Model Extraction and Active Learning

A THESIS

SUBMITTED FOR THE DEGREE OF

Master of Technology (Research)

IN THE

Computer Science and Engineering

BY

Aditya Shukla

Department of Computer Science and Automation
Indian Institute of Science
Bangalore – 560 012 (INDIA)

January, 2020

Declaration of Originality

I, **Aditya Shukla**, with SR No. **04-04-00-10-22-17-1-14940** hereby declare that the material presented in the thesis titled

Model Extraction and Active Learning

represents original work carried out by me in the **Department of Computer Science and Automation** at **Indian Institute of Science** during the years **2017-20**.
With my signature, I certify that:

- I have not manipulated any of the data or results.
- I have not committed any plagiarism of intellectual property. I have clearly indicated and referenced the contributions of others.
- I have explicitly acknowledged all collaborative research and discussions.
- I have understood that any false claim will result in severe disciplinary action.
- I have understood that the work may be screened for any form of academic misconduct.

Date: Wednesday 29$^{\text{th}}$ January, 2020 Student Signature

In my capacity as supervisor of the above-mentioned work, I certify that the above statements are true to the best of my knowledge, and I have carried out due diligence to ensure the originality of the report.

Advisor Name: Prof. Vinod Ganapathy Advisor Signature

Acknowledgements

First and foremost, I would like to extend my gratitude to my advisor Prof. Vinod Ganapathy for his invaluable guidance and support throughout my master's degree. He has always been very supportive and gave me a lot of freedom to choose a research problem of my interest. He has always been available to discuss any research ideas I had and to figure out how exactly to pursue it further. I have always been fascinated by his quick replies during our email conversations. Without all his encouragement and support, this thesis would not have been possible.

I thank Soham Pal and Yash Gupta, for giving conducive inputs in our collaborative work. The regular discussions we use to have were very helpful in formalizing and solving the problem. The learning experience I have had would have been incomplete without you both. I would also like to thank Prof. Aditya Kanade and Prof. Shirish Shevade for their guidance in our collaborative work.

Next, I would like to thank my lab mates at the Computer Systems Security Laboratory (CSSL) – Subhendu, Rounak, Kripa, Abhishek, Nikita, Arun, Rakesh, Ajay, Abhinivesh, Chinmay and Rishabh for making the lab a fun place. Also, I thank all my friends, including Abhishek, Monish, Vishal, Lokesh, Samadhan, Praveen, Anshuman, Gaurav, Sandeep, Saurabh for making my IISc campus life fun and enjoyable.

I would also like to thank all the office staff in the Department of Computer Science and Automation (CSA), including Mrs. Padmavathi, Mrs. Kushael, Mrs. Meenakshi, Mrs. Nishitha. Their effort to make administrative tasks smooth and easy for all the members of the department is invaluable.

Finally, I would like to especially thank my parents and siblings for their continuous love, support and guidance.

Abstract

Machine learning models are increasingly being offered as a service by big companies such as Google, Microsoft and Amazon. They use Machine Learning as a Service (MLaaS) to expose these machine learning models to the end-users through cloud-based Application Programming Interface (API). Such APIs allow users to query ML models with data samples in a black-box fashion, returning only the corresponding output predictions. MLaaS models are generally monetized by billing the user for each query made. Prior work has shown that it is possible to extract these models. They developed model extraction attacks that extract an approximation of the MLaaS model by making black-box queries to it. However, none of them satisfy all the four criteria essential for practical model extraction: (i) the ability to extract deep learning models, (ii) non-requirement of domain knowledge, (iii) the ability to work with a limited query budget and (iv) non-requirement of annotations. In collaboration with Pal et al., we propose a novel model extraction attack that makes use of active learning techniques and unannotated public data to satisfy all the aforementioned criteria. However, as we show in the experiments, no one active learning technique is well-suited for different datasets and under different query budget constraints. Given the plethora of active learning techniques at the adversary's disposal and the black-box nature of the model under attack, the choice of the technique to be used is difficult but integral: the chosen technique is a strong determinant of the quality of the extracted model. In this work, we wish to devise an active learning technique that combines the benefits of existing active learning techniques, as applicable to different budgets and different datasets, yielding on average extracted models that exhibit a high test agreement with the MLaaS model. In particular, we show that a combination of the DFAL technique of Ducoffe et al. and the Coreset technique of Sener et al. is able to leverage the benefits of both the base techniques, outperforming both DFAL and Coreset in a majority of our experiments. The model extraction attack using this technique achieves, on average, a performance of $4.70\times$ over uniform noise baseline by using only 30% ($30,000$ data samples) of the unannotated public data. Moreover, the attack using this technique remains undetected by PRADA, a state-of-the-art model extraction detection method.

