	0%	85.09%	86.13%	86.38%	0.01%
	20k	81.91%	85.21%	85.4%	0.01%	84.84%	87.32%	87.52%	0.01%	87.9%	89.01%	89.12%	0.06%
	45k	**84.28%**	**87.59%**	**87.83%**	**0.04%**	**86.88%**	**89.43%**	**89.64%**	**0.03%**	**90.42%**	**91.44%**	**91.76%**	**0.1%**
Artificial	1k	63.28%	66.2%	65.86%	0%	64.13%	66.06%	66.32%	0%	66.05%	66.83%	67.1%	**0.03%**
	5k	67.9%	71.11%	70.55%	0%	67.3%	69.44%	69.52%	0%	71.12%	71.97%	72.14%	0.01%
	10k	69.93%	73.04%	72.86%	0%	71.38%	73.72%	73.4%	0%	69.58%	70.46%	70.63%	0.02%
	20k	73.43%	76.72%	76.42%	0.01%	73.96%	76.31%	76.19%	**0.01%**	68.08%	69.01%	69.16%	0.01%
	45k	**75.69%**	**78.88%**	**78.78%**	**0.02%**	**74.94%**	**77.1%**	**77.28%**	0%	**72.96%**	**73.81%**	**74%**	**0.03%**

7.1.3 Attacker's Dataset

We now study how limited availability of data impacts the performance of attacks. In particular, we evaluate our novel data-free attack using artificial data, and compare it with attacks exploiting original and problem-domain data. For each dataset, a substitute ResNet-18 model is trained with five different query budgets, aiming to steal three target models. Overall, there are 45 experiments for which we report attack performance scores in Table 7.3. We summarise our key observations below.

- For all query budgets, the accuracy and fidelity scores are the highest for CIFAR10, and the lowest for the data-free attack using the artificial dataset. Moreover, using only 1,000 CIFAR10 queries results in a more effective attack than using all 45,000 queries from the artificial dataset.
- For CINIC10 dataset, where the attacks effectiveness lies between the two other datasets, we observe similar trends as for CIFAR10: the better the performance of the target model, the higher the accuracy and fidelity scores of the substitute model. Another common observation is that an increase in the query budget always leads to an increase in performance (except for transferability, which is however very low and not indicative).
- For both CIFAR10 and CINIC10 datasets, the highest substitute's accuracy and fidelity scores correlate with the performance of the target model. The highest scores are reached when attacking the pre-trained ResNet-34 and the lowest when attacking the SimpleNet model.
- In contrast, using the artificial data, the best-performing substitute is trained based on labels obtained from the SimpleNet target model. Moreover, contrary to CINIC10, increasing the query budget leads to a drop in performance: for ResNet-34 trained using transfer learning, a substitute trained with 5,000 queries outperforms

Table 7.4: Comparison of validation and test scores of attacks using CINIC10 dataset.

Target model	Query budget	Validation scores				Test Scores				Target accuracy
		Joint Acc	Acc	Fid	Tr	Joint Acc	Acc	Fid	Tr	
SimpleNet	1k	51.34%	61.22%	66.28%	0.03%	65.65%	68.60%	68.76%	0%	91.76%
	5k	54.98%	65.51%	70.34%	0.03%	76.33%	79.6%	79.41%	0.01%	
	10k	56.44%	66.99%	72.2%	0.04%	80.1%	83.48%	83.23%	0%	
	20k	56.95%	66.49%	74.01%	0.08%	81.91%	85.21%	85.4%	0.01%	
	45k	57.7%	67.24%	76.02%	0.12%	84.28%	87.59%	87.83%	0.04%	
ResNet-34 (from scratch)	1k	55.21%	64.3%	67.7%	0.01%	69.14%	71.32%	71.49%	0.01%	93.61%
	5k	58.99%	68.36%	72.16%	0.01%	79.79%	82.19%	82.24%	0.01%	
	10k	60.7%	70.25%	74.13%	0.01%	82.25%	84.83%	84.69%	0%	
	20k	61.4%	70.23%	75.89%	0.02%	84.84%	87.32%	87.52%	0.01%	
	45k	62.68%	71.02%	77.88%	0.06%	86.88%	89.43%	89.64%	0.03%	
ResNet-34 (transfer learning)	1k	62.09%	67.16%	72.18%	0.08%	72.19%	73.2%	73.41%	0.05%	97.14%
	5k	67.87%	73.22%	78.65%	0.1%	82.78%	83.81%	84.06%	0.03%	
	10k	69.53%	74.83%	80.73%	0.12%	85.09%	86.13%	86.38%	0.01%	
	20k	70.45%	75.79%	81.83%	0.3%	87.9%	89.01%	89.12%	0.06%	
	45k	71.96%	77.5%	83.48%	0.56%	90.42%	91.44%	91.76%	0.1%	

the models trained with 10,000 and 20,000 queries. However, we only spot such behaviour for this target model. It is an open question if its cause is the dataset or the target model.

- The transferability scores are negligible in all experiments.

For further analysis, we compare scores obtained on the attacker's validation set with the CIFAR10 test scores. This comparison shows the difference between the performance estimation of the attack by the adversary, and the eventual real success of the attack. As we have seen before in Table 7.1, for the CIFAR10 attacker's dataset, validation and test scores are very similar. We now compare these scores for CINIC10 and the artificial dataset. For CINIC10 (shown in Table 7.4), the performance on the attacker's validation set is notably lower than on the CIFAR10 test set. Hence, using CINIC10, an adversary tends to underestimate the power of their attacks. In contrast, the scores measured on the artificial validation set are way higher than the test scores (see Table 7.5). For example, with a query budget of 1,000, all substitute models reached accuracy over 93% on the validation set. With the same query budget, substitutes trained on the CIFAR10 data (Table 7.1) reached at most 82% of accuracy on the validation set. This further indicates that the artificial data is not complex enough, and its quality has to be improved. Using artificial data, an adversary will likely overestimate the performance of the attack on the real data.

Experimenting with other substitute architectures, we observe further interesting results. We already saw in Table 7.3 that with artificial data, the performance of substitutes is higher when stealing simpler models. In those experiments, we used ResNet-18 as substitute model. Using SimpleNet as substitute, we observe a similar behaviour, but with an even more remarkable difference in fidelity scores, shown in Table 7.6. With 45,000 queries, the substitute model reached 68.35% fidelity while attacking the SimpleNet target, but only 51.01% while attacking the ResNet-34 trained using transfer learning.

Table 7.5: Comparison of validation and test scores of attacks using artificial dataset.

Target model	Query budget	Validation scores				Test Scores				Target accuracy
		Joint Acc	Acc	Fid	Tr	Joint Acc	Acc	Fid	Tr	
SimpleNet	1k	85.84%	93.18%	88.66%	0.02%	63.28%	66.2%	65.86%	0%	91.76%
	5k	86.76%	94.26%	89.76%	0%	67.9%	71.11%	70.55%	0%	
	10k	86.46%	93.46%	89.82%	0.02%	69.93%	73.04%	72.86%	0%	
	20k	86.7%	93.58%	90.22%	0%	73.43%	76.72%	76.42%	0.01%	
	45k	86.86%	93.34%	90.8%	0.04%	75.69%	78.88%	78.78%	0.02%	
ResNet-34 (from scratch)	1k	89.14%	94.58%	91.22%	0%	64.13%	66.06%	66.32%	0%	93.61%
	5k	90.14%	95.74%	92.24%	0%	67.3%	69.44%	69.52%	0%	
	10k	89.88%	95.06%	92.38%	0%	71.38%	73.72%	73.4%	0%	
	20k	90.34%	95.58%	92.84%	0%	73.96%	76.31%	76.19%	0.01%	
	45k	90.12%	95.08%	92.94%	0%	74.94%	77.1%	77.28%	0%	
ResNet-34 (transfer learning)	1k	94.42%	95.94%	95.5%	0.02%	66.05%	66.83%	67.1%	0.03%	97.14%
	5k	95.28%	96.92%	96.46%	0.02%	71.12%	71.97%	72.14%	0.01%	
	10k	95.82%	97.38%	97.1%	0.04%	69.58%	70.46%	70.63%	0.02%	
	20k	95.9%	97.58%	97.1%	0.04%	68.08%	69.01%	69.16%	0.01%	
	45k	96.2%	97.74%	97.5%	0.08%	72.96%	73.81%	74%	0.03%	

Table 7.6: Fidelity scores of SimpleNet substitute model trained on artificial dataset.

	Fidelity					Target accuracy
Target model	1k	5k	10k	20k	45k	on validation set
SimpleNet	23.1%	40.77%	48.61%	58.12%	**68.35%**	88.46%
ResNet-34 (from scratch)	24.58%	37.78%	43.49%	54.81%	68.34%	91.4%
ResNet-34 (transfer learning)	23.57%	35.04%	38.28%	44.95%	51.01%	96.78%

To understand the cause of this behaviour, we also measured the accuracy of the target model on the validation set, and report it in the rightmost column of Table 7.6. There is almost a 10% difference in the accuracy of SimpleNet and the pre-trained ResNet-34 model, suggesting that these models could label the attacker's data quite differently. However, the model that better classifies the attacker's data returns less relevant labels for the original classification task and misleads the substitute training.

7.1.4 Query Optimisation

So far, we mainly investigated the impact of the attacker's knowledge on the performance of attacks. In this section, we evaluate query optimisation methods that can be applied by any attacker, regardless of their knowledge. In particular, we evaluate active learning, adversarial augmentation, and active adversarial augmentation. To demonstrate their impact on the attacks, in Table 7.7, we report the performance scores of the attacks optimised for the scenario of the weakest attacker. All attacks in the table are performed with a ResNet-18 substitute model trained on the artificial dataset. In each cell in Table 7.7, we also provide the difference to the scores from Table 7.3, which contained attacks with the same settings but without any optimisation techniques. Therefore, a positive number means the optimisation technique improved the performance, while negative numbers indicate a degradation of the attack's performance. For each target model and query budget, we highlighted the best-performing score among the optimisation

Table 7.7: Performance of the substitute model training attacks with different optimisation techniques measured on the CIFAR10 test set. Below each score, we report the difference with the non-optimised attack.