Keywords

Machine learning, Deep neural networks, Model extraction, Active learning, Security

Publications based on this Thesis

ActiveThief: Model Extraction Using Active Learning and Unannotated Public Data. Soham Pal, Yash Gupta, Aditya Shukla, Aditya Kanade, Shirish Shevade, Vinod Ganapathy. Thirty-Fourth AAAI Conference on Artificial Intelligence (AAAI-20). https://www.csa.iisc.ac.in/~vg/papers/aaai2020/aaai2020.pdf

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

In recent years, a popular class of machine learning (ML) models known as Deep Neural Networks (DNNs) have gained enormous popularity due to it's state-of-the-art performance in well known hard problems such as image recognition [5], video analysis [1], natural language processing [43, 17], time series forecasting [51], recommendation system [9] and others. Given its outstanding performance, many companies provide them as solutions to the aforementioned problems among many others. They use Machine Learning as a Service (MLaaS) such as Amazon's Amazon ML, Microsoft's Azure ML, IBM's Watson and Google's Cloud ML to expose these DNNs to the end-users through cloud based Application Programming Interface (API). Such APIs allow users to query these trained DNNs with data samples in a black-box fashion, returning only the corresponding output predictions. MLaaS models are generally monetized by billing the user for each query made.

Even though MLaaS can make these DNNs accessible to the mass population, it is shown to be vulnerable to model extraction attacks [45, 31, 28, 6]. An adversary can use these attacks to extract a copy of a machine learning model and use it to make unlimited free queries to it. It may also distribute extracted model anonymously over the internet which is a huge threat to the business model of the company. Moreover, the adversary can also use the extracted model to instrument following applications which requires gradient information. First, it can use the extracted model to craft adversarial data samples that are transferable to the MLaaS model [31]. For instance, an adversary having access to a extracted copy of a spam classifier can use it to craft spam mails which would be classified as non-spam by both the extracted copy and the original copy of the classifier. Second, the extracted model can also be used to infer data samples on which the MLaaS model was trained on, this process is popularly known as model inversion [10]. It is big risk to the confidentiality of the data of the people who shared it. For example, Fredrikson et al. [10] illustrated in their work that with the help of an extracted

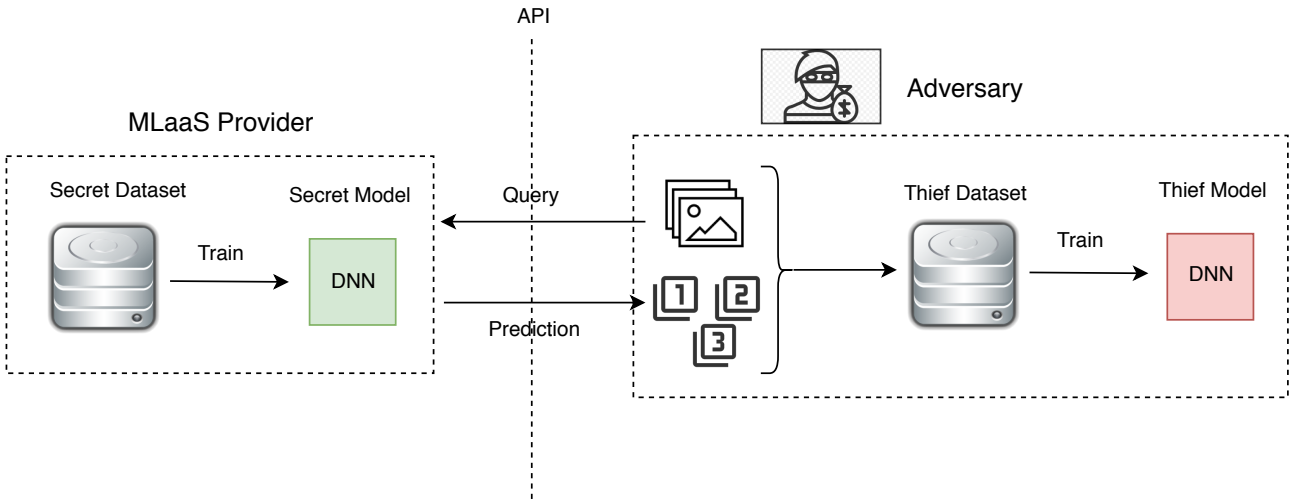


Figure 1.1: Overview of model extraction

copy of a facial recognition system and a person's name, it is possible to get the image of the corresponding person's face. Third, Bastani et al. [2] show that it is possible to interpret statistical properties of the complex MLaaS model by extracting it in an interpretable model such as decision trees.