	Target →	SimpleNet				ResNet-34 (from scratch)				ResNet-34 (transfer learning)			
Query optimisation ↓	Metric→ QB ↓	Joint Acc	Acc	Fid	Tr	Joint Acc	Acc	Fid	Tr	Joint Acc	Acc	Fid	Tr
Active learning	1k	**61.5% -1.78%**	**64.62% -1.58%**	**64.09% -1.77%**	**0.02% +0.02%**	**66.11% +1.98%**	**68.3% +2.24%**	**68.24% +1.92%**	0% +0.00%	**67.13% +1.08%**	**68.02% +1.19%**	**68.18% +1.08%**	0.01% -0.02%
	5k	**67.57% -0.33%**	**70.65% -0.46%**	**70.39% -0.16%**	0% +0.00%	70.46% +3.16%	72.75% +3.31%	72.66% +3.14%	0% +0.00%	**68.95% -2.17%**	**69.86% -2.11%**	**69.99% -2.15%**	0.01% +0.00%
	10k	**73.31% +3.38%**	**76.64% +3.6%**	**76.14% +3.28%**	0% +0.00%	**76.13% +4.75%**	**78.51% +4.79%**	**78.55% +5.15%**	0% +0.00%	**74.48% +4.9%**	**75.39% +4.93%**	**75.61% +4.98%**	0.01% -0.01%
	20k	**74.81% +1.38%**	**77.98% +1.26%**	**77.65% +1.23%**	**0.02% +0.01%**	**76.42% +2.46%**	**78.91% +2.6%**	**78.79% +2.6%**	0% -0.01%	72.58% +4.5%	73.47% +4.46%	73.62% +4.46%	**0.05% +0.04%**
Adversarial augmentation	1k	60.61% -2.67%	63.6% -2.6%	63.03% -2.83%	**0.02% +0.02%**	63.69% -0.44%	65.69% -0.37%	65.83% -0.49%	0% +0.00%	63.75% -2.3%	64.45% -2.38%	64.78% -2.32%	0% -0.03%
	5k	64.91% -2.99%	68.16% -2.95%	67.53% -3.02%	**0.04% +0.04%**	68.26% +0.96%	70.75% +1.31%	70.46% +0.94%	**0.01% +0.01%**	67.98% -3.14%	68.84% -3.13%	68.92% -3.22%	**0.04% +0.03%**
	10k	66.24% -3.69%	69.47% -3.57%	69.07% -3.79%	0.01% +0.01%	73.92% +2.54%	76.16% +2.44%	76.24% +2.84%	**0.01% +0.01%**	69.42% -0.16%	70.28% -0.18%	70.44% -0.19%	**0.04% +0.02%**
	20k	72.34% -1.09%	75.39% -1.33%	75.26% -1.16%	0% -0.01%	74.04% +0.08%	76.29% -0.02%	76.48% +0.29%	0% -0.01%	71.16% +3.08%	72.04% +3.03%	72.14% +2.98%	0.01% +0.00%
	45k	74.9% -0.79%	78.05% -0.83%	77.73% -1.05%	**0.01% -0.01%**	**75.76% +0.82%**	**77.96% +0.86%**	**78.18% +0.9%**	0% +0.00%	73.51% +0.55%	74.39% +0.58%	74.54% +0.54%	**0.1% +0.07%**
Active adversarial augmentation	1k	60.52% -2.76%	63.53% -2.67%	63.03% -2.83%	0.01% +0.01%	63.43% -0.7%	65.71% -0.35%	65.44% -0.88%	0% +0.00%	63.49% -2.56%	64.23% -2.6%	64.49% -2.61%	**0.03% +0.00%**
	5k	65.29% -2.61%	68.58% -2.53%	67.73% -2.82%	0% +0.00%	**72% +4.7%**	**74.34% +4.9%**	**74.2% +4.68%**	**0.01% +0.01%**	68.46% -2.66%	69.39% -2.58%	69.45% -2.69%	0.01% +0.00%
	10k	70.76% +0.83%	73.84% +0.8%	73.71% +0.85%	**0.02% +0.02%**	73.44% +2.06%	75.7% +1.98%	75.72% +2.32%	**0.01% +0.01%**	73.67% +4.09%	74.57% +4.11%	74.86% +4.23%	0.02% +0.00%
	20k	71.75% -1.68%	74.84% -1.88%	74.62% -1.8%	0.01% +0.00%	75.54% +1.58%	77.9% +1.59%	77.88% +1.69%	0% -0.01%	**73.58% +5.5%**	**74.41% +5.4%**	**74.74% +5.58%**	0.03% +0.02%
	45k	**76.3% +0.61%**	**79.42% +0.54%**	**79.16% +0.38%**	0% -0.02%	75.24% +0.3%	77.41% +0.31%	77.58% +0.3%	**0.01% +0.01%**	**74.57% +1.61%**	**75.5% +1.69%**	**75.71% +1.71%**	0.06% +0.03%

techniques, although sometimes these scores are lower than the baseline from Table 7.3. Also, there is no reported active learning attack with 45,000 queries, as that is the size of the whole dataset, and applying active learning here does not change anything. We list below the main insights from Table 7.7 regarding the accuracy and fidelity scores, as transferability remains too low.

- In most of the configurations, active learning performs the best, although, for the smaller query budgets (1,000 and 5,000), it does not always improve the baseline, and in a few cases, other optimisation techniques reach higher scores. However, active learning improves the attack and holds the second-best score in all such cases.
- In all configurations with a query budget of 10,000 and more, there is an optimisation technique that improves an attack. Hence, applying those techniques, especially active learning, is most useful for mid-range query budgets. We assume that smaller query budgets (1,000 and 5,000) are not enough for a substitute model to learn how to rank the most valuable samples.
- Increasing the number of queries does not always lead to a better attack performance. We have already seen this behaviour for the artificial dataset earlier. However, even with a "smarter" query selection strategy, this property of the dataset does not vanish.

- Although the scores were improved in most cases, they remain notably lower compared to problem-domain and original datasets.

7.1.5 Comparison with the State-of-the-art

We conclude our attack analysis by comparing the performance of the attacks to the state-of-the-art. As we shown in Section 3.3.6, 24 papers perform image classifier stealing and are relevant to our work. However, comparing attacks that steal target models trained on different datasets is not straightforward, as one also needs to consider the classification task complexity. Thus, our comparison is only against papers using CIFAR10 to train their target models. We summarise the performance scores in Table 7.8. Additionally, we provide information about the attacker's strength for each work. If some information was not reported, we mark a corresponding cell as N/A. Regarding the substitute architecture, we mark with ✗ cases when the architecture differs from the target, and with ✓ if it is the same as the target, and ✓✗ if both options were considered. For the last category, we report scores obtained when the architectures are the same, i.e. when the adversary is stronger. Note that for our work, we report scores for ResNet-18 as a substitute, hence we mark the corresponding cells with ✗. Some values are estimated based on the information provided in the papers:

- The target and substitute accuracy for [MSDH19] and substitute accuracy for [YHL$^+$22] was estimated from their plots, as no actual scores were reported.
- The number of queries used by [BCIP20] was not reported. However, the authors used CIFAR100, which contains 50,000 samples, as a *starting point* for their evolutionary algorithm that creates new query images. They claimed that optimising the query budget was not a priority, so we can assume that the number of queries could be significantly larger than 50,000. However, as that is only a speculation, we conservatively estimate that they used "more than 50,000" samples.
- The authors of [MDN19] also do not provide information about the total number of queries. However, they have an iterative algorithm, which generates 1,000,000 samples at each round. Assuming that there should be at least one round, we estimated the number of queries as "larger than 1,000,000".
- The query budget was also not given by [YDZ$^+$22]. However, the authors estimated the price of their attack on Amazon Web Services[1] to be \$360,000. Pricing from 2023 suggests that the price per query for the first million queries is usually around \$0.001, getting cheaper for the subsequent millions. That means that a million queries cost \$1000, and with this price, the authors could have made more than 300 million queries. However, since the prices could differ back then, we lowered our estimate to 1 million.

[1]https://aws.amazon.com/rekognition/pricing/

- Some of the papers did not report the architecture of the target model or used a custom architecture without reporting the number of trainable parameters. In these cases, we could not estimate the number of parameters of the target model and consequently not report the efficiency score of an attack. In cases when we estimated the number of queries, we also marked the efficiency score as estimated. In one paper [PYZ18], the authors only mention that they use a ResNet model, referring to the paper where ResNet models were introduced. As this original paper mainly focuses on ResNet-34 architecture, we assumed that ResNet-34 was used as a target architecture by [PYZ18] and calculated efficiency scores based on this assumption.

We group attacks with similar query budgets (provided in the first column) and the same data type (the second column) used for an attack. We provide further analysis by comparing attacks within the same group. We note that the attacks from one group are *not* necessarily all equal, as the knowledge about the target architecture and output may differ. As we do not implement an attack that uses NPD data, we compare our data-free attack with both data-free and NPD attacks. Below, we summarise our main findings from Table 7.8.

- As we already reported in Section 3.3.6, a lot of information on other attacks is missing. For instance, we can not categorise some papers based on the attacker's strength, as authors did not report which knowledge the attacker possesses. Another issue is that the accuracy of the target model was not always reported. Providing fidelity scores in that case becomes crucial, but not all papers did that. Finally, only one paper reported all three metrics for the substitute model (accuracy, fidelity, transferability).

- In terms of accuracy and fidelity, our work outperforms all attacks from Table 7.8 that use original or problem-domain data, except for [MSDH19]. However, that work assumed that the target model outputs gradients, which is a much stronger assumption than label outputs considered in this work.

- Regarding transferability, all works that reported transferability scores perform better than our attacks. However, getting high transferability scores was not the primary goal of this thesis, and we report these values mainly for comparability. Moreover, as there are different ways how transferability can be measured, it is unclear if the scores from Table 7.8 are comparable. We will discuss this question further in Chapter 8.

- For the query budget [10k, 15k], our data-free attack outperforms all NPD attacks that use only labels for training. When other works use probability scores (a stronger assumption than in our attack), they reach fidelity that are approximately 2% higher than our attack.

Table 7.8: Comparison of substitute model attacks implemented in this work with the state-of-the-art.

Query budget	Data	Paper	Same archit.	Outputs	Target Acc.	Sub. Acc.	Sub. Fid.	Sub. Tr.	Queries	Efficiency score
<2.5k	Original	[PYZ18]	✗	N/A	>91%		77.47%	65.84%	1.6k	0.00007 est.
		[MSDH19]	✓✗	Gradients	90% est.	88% est.			1k	0.00009
		this work	✗	Labels	97.14%	84.63%	84.85%	0.05%	1k	0.00005
	PD	this work	✗	Labels	97.14%	74.75%	74.79%	0.03%	1k	0.00005
	Data-free	this work	✗	Labels	97.14%	68.02%	68.18%	0.01%	1k	0.00005
[5k, 8k]	Original	[PYZ18]	✗	N/A	>91%		77.96%	72.53%	6.4k	0.0003 est.
		[YHL+22] (DTMEA)	✗	Probabilities +XAI	92.03%	77% est.			8k	0.00034
		this work	✗	Labels	97.14%	90.68%	90.94%	0.05%	5k	0.00023
	PD	this work	✗	Labels	97.14%	84.01%	84.18%	0.06%	5k	0.00023
	Data-free	this work	✗	Labels	97.14%	71.97%	72.14%	0.01%	5k	0.00023
[10k, 15k]	Original	[PYZ18]	✗	N/A	>91%		83.61%	73.15%	12.8k	0.0006 est.
		this work	✗	Labels	97.14%	93.90%	93.88%	0.07%	10k	0.00046
	PD	this work	✗	Labels	97.14%	87.13%	87.20%	0.01%	10k	0.00046
	NPD	[PGS+20] (Activethief)	✓✗	Labels	N/A		64.23%		10k	N/A
				Probabilities			77.29%			N/A
		[PGS+19]	✓✗	Labels	N/A		71.14%		15k	N/A
				Probabilities			77.29%		10k	N/A
		[GCY+21] (InverseNet)	✗	Labels	N/A		75.40%		10k	N/A
				Probabilities			77.80%			N/A
	Data-free	this work	✗	Labels	97.14%	75.39%	75.61%	0.01%	10k	0.00046
[20k, 30k]	Original	[PYZ18]	✗	N/A	>91%		84.12%	74.21%	25.6k	0.0012 est.
		this work	✗	Labels	97.14%	94.68%	94.73%	0.15%	20k	0.0009
	PD	this work	✗	Labels	97.14%	90.20%	90.54%	0.08%	20k	0.0009
	NPD	[PGS+20] (Activethief)	✓✗	Labels	N/A		78.36%		30k	N/A
		[PGS+19]	✓✗	Labels	N/A		78.36%		30k	N/A
		[WL22]	N/A	Labels	N/A	80.90%			30k	0.0014
		[WLL+22] (Black-box Dissector)	✓✗	Labels	91.56%	80.47%	82.14%	76.63%	30k	0.0014
	Data-free	this work	✗	Labels	97.14%	74.41%	74.74%	0.03%	20k	0.0009
≥45k	Original	this work	✗	Labels	97.14%	95.48%	95.50%	0.33%	45k	0.0021
	PD	[CSBB+18] (CopyCat)	✓	Probabilities	95.30%	90.00%			269k	0.002
		this work	✗	Labels	97.14%	91.87%	91.94%	0.08%	45k	0.0021
	NPD	[CSBB+18] (CopyCat)	✓	Probabilities	95.30%	94.00%			3.4m	0.0253
		[PGS+20] (Activethief)	✓✗	Labels	N/A		84.99%	85.76%	120k	N/A
							81.57%		100k	N/A
		[ASJ+20]	✓	Labels	94.60%	53.60%			100k	0.0047
				Probabilities		88.20%				0.0047
		[BCIP20]	✗	Probabilities	82.50%	79.00%			>50k est.	> 0.0009 est.
		[MDN19]	✗	Labels	90.48%	89.59%			>1m est.	> 0.0074 est.
	Data-free	[KPQ21a]	✗	Probabilities	92.26%	89.85%			30m	111
		[TMWP21] (DFME)	✗	Probabilities	95.50%	88.10%			20m	0.94
						89.90%			30m	1.41
		[MHS21] (MEGEX)	✗	Probabilities +XAI	95.50%	72.10%			1m	0.047
						90.40%			10m	0.47
						92.30%			20m	0.94
		[SAB22]	✗	Labels	95.50%	84.51%			8m	0.38
				Probabilities		91.24%				0.38
		[YDZ+22]	✓✗	Labels	91.93%	69.64%			>1m est.	> 0.047 est.
				Probabilities		80.79%		100%		> 0.047 est.
		this work	✗	Labels	97.14%	75.50%	75.71%	0.06%	45k	0.0021

- For the query budget [20k, 30k], the NPD attacks outperform our data-free attack by 4-8%.
- In the category of attacks that use 45,000 or more queries, our work outperforms one NPD attack and two data-free attacks. However, all works in this query budget group consider a stronger attacker (that uses probabilities, for instance) or use significantly more queries. In general, most of the works in this category use millions of queries, which is very inefficient and likely an unrealistic assumption.
- Two works have exactly the same weakest assumption about the attacker's knowledge as this thesis. The first work [YDZ$^+$22] reaches 69.64% of accuracy using more than a million queries, based on our estimation explained earlier. Our data-free attack has 75.5% of accuracy, being trained only on 45,000 samples. Another work [SAB22] has a higher accuracy score of 84.51%. However, it uses 8 million queries, which is 177 times more than our attack.

Overall, our data-free attack shows promising results for future work. As we did not invest time into improving the dataset's quality, we expect to reach higher scores with further research.

7.2 Defence Analysis

In this thesis, we test three defence approaches: input perturbation, output perturbation, and input-output perturbation. At first, we explore to which extent they impact the performance of the baseline attacks reported in Table 7.1. As stated earlier, those baselines correspond to the strongest attacker's knowledge in this thesis: they use the original data and the target model architecture to train a substitute. We report the performance of attacks against protected models in Table 7.9. In addition, we provide the difference in the baseline scores for each metric. Hence, a more significant attack mitigation means a larger negative difference. We highlight in bold the lowest scores, i.e. indicating the strongest defence, obtained for each target model and query budget. Below, we summarise the main observations based on changes in fidelity and accuracy scores.

- In most cases, the lowest attack performance against defended models is lower than the baseline. However, the difference is rather minute, especially considering that defences modify nearly 1% of output labels. Hence, we have to investigate further how the utility of the target model changes when defences are applied. If it turns out that the decrease in utility is comparable with the decrease in attack performance, applying these perturbation defences has no benefits compared to randomly outputting wrong labels for 1% of data.
- Sometimes, only one of three defences mitigates an attack performance. Moreover, there is no overall trend of one defence being better than others. Therefore, applying

Table 7.9: Performance of defences against substitute model training attacks measured on the CIFAR10 test set. Below each score we report the difference with the baseline results reported in Table 7.1.

	Target →	SimpleNet				ResNet-34 (from scratch)				ResNet-34 (transfer learning)			
Defence ↓	Metric→ QB ↓	Joint Acc	Acc	Fid	Tr	Joint Acc	Acc	Fid	Tr	Joint Acc	Acc	Fid	Tr
Input perturbation	1k	47.37% -0.7%	49.58% -0.68%	49.75% -0.64%	**1.3% -0.22%**	77.31% +0.62%	80.11% +0.76%	79.28% +0.54%	**0% +0.00%**	80.83% +0.25%	81.89% +0.28%	81.98% +0.19%	**0.07% +0.03%**
	5k	**70.17% +0.33%**	**72.34% +0.33%**	**73.5% +0.49%**	**2.48% +0.03%**	86.04% +0.72%	89.27% +0.73%	88.06% +0.82%	0.01% +0.01%	88.71% -0.76%	89.59% -0.7%	90.09% -0.82%	0.1% +0.00%
	10k	76.29% -0.1%	78.89% +0.05%	**79.69% -0.24%**	**3.08% -0.19%**	88.73% +0.22%	92.35% +0.44%	90.62% +0.11%	**0% +0.00%**	91.46% +0.16%	92.34% +0.08%	92.96% +0.23%	**0.14% -0.03%**
	20k	82.68% -0.02%	85.47% +0.09%	86.37% +0.04%	5.53% -0.33%	90.77% +0.04%	94.68% -0.05%	92.55% +0.16%	**0.01% +0.00%**	**93.36% -0.41%**	**94.38% -0.41%**	**94.82% -0.34%**	0.42% +0.14%
	45k	**86.76% -0.26%**	**89.7% -0.57%**	90.63% +0.02%	9.01% +0.58%	92.2% +0.05%	96.57% -0.1%	93.64% +0.14%	**0.02% -0.03%**	**95.17% -0.2%**	**96.35% -0.14%**	96.57% -0.27%	1.43% +0.21%
Output perturbation	1k	**46.58% -1.49%**	**48.72% -1.54%**	**48.99% -1.4%**	1.53% +0.01%	**76.68% -0.01%**	**79.42% +0.07%**	**78.66% -0.08%**	0.01% +0.01%	81.04% +0.46%	82.08% +0.47%	82.19% +0.4%	0.08% +0.04%
	5k	70.76% +0.92%	72.99% +0.98%	74.01% +1%	2.52% +0.07%	**85.65% +0.33%**	**89.02% +0.48%**	**87.74% +0.5%**	**0% +0.00%**	88.93% -0.54%	89.7% -0.59%	90.38% -0.53%	**0.09% -0.01%**
	10k	76.3% -0.09%	78.75% -0.09%	79.73% -0.2%	3.23% -0.04%	**88.29% -0.22%**	**91.8% -0.11%**	**90.24% -0.27%**	0.01% +0.01%	**91.08% -0.22%**	**92% -0.26%**	**92.57% -0.16%**	0.22% +0.05%
	20k	82.29% -0.41%	85.05% -0.33%	**85.9% -0.43%**	5.67% -0.19%	**90.64% -0.09%**	**94.54% -0.19%**	**92.37% -0.02%**	**0.01% +0.00%**	93.71% -0.06%	94.74% -0.05%	95.08% -0.08%	0.38% +0.1%
	45k	86.96% -0.06%	90.25% -0.02%	90.58% -0.03%	**8.5% +0.07%**	92.17% +0.02%	96.66% -0.01%	93.52% +0.02%	0.04% -0.01%	95.22% -0.15%	96.47% -0.02%	**96.55% -0.29%**	**1.31% +0.09%**
Input-output perturbation	1k	47.93% -0.14%	50% -0.26%	50.41% +0.02%	1.31% -0.21%	77.42% +0.73%	80.2% +0.85%	79.4% +0.66%	**0% +0.00%**	**80.74% +0.16%**	**81.8% +0.19%**	**81.93% +0.14%**	**0.07% +0.03%**
	5k	70.49% +0.65%	72.76% +0.75%	73.7% +0.69%	2.51% +0.06%	86.35% +1.03%	89.58% +1.04%	88.4% +1.16%	**0% +0.00%**	**88.58% -0.89%**	**89.58% -0.71%**	**89.9% -1.01%**	0.1% +0.00%
	10k	**76.24% -0.15%**	**78.68% -0.16%**	79.71% -0.22%	3.14% -0.13%	88.66% +0.15%	92.11% +0.2%	90.53% +0.02%	0.01% +0.01%	91.5% +0.2%	92.38% +0.12%	92.96% +0.23%	0.15% -0.02%
	20k	**82.25% -0.45%**	**84.91% -0.47%**	85.91% -0.42%	**5.45% -0.41%**	90.8% +0.07%	94.64% -0.09%	92.58% +0.19%	**0.01% +0.00%**	93.71% -0.06%	94.7% -0.09%	95.21% +0.05%	**0.31% +0.03%**
	45k	86.89% -0.13%	90.03% -0.24%	**90.55% -0.06%**	8.69% +0.26%	**92.09% -0.06%**	**96.51% -0.16%**	**93.51% +0.01%**	0.04% -0.01%	95.25% -0.12%	96.46% -0.03%	96.65% -0.19%	1.37% +0.15%

only one specific defence does not seem secure from the model owner's perspective, as it does not promise that the attack performance will decrease.

- For some configurations, all defences actually **increased** the performance of the attack, which completely contradicts their goals and raises the question whether they are fit for purpose.

During the hyperparameters tuning, we set the limit on the number of perturbed labels to 1%. It has to be noted that those flipped labels can negatively or **positively** impact the target model utility. If the label is flipped for a sample that was originally miss-classified by the target model, its perturbed label could turn out to be correct. In that case, the accuracy of the target model can increase, leading to no harm for benign model clients. To measure the utility, we check the accuracy of target models of the test set after applying a defence and compare these scores with the unprotected models. The results are reported in Table 7.10. The difference scores can now be used as a threshold for defence estimation. If a defence decreases the performance of a substitute model to a lesser extent than the performance of the target model, we can call this defence useless. Although in some cases reported in Table 7.9 the defences decrease an attack's performance more than they impact the target's utility, their contributions are negligible, and the performance is too unstable. Hence, these defences do not really protect a model against the strongest attacker.

Table 7.10: Accuracy of target models on their original task, after applying data perturbation defences compared to the unprotected models.

	SimpleNet	ResNet-34 (from scratch)	ResNet-34 (transfer learning)
Unprotected	**91.76%**	93.61%	97.14%
Input perturbation	91.56% -0.2%	**93.67%** **+0.06%**	97.18% +0.04%
Output perturbation	91.67% -0.09%	93.62% +0.01%	97.21% +0.07%
Input-output perturbation	91.42% -0.34%	93.59% -0.02%	**97.29%** **+0.15%**

Table 7.11: Performance of defences against data-free substitute model training attacks measured on the CIFAR10 test set. Below each score we report the difference with the results for unprotected models reported in Table 7.5.