The overall process of the model extraction is presented in Figure 1.1 in an abstract way. There are two parties involved in the process – MLaaS provider and Adversary. MLaaS provider trains a **secret model** on a **secret dataset**. The secret model is exposed to the end-users via an API, wherein they can query the secret model with input data sample and it returns back the corresponding output prediction. Since the input-output format of the API is public, it is fair to assume that the adversary knows how to format the query input and how to interpret the output prediction. As the adversary may not have access to the secret dataset (on which secret model is trained on), they curate **thief dataset** which can be used to query and extract the secret model. Thief dataset samples are queried for corresponding labels, then it is used to train adversary's **thief model** which approximates the functionality of the secret model.

With reference to Table 1.1, we compare our model extraction attack with prior extraction approaches on various criteria which we believe are essential to make a model extraction attack more practical to implement:

- *Ability to extract DNNs*: It refers to the ability of a model extraction attack to extract secret models that use DNN architectures. It is an important criterion because, given their capability, DNNs are increasingly being adopted to solve very well know hard problems and hence are generally used as a secret model in MLaaS. Except Tramèr et al. [45],

Table 1.1: Comparison of our model extraction framework against prior approaches.

Model extraction technique	Extracts deep neural networks	No problem domain dataset required	No annotations required	Works with a limited query budget
Tramèr et al. [45]	✗	✓	✓	✓
Papernot et al. [31]	✓	✗	✓	✓
Orekondy et al. [28]	✓	✓	✗	✓
Correia-Silva et al. [6]	✓	✓	✓	✗
Our framework	✓	✓	✓	✓

all other approaches including our model extraction framework works on DNNs. Tramèr et al. show applicability of their technique on only basic machine learning classifiers such as logistic regression, SVMs, decision trees and a shallow feed-forward neural network with one hidden layer. We also demonstrate in our experiments that their approach does not scale well to the DNN architecture we use.

- *No problem domain dataset required*: It refers to the kind of thief dataset an adversary uses. In model extraction literature, researchers have used variety of thief datasets. It can be categorised into three types – Problem Domain (**PD**) dataset, Synthetic Non-Problem Domain (**SNPD**) dataset and Natural Non-Problem Domain (**NNPD**) dataset. PD dataset belongs to the distribution of the secret dataset itself, having access to such a thief dataset is a strong assumption to make as it can be very hard and expensive to get it even in small quantity. For example, to extract a secret model trained on medical images, a technique based on the PD dataset would require access to medical images. Papernot et al. [31] assumed access to either a subset of secret (PD) dataset or fabricate one which closely resembles it. Hence, this criterion just discourages the use of PD data for model extraction. Tramèr et al. [45] used SNPD data to perform model extraction, which is sampled from standard probability distributions like uniform distribution. Correia-Silva et al. [6], Orekondy et al. [28] and our model extraction framework uses NNPD data, which on the other hand, is sampled from publicly available data of same content type as of secret dataset (by content type we mean if secret model is trained on images then content type is image, similarly, if it is trained on text then content type is text). Since NNPD data has more natural samples to query the secret model with, it is shown to be more effective than SNPD to be able to trigger more output classes (or labels) of the secret model. Hence, better chances of model extraction with NNPD.

- *No annotations required*: By annotations, we refer to the categorical information of thief dataset samples. For example, we use ImageNet [33] as thief dataset, thus, categor-

ical information is its sample's true classes such as espresso, volleyball, speedboat, etc. Orekondy et al. [28] require hierarchical annotations of ImageNet [33] to make their model extraction technique work well. It is a strong assumption as it might not always be feasible to get high-quality annotations of the thief dataset. Unlike Orekondy et al., all other approaches including our model extraction framework works on an unannotated thief dataset, i.e, we do not require annotations at all. Our extraction framework makes use of freely and easily available unannotated public data to perform model extraction.

- *Works with a limited query budget*: It refers to the ability of a model extraction approach to extract a secret model in limited number of queries to it. It is an essential criterion because generally each input query to a secret model is billed on a per-query basis. Except Correia-Silva et al. [6], all other approaches including our model extraction framework considers a limited query budget criterion.