	Target →	SimpleNet				ResNet-34 (from scratch)				ResNet-34 (transfer learning)			
Defence ↓	Metric→ QB ↓	Joint Acc	Acc	Fid	Tr	Joint Acc	Acc	Fid	Tr	Joint Acc	Acc	Fid	Tr
Input perturbation	1k	**63.42%** **+0.14%**	**66.4%** **+0.2%**	**66%** **+0.14%**	**0%** **+0.00%**	65.54% +1.41%	67.46% +1.4%	67.73% +1.41%	**0%** **+0.00%**	66.81% +0.76%	67.61% +0.78%	67.87% +0.77%	0.03% +0.00%
	5k	66.55% -1.35%	69.81% -1.3%	69.1% -1.45%	0.01% +0.01%	**68.15%** **+0.85%**	**70.29%** **+0.85%**	**70.22%** **+0.7%**	**0%** **+0.00%**	69.52% -1.6%	70.49% -1.48%	70.56% -1.58%	**0.02%** **+0.01%**
	10k	70.59% +0.66%	73.75% +0.71%	73.37% +0.51%	0.01% +0.01%	72.15% +0.77%	74.38% +0.66%	74.4% +1%	**0%** **+0.00%**	67.95% -1.63%	68.91% -1.55%	68.97% -1.66%	0.04% +0.02%
	20k	72.72% -0.71%	75.86% -0.86%	75.66% -0.76%	0.02% +0.01%	76.67% +2.71%	79.04% +2.73%	79% +2.81%	**0%** **-0.01%**	**68.27%** **+0.19%**	**69.15%** **+0.14%**	**69.27%** **+0.11%**	0.03% +0.02%
	45k	76.99% +1.3%	80.13% +1.25%	80% +1.22%	**0%** **-0.02%**	78.25% +3.31%	80.49% +3.39%	80.67% +3.39%	**0%** **+0.00%**	73.67% +0.71%	74.51% +0.7%	74.76% +0.76%	**0.03%** **+0.00%**
Output perturbation	1k	63.46% +0.18%	66.45% +0.25%	66.05% +0.19%	**0%** **+0.00%**	65.48% +1.35%	67.4% +1.34%	67.71% +1.39%	**0%** **+0.00%**	**65.63%** **-0.42%**	**66.44%** **-0.39%**	**66.68%** **-0.42%**	**0.01%** **-0.02%**
	5k	**66.3%** **-1.6%**	**69.48%** **-1.63%**	**68.78%** **-1.77%**	**0%** **+0.00%**	69.67% +2.37%	72 +2.56%	71.85% +2.33%	0.01% +0.01%	**68.28%** **-2.84%**	**69.09%** **-2.88%**	**69.31%** **-2.83%**	**0.02%** **+0.01%**
	10k	**70.06%** **+0.13%**	**73.26%** **+0.22%**	**72.9%** **+0.04%**	0.01% +0.01%	**69.48%** **-1.9%**	**71.69%** **-2.03%**	**71.61%** **-1.79%**	**0%** **+0.00%**	**67.83%** **-1.75%**	**68.81%** **-1.65%**	**68.74%** **-1.89%**	**0.01%** **-0.01%**
	20k	**72.34%** **-1.09%**	**75.67%** **-1.05%**	**75.29%** **-1.13%**	**0.01%** **+0.00%**	**73.77%** **-0.19%**	**76.09%** **-0.22%**	**75.98%** **-0.21%**	**0%** **-0.01%**	68.57% +0.49%	69.36% +0.35%	69.61% +0.45%	**0.01%** **+0.00%**
	45k	78.07% +2.38%	81.23% +2.35%	81.06% +2.28%	0.01% -0.01%	77.03% +2.09%	79.3% +2.2%	79.27% +1.99%	0.01% +0.01%	**71.91%** **-1.05%**	**72.82%** **-0.99%**	**73.03%** **-0.97%**	**0.03%** **+0.00%**
Input-output perturbation	1k	63.43% +0.15%	66.48% +0.28%	**66%** **+0.14%**	**0%** **+0.00%**	**65.33%** **+1.2%**	**67.33%** **+1.27%**	**67.5%** **+1.18%**	0.01% +0.01%	66.3% +0.25%	67.11% +0.28%	67.37% +0.27%	0.03% +0.00%
	5k	67.51% -0.39%	70.76% -0.35%	70.22% -0.33%	**0%** **+0.00%**	68.34% +1.04%	70.55% +1.11%	70.53% +1.01%	**0%** **+0.00%**	68.67% -2.45%	69.46% -2.51%	69.77% -2.37%	0.04% +0.03%
	10k	70.53% +0.6%	73.71% +0.67%	73.38% +0.52%	**0%** **+0.00%**	72.66% +1.28%	74.98% +1.26%	74.87% +1.47%	**0%** **+0.00%**	68.33% -1.25%	69.28% -1.18%	69.32% -1.31%	0.02% +0.00%
	20k	72.96% -0.47%	76.21% -0.51%	75.89% -0.53%	**0.01%** **+0.00%**	75.46% +1.5%	77.82% +1.51%	77.75% +1.56%	**0%** **-0.01%**	70% +1.92%	70.8% +1.79%	71.06% +1.9%	0.05% +0.04%
	45k	**74.58%** **-1.11%**	**77.76%** **-1.12%**	**77.57%** **-1.21%**	0.02% +0.00%	**76.26%** **+1.32%**	**78.48%** **+1.38%**	**78.62%** **+1.34%**	0.01% +0.01%	75.33% +2.37%	76.23% +2.42%	76.48% +2.48%	0.07% +0.04%

Table 7.10 also shows that for the pretrained ResNet-34 model, all defences improved its performance. Hence, they could be used as a post-processing step to improve the model's prediction quality. However, as the scores from Table 7.9 suggest, the attacker can also learn more about the model in this case.

We conclude our defence analysis by investigating if data-perturbation defences are effective against attacks with limited attacker knowledge. In particular, we test the defences against our weakest data-free attack that uses ResNet-18 as a substitute architecture. We already reported the attack performance against unprotected target models in Table 7.5. We show the impact of defences in Table 7.11. As before, we highlight the lowest scores in bold. Our key observations are:

- The impact of the defences applied is more significant compared to the strongest adversary that uses the original data and the target model's architecture to train a substitute model (Table 7.9). In particular, for the pre-trained ResNet-34, the fidelity for 5,000 queries is decreased by almost 3%. At the same time, as we have seen before, the utility of this target model only increases when defences are applied.
- However, unwanted improvements of attacks are also more pronounced, sometimes improving the attack performance by 3%.
- Even if the performance of an attack is decreased, it is still a relatively minor change that does not eliminate the success of the attack. It is an open, subjective and case-dependant question of how significantly the performance should drop to make an attack useless, but in many cases, 3% is likely insufficient.
- The output perturbation defence results in the lowest scores in most of the configurations. We did not observe that trend in Table 7.9.

Hence, although we observe more notable changes than for the baseline attacks, the results are still too unstable and minute to recommend using these defences as mitigation techniques. Moreover, the number of changes most likely increased, as more than 1% of labels were flipped[2]. As the artificial dataset used in these experiments differs from CIFAR10, the optimal defence hyperparameters are likely different. However, as the model owner does not know which data is used for an attack, these hyperparameters can not be estimated better.

[2]As the hyperparameters were tuned to flip 1% for the CIFAR10 validation set, a slightly different impact on a different dataset is expected

CHAPTER 8

Discussion

As conclusion of this thesis, we highlight the main insights and summarise our results. In particular, we discuss non-comparability issues and inconsistency of the related work and propose some suggestions to address these problems. Additionally, we address the issues of this work and explain why certain approaches could not work. Afterwards, we summarise this thesis's contributions and main results and answer the research questions formulated at the beginning. We conclude our discussion by describing directions of future work that cover both attack and defence development.

8.1 Insights and Issues

While reviewing the related work, we faced a significant lack of harmonisation and standardisation of how model stealing attacks are designed and evaluated. In the previous chapters, we already highlighted some of the issues. Below, we list them all together and propose solutions for future works, so that these problems could be avoided.

- **Non-comparable attack settings.** As different attacker capabilities are assumed in different papers, they need be taken into account when comparing attacks. Moreover, information about the attacker's knowledge is sometimes missing, leading to further comparability issues.

  Proposed solution. We suggest always defining a threat model and using our categorisation from Figure 3.3 for substitute training attacks. It shows which group an attack belongs to, and which other papers are most suitable for comparison.

- **Non-comparable performance scores.** As stated in Section 3.3.6, there is no single metric that was reported in all papers performing substitute training attacks against image classifiers. If the accuracy of the target model is given, one can evaluate the performance of a substitute compared to the target model and use this estimation for paper comparison. However, not all works provide the target accuracy, making such comparison impossible.

Proposed solution. We propose reporting the accuracy score for target models and the following metrics for substitute models: accuracy on the original test set, fidelity, and (untargeted) transferability.

- **Inconsistent way of measuring transferability.** We suggested above to report transferability for every substitute training attack. However, transferability has no unified way of measuring, unlike accuracy and fidelity. One needs to decide which adversarial example crafting technique to use, and to set its hyperparameters. Depending on the selected technique, adversarial examples may lay closer or further from the decision boundary. In this work, for instance, we used DeepFool, known as an approach to minimise the distance to the decision boundary. Hence, adversarial examples crafted in this work were very sensitive and specific to the target model's decision boundary, so them being also adversarial for the target model is less likely. If one selects a less sensitive method, it may turn out to be too strong, making a lot of other models fooled by adversarial examples.

  Proposed solution. Firstly, we strongly recommend reporting how transferability scores are measured, including algorithm-specific hyperparameter values. Secondly, in the perfect scenario, the adversarial crafting technique should be unified. We suggest using DeepFool as one of the most sensitive and challenging approaches, which is thus powerful in measuring how well the decision boundary is replicated. We also encourage further research to investigate how transferability scores can be put in relation to the strength of the adversarial crafting approach. We leave it for future work to explore how the strength should be defined, although we can suggest using, e.g. the average distance of adversarial examples to the decision boundary.

- **Non-comparable original datasets.** While comparing the results of this thesis with the state-of-the-art, we met another comparability issue. Original datasets used for target model training differ in complexity and lead to a difference in stealing complexity. For instance, stealing a neural network trained on MNIST is more straightforward than stealing a network trained on CIFAR10, since the first classification task is generally simpler. Hence, we only compare results from this work with attacks against CIFAR10 classifiers.

  Proposed solution. Limiting all research in model stealing to several datasets limits the research in general, so a recommendation to use only several benchmark datasets is too restricting. However, launching at least a few key experiments on some commonly used datasets from the corresponding data domain would be a good calibration indicator. As we showed in Section 6.1, for image classification, the most popular choice of the dataset is CIFAR10. Hence, we recommend using it to increase comparability in this domain.

- **Low-performing target models.** In general, model stealing does not require a well-performing target model, and also applies to models with random weights.

However, a target model may perform poorly compared to the state-of-the-art. Then, a model stealing attack can be evaluated by comparing the accuracy of a substitute and the target models' accuracy. However, since reaching lower performance is generally an easier task, such attack evaluation can be misleading.

Proposed solution. We recommend (1) always reporting fidelity and (2) training a target model until it reaches a reproducible benchmark performance.

- **Manipulations with the original test set.** One must ensure that attacks are evaluated on the same test set for a fair evaluation. However, we have seen two categories of manipulations that lead to modifications in the test set. The first category of papers used part of the test data for training a substitute model. Although we can understand the motivation to do so (getting original data, but not from the target model's train set), it impacts the final test scores of the attack as they are measured on the remaining part of the test set. For instance, the distribution of classes can be different, or some of the samples more difficult to predict can be eliminated from the test set. The second manipulation category we encountered was removing a complete class from the CIFAR10 dataset to get the same set of classes as in one of the PD datasets. This modification leads to an evaluation of a different classification task than the original CIFAR10, and scores are hardly comparable.

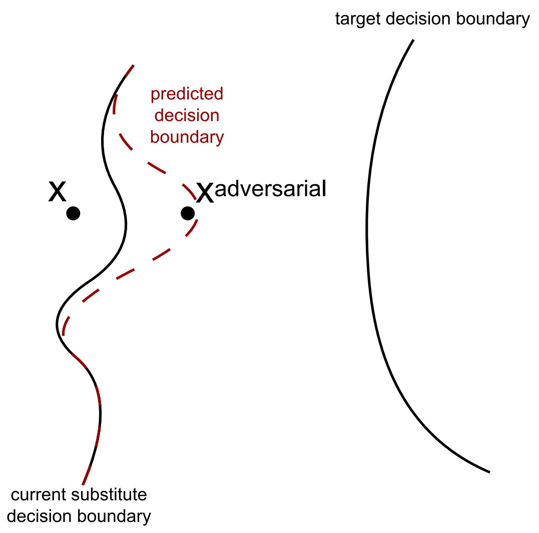
  Proposed solution. We do not recommend modifying the original test set in any way. In case an attack assumes such modifications, it should be considered while comparing with other works.

- **Wrongly defined data category.** CIFAR100 is used as a non-problem domain dataset for CIFAR10 in some works. However, although these datasets depict different objects, they have similar visual properties. For instance, they are more alike than CIFAR10 and ImageNet.

  Proposed solution. We would suggest using ImageNet instead. Generally, it is "safer" to use a dataset gathered separately from the original dataset.

We summarised above the problems we faced while preparing the theoretical part of this thesis. However, some of our designed and implemented experiments did not meet their goals and our expectations. We analyse their behaviour and list below our speculations on those topics.

- **Low transferability scores**. All attacks implemented in this thesis performed poorly in terms of transferability. Although high transferability was not our goal, we still would like to investigate the reasons for such behaviour. In two of our query optimisation algorithms, adversarial examples were used. As they lay close to the decision boundary, we expected that these optimisations would lead to higher transferability. However, this was not the case. We assume that the reason lies in the adversarial crafting algorithm we chose. Deepfool is one of the least-perturbing algorithms for crafting adversarial examples. Most likely, perturbations that fooled our substitute model were not significant enough to also fool a target model that has ever an even just slightly different decision boundary. We illustrate how the decision boundary of the substitute model is changing in this case in Figure 8.1b. As an adversarial example gets assigned the same label from the target model as the unperturbed sample, the substitute model must adapt its decision boundary. However, if such an update is too small, the substitute model's decision boundary will not reach the target even after many steps. However, even if an adversarial example fools the target model (which we assume was barely happening) and is mislabelled compared to the unperturbed sample, the predicted decision boundary will be between those two samples. It will not reach the decision boundary of the target model even after several training steps as shown in Figure 8.1.


target decision boundary
predicted decision boundary
X
X adversarial
current substitute decision boundary
(a) $x^{\text{adversarial}}$ does not fool the target model, which lead to a local decision boundary update.

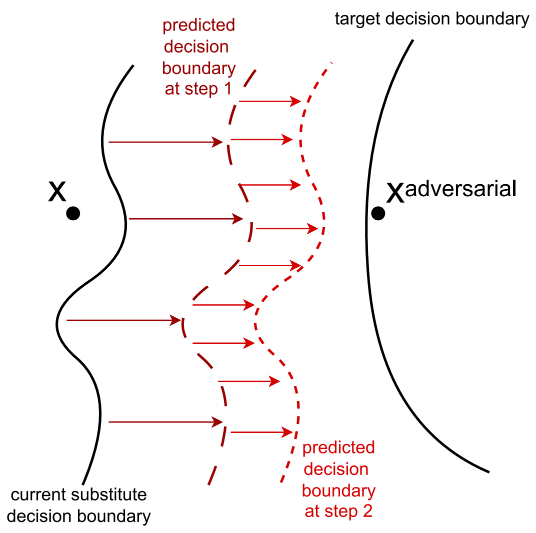

predicted decision boundary at step 1
target decision boundary
X
X adversarial
predicted decision boundary at step 2
current substitute decision boundary
(b) Even though $x^{\text{adversarial}}$ fools the target model, the predicted decision boundary does not reach the target.

Figure 8.1: Predicted substitute decision boundary when using adversarial examples with small perturbations.

- **Unstable defences behaviour.** As we concluded in the previous section, the defences considered in this thesis can not be recommended as a protection mechanism against model stealing. First of all, their impacts are rather unpredictable. While a defence decreases an attack's performance in one configuration, it improves its performance in another configuration. Moreover, the difference in performance is not significant enough to (i) be worth the risk of potentially helping the attacker, and (ii) paying the cost of an ineffective defence by having a less performing target model. We further investigated the following questions: (1) why the performance of defences is unstable and changes depending on the query budget, and (2) why the difference in performance of protected and unprotected models is more significant when artificial data is used. For (1), we assume that the distribution of perturbed samples for each query budget could differ, which lead to different impact on the attack's performance. Regarding (2), defence hyperparameters were tuned on CIFAR10, hence the number of flipped labels was under control only for this dataset. The defences likely perturbed significantly more labels in the artificial dataset, causing a larger difference in attack performance.

8.2 Contributions and Main Results

In this thesis, new attack and defence settings were investigated through adapting methods proposed earlier to new scenarios and developing new approaches. Our main contributions are:

- We highlighted problems of non-comparability in the related work and proposed a solution for each of them.
- In particular, we introduced a categorisation for substitute training attacks focused on stealing image classifiers. Our categorisation makes the comparison of different attack techniques fairer, by considering the attacker's strength.
- We explored how the target model's complexity and transfer learning usage impact the attack performance.
- We investigated how different hyperparameter configurations impact the performance of attacks that use query optimisation techniques such as active learning and adversarial augmentation.
- We proposed an algorithm combining active learning and adversarial augmentation into one query optimisation technique.
- We developed a novel data-free attack based on a text-to-image stable diffusion model.
- Finally, we adapted a defence against adversarial examples to model stealing attacks and tested it together with another data-perturbation defence.

In Chapter 7, we described all experiments we performed, as well as their outcomes, in detail. As a summary, we provide our key observations from the obtained results.

- **Transfer learning.** Attacks that use the same training strategy as the target model (training from scratch or using transfer learning) for a substitute model perform better than the ones that use the opposite strategy.

- **Data-free attack.** Using artificial data generated by a stable diffusion model, we achieved nearly 75% fidelity with 45,000 queries. Although there are data-free attacks that are more effective, they are significantly less efficient. In particular, we outperformed one of two attacks that assume the same weakest possible attacker's knowledge. Although another attack is better than ours by almost 10%, it uses 8,000,000 samples, 177 times more than ours.

- **Query optimisation.** The Active Learning approach showed the highest overall improvement of attack performance in various scenarios. It improved the performance of all attacks that used 10,000 or 20,000 samples compared to the non-optimised attacks.

- **Data-perturbation defences.** None of the defences implemented in this work protect target models well enough – and sometimes even improve the performance of an attack. However, they conversely can also improve the performance of some target models.

- **Original and PD state-of-the-art.** Among attacks on CIFAR10 classifiers that use original or problem-domain data, our attacks reach the highest accuracy and fidelity scores having the same or smaller budget than related work.

8.3 Research Questions

To conclude the analysis of the obtained results, we revisit the research questions formulated at the beginning of the thesis. We answer them using our observations from Chapter 7.

1. **To what degree are model stealing attacks effective?** This generic question wraps up all the experiments conducted during attack analysis. As we have seen before, the effectiveness of an attack is affected by many factors, including the attacker's knowledge, capabilities, and the query budget. We analyse how each factor influences substitute training attacks by answering the following sub-questions.

    a) **To what extent does the effectiveness of attacks depend on its query budget?** In the thesis, five different query budgets were analysed, namely 1,000 queries, 5,000 queries, 10,000 queries, 20,000 queries, and 45,000 queries, which corresponds to the size of the original training set. As we started our analysis from the strongest attacker, which uses original data and target architecture to train a substitute, the answer looked very straightforward: the more queries, the better the attack's performance. However, this was not the case for the artificial data. For one of the models, an attack with 5,000 queries outperformed attacks using 10,000 and 20,000 queries. We suspect that the artificial dataset is too homogeneous. Later in the experiments, we saw that models that only used 1,000 artificial queries reached over 90%

accuracy on the validation set, suggesting that the data samples have similar characteristics that can be learned from a small subset. Hence, we conclude that the performance improves if the dataset is too complex to learn its features from a smaller subset. We assume that this is the case for natural (non-artificial) data.

Another observation is that the improvement in the attack performance slows down with the increase in queries. We obtained the most significant improvement when the number of queries increased from 1,000 to 5,000. At the same time, improvement between 20,000 and 45,000 are the smallest. For instance, ResNet-18 trained on 20,000 original samples reaches fidelity scores of 91.15%-94.13% for different target models. With 45,000 original queries, this range changes to 91.71%-95.44%. Therefore, from a certain point, a minute improvement may not be worth the effort needed to gather more data.

b) **To what extent do the effectiveness and efficiency of attacks change depending on the complexity of the target model?** We explored two target architectures: SimpleNet, and the more complex ResNet-34. We trained two ResNet models, one from scratch and one using transfer learning, with the model being pre-trained on ImageNet. The SimpleNet model had the lowest performance on the original dataset, followed by the ResNet-34 trained from scratch. The ResNet-34 trained with transfer learning reached the highest accuracy score. We observed from the experiment results that the performance of the target models correlates with the fidelity score reached by an attack. It turned out that it is easier to steal a better-performing model. Hence, the attack performance was the best for ResNet-34 trained using transfer learning and the worst for SimpleNet. The question, however, asks about the complexity of the target model and not about its performance. As the complexity represents the learning capability of the model, it directly influences the performance. Hence, for the models we explored, the naive conclusion is that it is easier to steal models of higher complexity. However, for a complete answer, one needs to investigate if there is a difference in stealing models of different complexity but the same performance.

c) **To what extent does the effectiveness of model stealing attacks change when the target model architecture is not known?**

Three substitute models were trained for each target model: SimpleNet, ResNet-18, and ResNet-34. The highest scores were obtained with ResNet-34 and the lowest with SimpleNet, which means that a more complex substitute model architecture leads to better results. However, for smaller query budgets, ResNet-18 outperformed ResNet-34. Thus, we would rather conclude that the architecture of the substitute model should be complex enough to learn the original dataset. At least in our case, this means that the knowledge about the target model architecture has only a minor effect on the attack's performance.

d) **How does the effectiveness of model stealing attacks change if the**

target model is trained with transfer learning? As mentioned earlier, we attacked two ResNet-34 target models, one trained from scratch and one trained using transfer learning. With this research question, we wanted to explore if there is a difference in stealing those models, considering that an adversary may have access to the same pre-trained weights. For ResNet-18 and ResNet-34 substitute models, which were trained using the same transfer learning approach as the target model, we observed that the attack performance scores are higher when stealing the pre-trained ResNet-34. However, this target model also performs better on the original test than the model trained from scratch. Without further research, it is difficult to state the cause for the difference in the stealing performance. The reason is either the usage of transfer learning or the difference in the performance of the target models.

We observe a different behaviour pattern when using SimpleNet as substitute model. This model scored higher when stealing target models trained from scratch (SimpleNet, ResNet-34) than the pre-trained ResNet-34. This observation suggests that substitute models reach higher scores when their training strategy coincides with the training strategy of the target model.

e) **To what extent does the effectiveness of model stealing attacks change depending on the availability of data?** We explored three configurations for the attacker's dataset: original data, problem-domain data, and artificially generated data. The best scores were obtained with the original data. With the problem-domain data, the performance was lower, but by increasing the query budget, it got closer to the original data. For the artificial data, the drop in the attack's performance was the most significant. Interestingly, for the artificial data, the best stealing performance was reached with larger query budgets when attacking the SimpleNet target model, whereas the worst was obtained for the pre-trained ResNet-34. Most likely, those target models had different outputs for a significant amount of the artificial samples, leading to a difference in the substitute's performance.

We additionally explored the difference in the performance of a substitute model measured on the attacker's validation and original test sets. When the attacker used the original data, this difference was minimal, with slightly better scores on the validation set. For the problem-domain dataset, which was more complex than the original data, validation scores were significantly lower than the test scores. For the artificial dataset, the situation was the opposite. The validation scores were much higher than the test ones, suggesting that the artificial data is simpler than the original. Based on those observations, we conclude that an attack is more effective when the complexity of the attacker's dataset is higher than the original, compared with the dataset with lower complexity than the original data.

f) **To what extent does query optimisation of model stealing attacks improve their effectiveness?** In this thesis, three query optimisation strategies were studied: active learning, adversarial augmentation, and their combination, referred to as active adversarial augmentation. For 1,000 and 5,000 query budgets, for some target models, these optimisation techniques made the performance of the corresponding substitute model worse. However, the performance was always improved for 10,000 queries and more, sometimes by up to 5%. In most cases, active learning performed better than other optimisation techniques. However, it was tested for at most 20,000 queries, as applying it when all the attacker's data is used does not change anything. For 45,000 queries, active adversarial augmentation always improved the attack's performance. Hence, we suggest using active learning from 10,000 queries, and its application does not make sense for the given query budget, switching to active adversarial augmentation.