As can be seen from Table 1.1, we are the first one to satisfy all the four criteria of practical model extraction. We extract DNNs without using domain knowledge or annotations while working in a limited query budget. To achieve this, we needed a solution that could filter out freely and easily available huge amounts of unannotated public data to a very small subset of informative samples. We observed that this objective is very similar to the objective of *active learning* [36]. Active learning is a special case of machine learning in which learning algorithm can interactively query an expert (or oracle) to label new data samples. It is generally applied in situations where unlabelled data is present in an abundant amount but manually labeling it is expensive. Learning algorithms in such a case actively query an expert (or oracle) to label a few informative data samples selected out of the complete set. Figure 1.2 shows complete overview of active learning. Initially, active learner trains a model on a very few labeled samples of the abundant unlabelled data. It then uses the trained model to select interesting or informative samples from the abundant unlabelled data, which is queried to the expert or oracle for its corresponding labels. The model is then trained on this updated subset of labeled data. Steps 2-5 in Figure 1.2 goes on in the loop until the model has achieved a desired accuracy. Note that this process is similar to model extraction except for the part that instead of expert (or oracle) we have a secret model, which labels the input queries. Thus, our model extraction framework exploits freely available abundant unannotated public data (NNPD) by using pool-based active learning techniques (defined in Section 2.2) to achieve all four criteria of practical model extraction.

However, as we show in the experiments (Section 5.2), no one active learning technique is well-suited for different datasets and under different query budget constraints. Given the

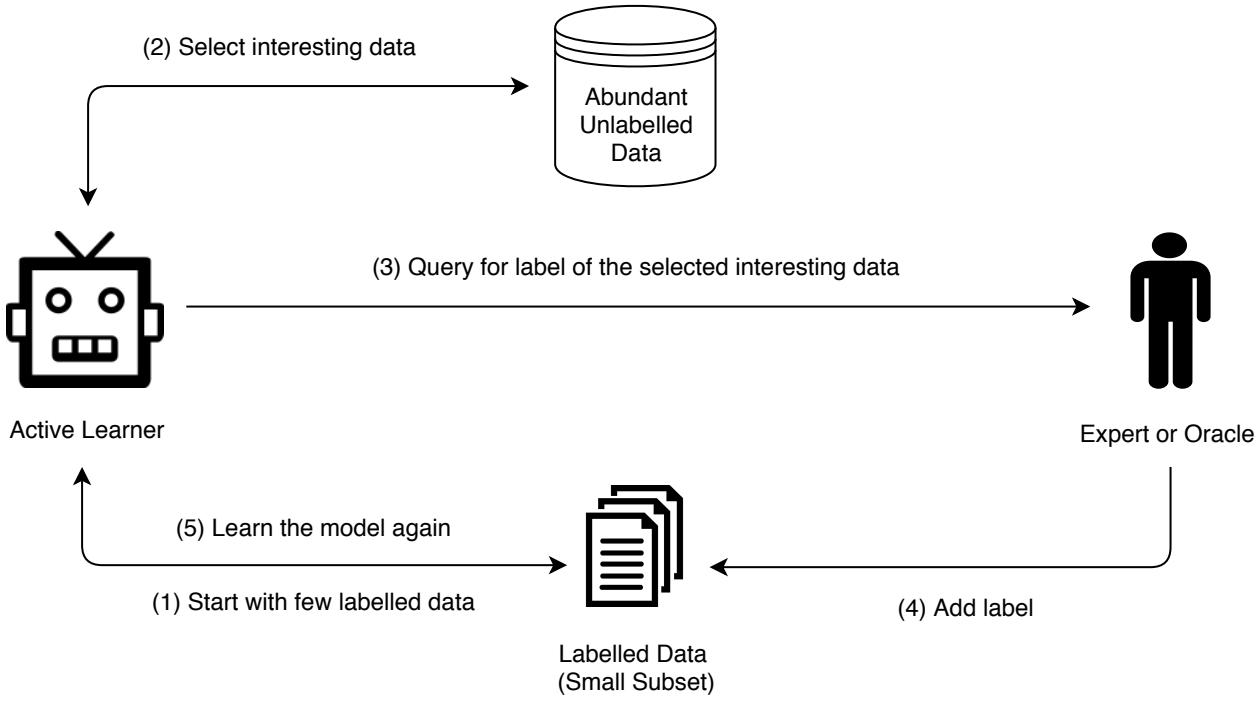


Figure 1.2: Overview of active learning

plethora of active learning techniques at the adversary's disposal and the black-box nature of the model under attack, the choice of the technique to be used is difficult but integral: the chosen technique is a strong determinant of the quality of the extracted model. In this work, we wish to devise an active learning technique that combines the benefits of existing active learning techniques, as applicable to different budgets and different datasets, yielding on average extracted models that exhibit a high test agreement with the MLaaS model. In particular, we show that a combination of the DFAL technique of Ducoffe and Precioso [7] and the Coreset technique of Sener and Savarese [34] is able to leverage the benefits of both the base techniques, outperforming both DFAL and Coreset in a majority of our experiments. We evaluate our technique on four different tasks – MNIST [20], Fashion-MNIST [48], CIFAR-10 [19], GTSRB [41]. The model extraction attack using this technique achieves, on an average, a performance of 4.70× over uniform noise baseline of Tramèr et al. [45] by using only 30% (30,000 data samples) of the unannotated public data.