2. **To what degree are data perturbation defences against model stealing attacks useful?**

   Our exploration of the data-perturbation defences did not yield almost any positive results. We concluded that in the scenarios considered in this thesis, such defences are not useful and can even improve an attack's performance. However, we also answer the subsequent questions to provide a more detailed overview.

   a) **To what extent does the effectiveness of model stealing attacks decrease when a certain defence is applied?** We started our defence analysis by comparing the performance of an attack conducted by the strongest attacker against protected and unprotected models. The attack performance was decreased by at most 1.5% in one particular configuration, whereas sometimes it was improved by more than 1%. We did not find a specific pattern to conclude in which settings the defences are helpful. In general, the results are unstable. Even in the best case, the effect of the defences is not significant enough to conclude that they can mitigate a substitute training attack.

   b) **To what extent does the utility of the target model change when a data perturbation defence is applied?** This part of the defence study brought us the most positive results. While we expected that the accuracy of the protected model on the original test set would decrease by 1% for input and output perturbation defences and by up to 2% for their combination, it was not the case. The most notable drop of 0.34% was obtained for the SimpleNet target model protected by input and output perturbation defences. For the ResNet target models, the accuracy scores were even improved in most cases. Based on that, we have two conclusions: (1) these defences do not notably harm the utility of the target model, and (2) one can probably try more significant perturbations than was done in this work.

   c) **To what extent does the utility of defences change when attacker's knowledge about the target model is limited?** Finally, we verified if our observations from attacks launched by the strongest attacker also took place for a weaker attacker. We analysed the performance of defences against an adversary that used the artificial data and ResNet-18 substitute architecture. The range of changes has increased: sometimes, defences decreased the attack's performance by almost 3%, and sometimes improved by nearly 2.5%. In this scenario, the output perturbation defence performed the best. However, the inconsistency and the volume of the attack's performance change do not allow us to recommend this defence as a mitigation technique.

8.4 Future Work

There are several directions in which the future work can be conducted. First of all, some of our experiments showed promising results and suggested that some of our approaches can be improved and developed further. Then, several experiments did not bring us positive results. We propose ways to overcome the problems we met or other approaches that can be tested. Finally, as there are many paths out of the scope of this thesis, we describe other unexplored areas available for further research.

Improving the quality of the artificial data. Our data-free attack is significantly more efficient than previous data-free attacks and even outperforms some of them in terms of effectiveness. However, as we have seen, it is not complex enough compared to the CIFAR10 dataset, which bounds the attack performance. With further investigation, one can verify if this attack can be a real threat and if it should be considered when developing defences against model stealing. Whereas one option is to improve the quality of data generated by a single model, one can also try to combine outputs of different text-to-image models (Stable Diffusion[1], Midjourney[2], DALL-E[3], etc.) to make the dataset more heterogeneous.

Developing guidelines for measuring transferability. Contrary to accuracy and fidelity, there is no unified way to measure transferability. Different adversarial algorithms modify data samples with different amounts of perturbation, resulting in diverse transferability scores. Hence, exploring how transferability changes depending on the algorithm would help to compare different works more fairly. Moreover, one can then develop recommendations on which algorithms to use for transferability evaluation.

Reaching high transferability evaluated using DeepFool. Another challenge for future work is to "fix" the transferability scores obtained in this work. Although we probably picked the most strict evaluation approach, we would like to explore whether achieving high transferability using DeepFool is possible.

Exploring other query optimisation techniques. Adversarial augmentation and active adversarial augmentation mostly performed worse than active learning. However, there could be a more prominent strategy for combining active learning with adversarial

[1]`https://clipdrop.co/stable-diffusion`
[2]`https://docs.midjourney.com/`
[3]`https://openai.com/dall-e-3`

augmentation, or maybe using another algorithm instead of DeepFool is better. Although some research was already done in this direction, we did not obtain the same results, so further investigation is needed.

Investigating if more significant perturbations can lead to a more effective data-perturbation defence. The data-perturbation defences considered in this work did not help to protect target models from stealing. However, during our analysis, we found out that the accuracy of the protected target model decreased notably less than we expected and, in some cases, even increased. That signifies that we could add more perturbation without harming the target model's utility. However, we do not know if that would help to increase the defence effectiveness.

Testing our approaches on other datasets. In this work, we used only one dataset for training target models. Firstly, we would like to confirm that our observations do not depend on a particular dataset used. Secondly, as some works used other datasets, we would like to compare our results with theirs.

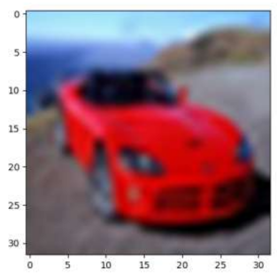
Applying reactive defences. In the thesis, we explore data-perturbation defences, which are proactive approaches. However, since they do not work in considered scenarios, other defence approaches must be studied. In particular, we would like to pay attention to watermarking and monitors. Watermarking allows verifying that the target model was stolen if its illegal copy becomes publicly available with white- or black-box access. Monitors track the behaviour of each client interacting with the target model and hence can detect an attack in its early stage. We would like to investigate if our data-free attack can be detected by one of them.

Exploring other data domains. In the thesis, we focused on the most popular data domain and task within the related work, namely image classification. However, substitute training attacks can be applied to any machine learning model and even to program logic that implements an algorithm. Some domains, like audio, were not explored in the previous work at all. Hence, one can investigate whether the same threats and scenarios apply to other data domains.

Investigating scenarios with an adaptive attacker. An adaptive attacker possesses additional knowledge about defence mechanisms applied to the target model. Consequently, the attacker can adapt their stealing strategy to surpass the defence. Such a scenario is essential for testing defence robustness and, in general, should be considered while developing a new defence approach. For reactive defences like monitors, an adaptive attacker means that the detection criteria are publicly known, hence giving the attacker an obvious hint on how to evade being detected. For proactive defences like data perturbation, an adaptive attacker is assumed to know the perturbation method, which allows to reverse-engineer the modifications in some cases. However, adaptive attackers have not been previously explored in model stealing.

APPENDIX A

Appendix


(a) Real automobile.

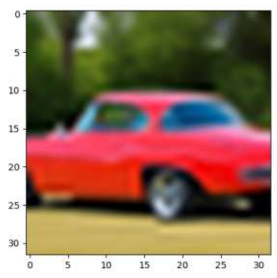
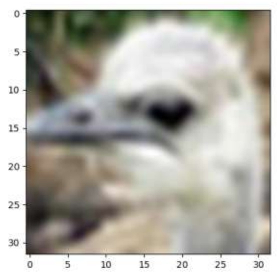

(b) Fake automobile.

Figure A.1: Survey question for the "automobile" class. The real automobile was correctly identified by 66% of respondents.


(a) Real bird.

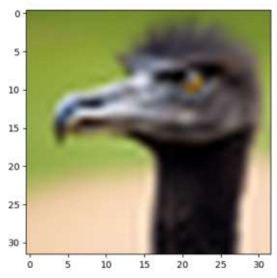
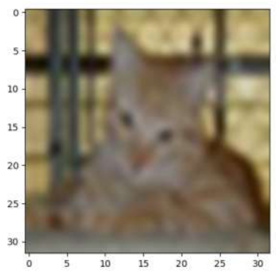
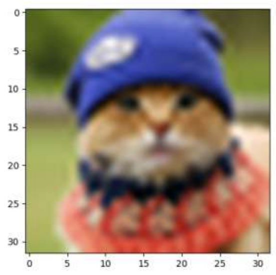

(b) Fake bird.

Figure A.2: Survey question for the "bird" class. The real bird was correctly identified by 54.4% of respondents.

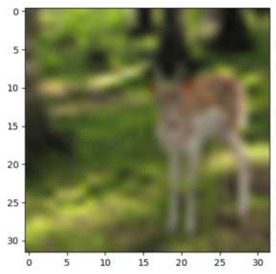


(a) Real cat.

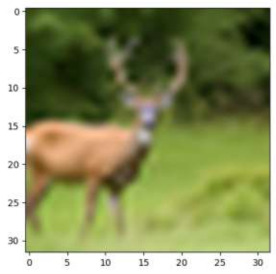


(b) Fake cat.

Figure A.3: Survey question for the "cat" class. The real cat was correctly identified by 30.7% of respondents.

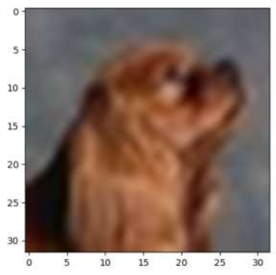


(a) Real deer.

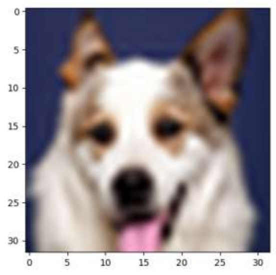


(b) Fake deer.

Figure A.4: Survey question for the "deer" class. The real deer was correctly identified by 39% of respondents.

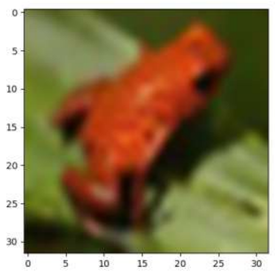


(a) Real dog.

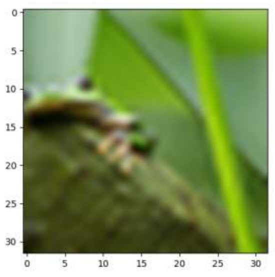


(b) Fake dog.

Figure A.5: Survey question for the "dog" class. The real dog was correctly identified by 67.6% of respondents.

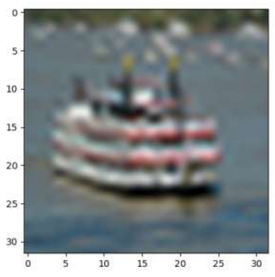


(a) Real frog.

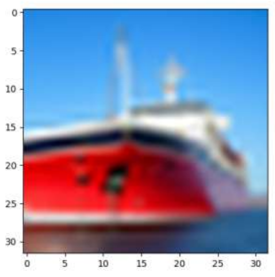


(b) Fake frog.

Figure A.6: Survey question for the "frog" class. The real frog was correctly identified by 44.8% of respondents.

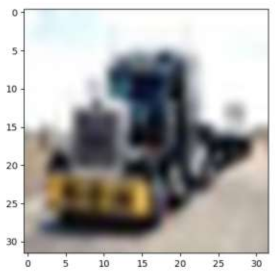


(a) Real ship.

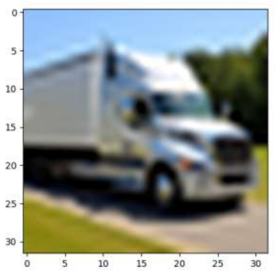


(b) Fake ship.

Figure A.7: Survey question for the "ship" class. The real ship was correctly identified by 60.6% of respondents.



(a) Real truck.



(b) Fake truck.

Figure A.8: Survey question for the "truck" class. The real truck was correctly identified by 27.8% of respondents.

List of Figures

List of Tables

List of Algorithms

Bibliography

[ASJ+20] Buse Gul Atli, Sebastian Szyller, Mika Juuti, Samuel Marchal, and N. Asokan. Extraction of Complex DNN Models: Real Threat or Boogeyman? In *Engineering Dependable and Secure Machine Learning Systems*, volume 1272 of *EDSMLS*, pages 42–57, Cham, 2020. Springer International Publishing.

[BBJP19] Lejla Batina, Shivam Bhasin, Dirmanto Jap, and Stjepan Picek. CSI NN: Reverse Engineering of Neural Network Architectures Through Electromagnetic Side Channel. In *USENIX Security Symposium*, pages 515–532, Santa Clara, CA, August 2019. USENIX Association.

[BCIP20] Antonio Bărbălău, Adrian Cosma, Radu Tudor Ionescu, and Marius Popescu. Black-Box ripper: copying black-box models using generative evolutionary algorithms. In *International Conference on Neural Information Processing Systems*, NeurIPS, pages 20120–20129, Red Hook, NY, USA, December 2020. Curran Associates Inc.

[BH19] Vahid Behzadan and William Hsu. Adversarial Exploitation of Policy Imitation, June 2019. arXiv:1906.01121.

[BR18] Battista Biggio and Fabio Roli. Wild patterns: Ten years after the rise of adversarial machine learning. *Pattern Recognition*, 84:317–331, December 2018.

[CCG+20] V. Chandrasekaran, K. Chaudhuri, I. Giacomelli, S. Jha, and Songbai Yan. Exploring Connections Between Active Learning and Model Extraction. In *USENIX Security Symposium*, August 2020.

[CDG20] Hervé Chabanne, Vincent Despiegel, and Linda Guiga. A Protection against the Extraction of Neural Network Models, July 2020. arXiv:2005.12782.

[CG17] Xiaoyu Cao and Neil Zhenqiang Gong. Mitigating evasion attacks to deep neural networks via region-based classification. In *Proceedings of the 33rd Annual Computer Security Applications Conference*, ACSAC '17, page 278–287, New York, NY, USA, 2017. Association for Computing Machinery.

[CGZ+21] Kangjie Chen, Shangwei Guo, Tianwei Zhang, Xiaofei Xie, and Yang Liu. Stealing Deep Reinforcement Learning Models for Fun and Profit. In *Asia Conference on Computer and Communications Security*, ASIA CCS, pages 307–319, Virtual Event Hong Kong, May 2021. ACM.

[CJM20] Nicholas Carlini, Matthew Jagielski, and Ilya Mironov. Cryptanalytic Extraction of Neural Network Models. In *Advances in Cryptology – CRYPTO 2020*, volume 12172, pages 189–218, Cham, 2020. Springer International Publishing. Series Title: Lecture Notes in Computer Science.

[CSBB+18] Jacson Rodrigues Correia-Silva, Rodrigo F. Berriel, Claudine Badue, Alberto F. de Souza, and Thiago Oliveira-Santos. Copycat CNN: Stealing Knowledge by Persuading Confession with Random Non-Labeled Data. In *International Joint Conference on Neural Networks*, IJCNN, pages 1–8, Rio de Janeiro, July 2018. IEEE.

[CWS+20] Jinyin Chen, Changan Wu, Shijing Shen, Xuhong Zhang, and Jianhao Chen. DAS-AST: Defending Against Model Stealing Attacks Based on Adaptive Softmax Transformation. In *International Conference on Information Security and Cryptology*, volume 12612, pages 21–36, Cham, 2020. Springer International Publishing.

[CXLS22] Abhishek Chakraborty, Daniel Xing, Yuntao Liu, and Ankur Srivastava. DynaMarks: Defending Against Deep Learning Model Extraction Using Dynamic Watermarking, 2022.

[DBK+21] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. In *International Conference on Learning Representations*, 2021.

[DCAS18] Luke N. Darlow, Elliot J. Crowley, Antreas Antoniou, and Amos J. Storkey. Cinic-10 is not imagenet or cifar-10, 2018.

[DDS+09] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. ImageNet: A large-scale hierarchical image database. In *IEEE Conference on Computer Vision and Pattern Recognition*, CVPR, pages 248–255, Miami, FL, June 2009. IEEE.

[Den12] Li Deng. The mnist database of handwritten digit images for machine learning research. *IEEE Signal Processing Magazine*, 29(6):141–142, 2012.

[DKLP22] Adam Dziedzic, Muhammad Ahmad Kaleem, Yu Shen Lu, and Nicolas Papernot. Increasing the Cost of Model Extraction with Calibrated Proof of Work. In *International Conference on Learning Representations*, 2022.

[DP18] Melanie Ducoffe and Frederic Precioso. Adversarial active learning for deep networks: a margin based approach, 2018.

[DR20] David DeFazio and Arti Ramesh. Adversarial Model Extraction on Graph Neural Networks. In *International Workshop on Deep Learning on Graphs: Methodologies and Applications*, DLGMA, New York, NY, USA, February 2020.

[DSRB19] Vasisht Duddu, Debasis Samanta, D. Vijay Rao, and Valentina E. Balas. Stealing Neural Networks via Timing Side Channels, July 2019. arXiv:1812.11720.

[FJR15] Matt Fredrikson, Somesh Jha, and Thomas Ristenpart. Model Inversion Attacks that Exploit Confidence Information and Basic Countermeasures. In *ACM SIGSAC Conference on Computer and Communications Security*, CCS, pages 1322–1333, Denver Colorado USA, October 2015. ACM.

[GCY$^+$21] Xueluan Gong, Yanjiao Chen, Wenbin Yang, Guanghao Mei, and Qian Wang. InverseNet: Augmenting Model Extraction Attacks with Training Data Inversion. In *International Joint Conference on Artificial Intelligence (IJCAI)*, pages 2439–2447, Montreal, Canada, August 2021.

[GR20] L. Guiga and A. W. Roscoe. Neural network security: Hiding CNN parameters with guided grad-CAM. In *ICISSP 2020 - Proceedings of the 6th International Conference on Information Systems Security and Privacy*, pages 611–618, 2020.

[Gra20] Justin Grana. Perturbing Inputs to Prevent Model Stealing. In *IEEE Conference on Communications and Network Security*, CNS, pages 1–9, Avignon, France, June 2020. IEEE.

[HDK$^+$20] Sanghyun Hong, Michael Davinroy, Yiğitcan Kaya, Stuart Nevans Locke, Ian Rackow, Kevin Kulda, Dana Dachman-Soled, and Tudor Dumitraş. Security Analysis of Deep Neural Networks Operating in the Presence of Cache Side-Channel Attacks, January 2020. arXiv:1810.03487.

[HJA20] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 6840–6851. Curran Associates, Inc., 2020.

[HRFS16] Seyyed Hossein Hasanpour, Mohammad Rouhani, Mohsen Fayyaz, and Mohammad Sabokrou. Lets keep it simple, using simple architectures to outperform deeper and more complex architectures, 2016.

[HVD14] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the Knowledge in a Neural Network. In *NIPS 2014 Deep Learning Workshop*, Montréal, Canada, 2014.

[HZRS16] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep Residual Learning for Image Recognition. In *IEEE Conference on Computer Vision and Pattern Recognition*, CVPR, pages 770–778, Las Vegas, NV, USA, June 2016. IEEE.

[IS15] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *Proceedings of the 32nd International Conference on International Conference on Machine Learning - Volume 37*, ICML'15, page 448–456. JMLR.org, 2015.

[JCB$^+$20] Matthew Jagielski, Nicholas Carlini, David Berthelot, Alex Kurakin, and Nicolas Papernot. High Accuracy and High Fidelity Extraction of Neural Networks. In *USENIX Security Symposium*, pages 1345–1362. USENIX Association, August 2020.

[JSMA19] Mika Juuti, Sebastian Szyller, Samuel Marchal, and N. Asokan. PRADA: Protecting Against DNN Model Stealing Attacks. In *IEEE European Symposium on Security and Privacy*, EuroS&P, pages 512–527, Stockholm, Sweden, June 2019. IEEE.

[KH09] Alex Krizhevsky and Geoffrey Hinton. Learning Multiple Layers of Features from Tiny Images. Technical report, University of Toronto, 2009.

[KMAM18] Manish Kesarwani, Bhaskar Mukhoty, Vijay Arya, and Sameep Mehta. Model Extraction Warning in MLaaS Paradigm. In *Annual Computer Security Applications Conference*, ACSAC, pages 371–380, San Juan PR USA, December 2018. ACM.

[KPB$^+$09] Barbara Kitchenham, O. Pearl Brereton, David Budgen, Mark Turner, John Bailey, and Stephen Linkman. Systematic literature reviews in software engineering – a systematic literature review. *Information and Software Technology*, 51(1):7–15, 2009. Special Section - Most Cited Articles in 2002 and Regular Research Papers.

[KPQ21a] Sanjay Kariyappa, Atul Prakash, and Moinuddin K Qureshi. MAZE: Data-Free Model Stealing Attack Using Zeroth-Order Gradient Estimation. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, CVPR, pages 13809–13818, Nashville, TN, USA, June 2021. IEEE.

[KPQ21b] Sanjay Kariyappa, Atul Prakash, and Moinuddin K. Qureshi. Protecting DNNs from Theft using an Ensemble of Diverse Models. In *International Conference on Learning Representations*, 2021.

[KQ20] Sanjay Kariyappa and Moinuddin K. Qureshi. Defending Against Model Stealing Attacks With Adaptive Misinformation. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, CVPR, pages 767–775, Seattle, WA, USA, June 2020. IEEE.

[KTP$^+$20] Kalpesh Krishna, Gaurav Singh Tomar, Ankur Parikh, Nicolas Papernot, and Mohit Iyyer. Thieves of Sesame Street: Model Extraction on BERT-based APIs. In *International Conference on Learning Representations*, ICLR, Virtual Event, April 2020.

[LBOM12] Yann A. LeCun, Léon Bottou, Genevieve B. Orr, and Klaus-Robert Müller. *Efficient BackProp*, pages 9–48. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.

[LEMS19] Taesung Lee, Benjamin Edwards, Ian Molloy, and Dong Su. Defending Against Neural Network Model Stealing Attacks Using Deceptive Perturbations. In *IEEE Security and Privacy Workshops (SPW)*, pages 43–49, San Francisco, CA, USA, May 2019. IEEE.

[LFS20] Hsiao-Ying Lin, Chengfang Fang, and Jie Shi. Bident Structure for Neural Network Model Protection:. In *International Conference on Information Systems Security and Privacy*, ICISSP, pages 377–384, Valletta, Malta, 2020. SciTePress.

[LHL22] Jeonghyun Lee, Sungmin Han, and Sangkyun Lee. Model Stealing Defense against Exploiting Information Leak through the Interpretation of Deep Neural Nets. In *International Joint Conference on Artificial Intelligence (IJCAI)*, pages 710–716, Vienna, Austria, July 2022.

[LJLG22] Yupei Liu, Jinyuan Jia, Hongbin Liu, and Neil Zhenqiang Gong. StolenEncoder: Stealing Pre-trained Encoders in Self-supervised Learning. In *ACM SIGSAC Conference on Computer and Communications Security (CCS)*, Los Angeles, CA, USA, 2022. ACM. event-place: Los Angeles, U.S.A.

[LM05] Daniel Lowd and Christopher Meek. Adversarial Learning. In *ACM SIGKDD International Conference on Knowledge Discovery in Data Mining*, KDD, pages 641–647, Chicago, Illinois, USA, 2005. ACM Press. event-place: Chicago, Illinois, USA.

[LML$^+$22] Xinjing Liu, Zhuo Ma, Yang Liu, Zhan Qin, Junwei Zhang, and Zhuzhu Wang. SeInspect: Defending Model Stealing via Heterogeneous Semantic Inspection. In *European Symposium on Research in Computer Security (ESORICS)*, volume 13554, pages 610–630, Cham, 2022. Springer International Publishing.

[LMR23] Isabell Lederer, Rudolf Mayer, and Andreas Rauber. Identifying Appropriate Intellectual Property Protection Mechanisms for Machine Learning Models: A Systematisation of Watermarking, Fingerprinting, Model Access, and Attacks. *IEEE Transactions on Neural Networks and Learning Systems*, 2023.

[LZK21] Nils Lukas, Yuxuan Zhang, and Florian Kerschbaum. Deep Neural Network Fingerprinting by Conferrable Adversarial Examples. In *International Conference on Learning Representations*, 2021.

[MDFF15] Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, and Pascal Frossard. Deepfool: A simple and accurate method to fool deep neural networks. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2574–2582, 2015.

[MDN19] Itay Mosafi, Eli Omid David, and Nathan S. Netanyahu. Stealing Knowledge from Protected Deep Neural Networks Using Composite Unlabeled Data. In *International Joint Conference on Neural Networks*, IJCNN, pages 1–8, Budapest, Hungary, July 2019. IEEE.

[MHS21] Takayuki Miura, Satoshi Hasegawa, and Toshiki Shibahara. MEGEX: Data-Free Model Extraction Attack against Gradient-Based Explainable AI, July 2021. arXiv:2107.08909 [cs].

[MLF22] Mantas Mazeika, Bo Li, and David Forsyth. How to Steer Your Adversary: Targeted and Efficient Model Stealing Defenses with Gradient Redirection. In *International Conference on Machine Learning (ICML)*, volume 162 of *Proceedings of Machine Learning Research*, pages 15241–15254. PMLR, July 2022.

[MSDH19] Smitha Milli, Ludwig Schmidt, Anca D. Dragan, and Moritz Hardt. Model Reconstruction from Model Explanations. In *Conference on Fairness, Accountability, and Transparency*, FAT, pages 1–9, Atlanta GA USA, January 2019. ACM.

[MYP21] Pratyush Maini, Mohammad Yaghini, and Nicolas Papernot. Dataset Inference: Ownership Resolution in Machine Learning. In *International Conference on Learning Representations*, 2021.

[OAFS18] Seong Joon Oh, M. Augustin, M. Fritz, and B. Schiele. Towards Reverse-Engineering Black-Box Neural Networks. In *International Conference on Learning Representations*, ICLR, Vancouver, B.C., Canada, 2018.

[OMR23] Daryna Oliynyk, Rudolf Mayer, and Andreas Rauber. I know what you trained last summer: A survey on stealing machine learning models and defences. *ACM Comput. Surv.*, apr 2023.

[OSF19] Tribhuvanesh Orekondy, Bernt Schiele, and Mario Fritz. Knockoff Nets: Stealing Functionality of Black-Box Models. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, CVPR, pages 4949–4958, Long Beach, CA, USA, June 2019. IEEE.

[OSF20] Tribhuvanesh Orekondy, Bernt Schiele, and Mario Fritz. Prediction Poisoning: Towards Defenses Against DNN Model Stealing Attacks. In *International Conference on Learning Representations*, ICLR, Virtual Event, April 2020.

[PGKS21] Soham Pal, Yash Gupta, Aditya Kanade, and Shirish Shevade. Stateful Detection of Model Extraction Attacks, July 2021. arXiv:2107.05166 [cs, stat].

[PGS$^+$19] Soham Pal, Yash Gupta, Aditya Shukla, Aditya Kanade, Shirish Shevade, and Vinod Ganapathy. A framework for the extraction of Deep Neural Networks by leveraging public data, May 2019. arXiv:1905.09165.

[PGS$^+$20] Soham Pal, Yash Gupta, Aditya Shukla, Aditya Kanade, Shirish Shevade, and Vinod Ganapathy. ActiveThief: Model Extraction Using Active Learning and Unannotated Public Data. In *AAAI Conference on Artificial Intelligence*, volume 34, pages 865–872, April 2020.

[PMG16] Nicolas Papernot, Patrick McDaniel, and Ian Goodfellow. Transferability in Machine Learning: from Phenomena to Black-Box Attacks using Adversarial Samples, May 2016. arXiv:1605.07277.

[PMG$^+$17] Nicolas Papernot, Patrick McDaniel, Ian Goodfellow, Somesh Jha, Z. Berkay Celik, and Ananthram Swami. Practical Black-Box Attacks against Machine Learning. In *ACM Asia Conference on Computer and Communications Security*, ASIA CCS, pages 506–519, Abu Dhabi United Arab Emirates, April 2017. ACM.

[PYZ18] Li Pengcheng, Jinfeng Yi, and Lijun Zhang. Query-Efficient Black-Box Attack by Active Learning. In *IEEE International Conference on Data Mining*, ICDM, pages 1200–1205, Singapore, November 2018. IEEE.

[RBL$^+$22] Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 10684–10695, June 2022.

[RK20] David Rolnick and Konrad Kording. Reverse-engineering deep ReLU networks. In Hal Daumé III and Aarti Singh, editors, *International Conference on Machine Learning*, volume 119 of *ICML*, pages 8178–8187. PMLR, July 2020.

[Ros58] F. Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65(6):386–408, 1958.

[RPM19] Nicholas Roberts, Vinay Uday Prabhu, and Matthew McAteer. Model Weight Theft With Just Noise Inputs: The Curious Case of the Petulant Attacker. In *ICML Workshop on the Security and Privacy of Machine Learning*, Long Beach, CA, June 2019.

[RST19] Robert Nikolai Reith, Thomas Schneider, and Oleksandr Tkachenko. Efficiently Stealing Your Machine Learning Models. In *ACM Workshop on Privacy in the Electronic Society*, WPES, pages 198–210, London, United Kingdom, 2019. ACM Press.

[Rud16] Sebastian Ruder. An overview of gradient descent optimization algorithms, 2016. arXiv:1609.04747.

[SAAH20] Kálmán Szentannai, Jalal Al-Afandi, and András Horváth. Preventing Neural Network Weight Stealing via Network Obfuscation. In *Computing Conference*, volume 1230, pages 1–11, Cham, 2020. Springer International Publishing.

[SAB22] Sunandini Sanyal, Sravanti Addepalli, and R. Venkatesh Babu. Towards Data-Free Model Stealing in a Hard Label Setting. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 15263–15272, New Orleans, LA, USA, June 2022. IEEE.

[SAMA21] Sebastian Szyller, Buse Gul Atli, Samuel Marchal, and N. Asokan. DAWN: Dynamic Adversarial Watermarking of Neural Networks. In *ACM International Conference on Multimedia*, pages 4417–4425, Virtual Event China, October 2021. ACM.

[SBV$^+$22] Christoph Schuhmann, Romain Beaumont, Richard Vencu, Cade W Gordon, Ross Wightman, Mehdi Cherti, Theo Coombes, Aarush Katta, Clayton Mullis, Mitchell Wortsman, Patrick Schramowski, Srivatsa R Kundurthy, Katherine Crowson, Ludwig Schmidt, Robert Kaczmarczyk, and Jenia Jitsev. LAION-5b: An open large-scale dataset for training next generation image-text models. In *Thirty-sixth Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2022.

[SCD$^+$17] Ramprasaath R. Selvaraju, Michael Cogswell, Abhishek Das, Ramakrishna Vedantam, Devi Parikh, and Dhruv Batra. Grad-CAM: Visual Explanations from Deep Networks via Gradient-Based Localization. In *IEEE International Conference on Computer Vision*, ICCV, pages 618–626, Venice, October 2017. IEEE.

[SDGA21] Sebastian Szyller, Vasisht Duddu, Tommi Gröndahl, and N. Asokan. Good Artists Copy, Great Artists Steal: Model Extraction Attacks Against Image Translation Generative Adversarial Networks, April 2021. arXiv:2104.12623 [cs].

[SDWMG15] Jascha Sohl-Dickstein, Eric A. Weiss, Niru Maheswaranathan, and Surya Ganguli. Deep unsupervised learning using nonequilibrium thermodynamics. In *Proceedings of the 32nd International Conference on International Conference on Machine Learning - Volume 37*, ICML'15, page 2256–2265. JMLR.org, 2015.

[SS17] Yi Shi and Yalin E. Sagduyu. Evasion and causative attacks with adversarial deep learning. In *IEEE Military Communications Conference*, MILCOM, pages 243–248, Baltimore, MD, October 2017. IEEE.

[SS18] Ozan Sener and Silvio Savarese. Active learning for convolutional neural networks: A core-set approach. In *International Conference on Learning Representations*, 2018.

[SSDJ22] Amir Mahdi Sadeghzadeh, Amir Mohammad Sobhanian, Faezeh Dehghan, and Rasool Jalili. HODA: Hardness-Oriented Detection of Model Extraction Attacks, February 2022. arXiv:2106.11424 [cs].

[STIM18] Shibani Santurkar, Dimitris Tsipras, Andrew Ilyas, and Aleksander Mądry. How does batch normalization help optimization? In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, NIPS'18, page 2488–2498, Red Hook, NY, USA, 2018. Curran Associates Inc.

[SZS$^+$14] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian J. Goodfellow, and Rob Fergus. Intriguing properties of neural networks. In *International Conference on Learning Representations,*, ICLR, Banff, AB, Canada, April 2014.

[TCBM20] Florian Tramer, Nicholas Carlini, Wieland Brendel, and Aleksander Madry. On Adaptive Attacks to Adversarial Example Defenses. In *Advances in Neural Information Processing Systems*, volume 33, pages 1633–1645. Curran Associates, Inc., 2020.

[TMWP21] Jean-Baptiste Truong, Pratyush Maini, Robert J. Walls, and Nicolas Papernot. Data-Free Model Extraction. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4769–4778, Nashville, TN, USA, June 2021. IEEE.

[TYF20] Tatsuya Takemura, Naoto Yanai, and Toru Fujiwara. Model Extraction Attacks against Recurrent Neural Networks, January 2020. arXiv:2002.00123.

[TZJ$^+$16] Florian Tramèr, Fan Zhang, Ari Juels, Michael K. Reiter, and Thomas Ristenpart. Stealing Machine Learning Models via Prediction APIs. In *USENIX Security Symposium*, pages 601–618, Austin, TX, USA, August 2016. USENIX Association.

[WG15] Haibing Wu and Xiaodong Gu. Towards dropout training for convolutional neural networks. *Neural Netw.*, 71(C):1–10, nov 2015.

[WG18] Binghui Wang and Neil Zhenqiang Gong. Stealing Hyperparameters in Machine Learning. In *IEEE Symposium on Security and Privacy*, SP, pages 36–52, San Francisco, CA, May 2018. IEEE.

[WH00] R. Wirth and Jochen Hipp. Crisp-dm: Towards a standard process model for data mining. *Proceedings of the 4th International Conference on the Practical Applications of Knowledge Discovery and Data Mining*, 01 2000.

[WL22] Yixu Wang and Xianming Lin. Enhance Model Stealing Attack via Label Refining. In *International Conference on Intelligent Computing and Signal Processing (ICSP)*, pages 1040–1043, Xi'an, China, April 2022. IEEE.

[WLL$^+$22] Yixu Wang, Jie Li, Hong Liu, Yan Wang, Yongjian Wu, Feiyue Huang, and Rongrong Ji. Black-Box Dissector: Towards Erasing-Based Hard-Label Model Stealing Attack. In *European Conference on Computer Vision (ECCV)*, volume 13665, pages 192–208, Cham, 2022. Springer Nature Switzerland.

[XHZ$^+$22] Yi Xie, Mengdie Huang, Xiaoyu Zhang, Changyu Dong, Willy Susilo, and Xiaofeng Chen. GAME: Generative-Based Adaptive Model Extraction Attack. In *European Symposium on Research in Computer Security (ES-ORICS)*, volume 13554, pages 570–588, Cham, 2022. Springer International Publishing.

[XSZ$^+$18] Hui Xu, Yuxin Su, Zirui Zhao, Yangfan Zhou, Michael R. Lyu, and Irwin King. DeepObfuscation: Securing the Structure of Convolutional Neural Networks via Knowledge Distillation, June 2018. arXiv:1806.10313.

[YCBL14] Jason Yosinski, Jeff Clune, Yoshua Bengio, and Hod Lipson. How transferable are features in deep neural networks? In *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'14, page 3320–3328, Cambridge, MA, USA, 2014. MIT Press.

[YDZ$^+$22] Xiaoyong Yuan, Leah Ding, Lan Zhang, Xiaolin Li, and Dapeng Oliver Wu. ES Attack: Model Stealing Against Deep Neural Networks Without Data Hurdles. *IEEE Transactions on Emerging Topics in Computational Intelligence*, pages 1–13, 2022.

[YHL$^+$22] Anli Yan, Ruitao Hou, Xiaozhang Liu, Hongyang Yan, Teng Huang, and Xianmin Wang. Towards explainable model extraction attacks. *International Journal of Intelligent Systems*, 37(11):9936–9956, November 2022.

[YYZ$^+$20] Honggang Yu, Kaichen Yang, Teng Zhang, Yun-Yun Tsai, Tsung-Yi Ho, and Yier Jin. CloudLeak: Large-Scale Deep Learning Models Stealing

Through Adversarial Examples. In *Network and Distributed System Security Symposium (NDSS)*, San Diego, CA, 2020. Internet Society.

[ZCW21] Zhanyuan Zhang, Yizheng Chen, and David Wagner. SEAT: Similarity Encoder by Adversarial Training for Detecting Model Extraction Attack Queries. In *ACM Workshop on Artificial Intelligence and Security*, pages 37–48, Virtual Event Republic of Korea, November 2021. ACM.

[ZCZL21] Yuankun Zhu, Yueqiang Cheng, Husheng Zhou, and Yantao Lu. Hermes Attack: Steal DNN Models with Lossless Inference Accuracy. In *USENIX Security Symposium*, August 2021.

[ZFS21] Xinyi Zhang, Chengfang Fang, and Jie Shi. Thief, Beware of What Get You There: Towards Understanding Model Extraction Attack, April 2021. arXiv:2104.05921 [cs].