Finally, it is evident through prior and this work that model extraction attacks are indeed a threat to the confidentiality of the MLaaS models. In light of this, Juuti et al. [16] proposed a generic model extraction detection framework, PRADA (Protecting against DNN Model Stealing Attacks), which analyses the distribution of the distances between successive queries from end-users. It checks for the degree of deviation of the distribution with the normal

distribution. The end-user is considered an adversary if the deviation is beyond a threshold. We empirically show that the attack using our DFAL+Coreset ensemble technique remains undetected by PRADA.

1.1 Collaboration and Contribution of the Thesis

This work was done in collaboration with Pal et al. [29], of which my key contributions are summarized as follows:

1. We define the notion of ensemble active learning. In particular, we instantiate a new ensemble active learning technique, *DFAL+Coreset* and demonstrate its efficacy, comparing and contrasting with existing active learning techniques.

2. We demonstrate that the models extracted using the DFAL+Coreset technique exhibit more consistent performance than the existing active learning techniques, with it consistently producing extracted models that have a higher agreement with the confidential MLaaS model.

3. Finally, we show that the model extraction attack using the DFAL+Coreset active learning technique is not detected by the state-of-the-art model extraction detection method, PRADA.

1.2 Outline of the Thesis

The rest of the thesis is organized as follows:

- **Chapter 2:** We provide necessary background details in this chapter required to understand the thesis. It consists of defining Deep Neural Networks (DNNs), active learning, explaining general adversarial sample generation process and defining Papernot's attack.

- **Chapter 3:** We define the threat model our extraction framework follows which includes defining attack surface, capabilities and adversary's goal.

- **Chapter 4:** Our model extraction framework is explained in this chapter. We also explain a general active learning algorithm and give details about all the existing active learning techniques used in our model extraction framework and introduce a new DFAL+Coreset ensemble active learning technique.

- **Chapter 5:** We evaluate the effectiveness of the new DFAL+Coreset ensemble active learning technique by comparing it with the existing active learning techniques and evaluating its effectiveness with our model extraction framework. Finally, we show our results on PRADA evasion.
- **Chapter 6:** All the related work is explained in this chapter. We categorize them into model extraction, active learning and model reverse-engineering domains.
- **Chapter 7:** In this chapter, we conclude our thesis and provide directions for future work.

Chapter 2

Background

In this chapter, we first talk about DNNs – their general architecture, training phase and testing phase. We then introduce active learning from machine learning literature. We also discuss about adversarial sample generation process which is used in one of the existing active learning technique. Lastly, we describe a prior model extraction attack which we compare against (in context of PRADA evasion).

2.1 Deep Neural Network (DNN)

A Deep Neural Network (DNN), as shown in Figure 2.1 and stated in [31], is a special class of machine learning model that uses a hierarchical composition of n parametric functions to model an input x. Each function f_i, where i varies from 1 to n, is modeled using a layer of neurons. These neurons are elementary computing units which applies an activation function on the weighted representation of the previous layer to generate a new representation. Each layer is assigned a weight vector θ_i, which impacts each neurons activation. Such weights hold the knowledge of a DNN model F. Thus, a DNN model F is defined as follows:

$$F(x) = f_n(\theta_n, f_{n-1}(\theta_{n-1}, \dots f_2(\theta_2, f_1(\theta_1, x))))$$

During the *training phase* of the DNN model F, it learns values for its parameters $\theta_F = \{\theta_1, \dots, \theta_n\}$. In this work, we focus on classification tasks where the goal is to assign a label l_j to the input sample x from a predefined set of labels $l_1, \dots, l_Q$. The DNN is given a large set of known x and y pairs, i.e., (x, y). It is trained on these pairs by adjusting weight parameters to reduce a cost quantifying the prediction error $F(x)$ and the true output y. The adjustment is usually performed using techniques derived from the back-propagation algorithm. Such techniques successively propagate the error gradients with respect to network parameters from

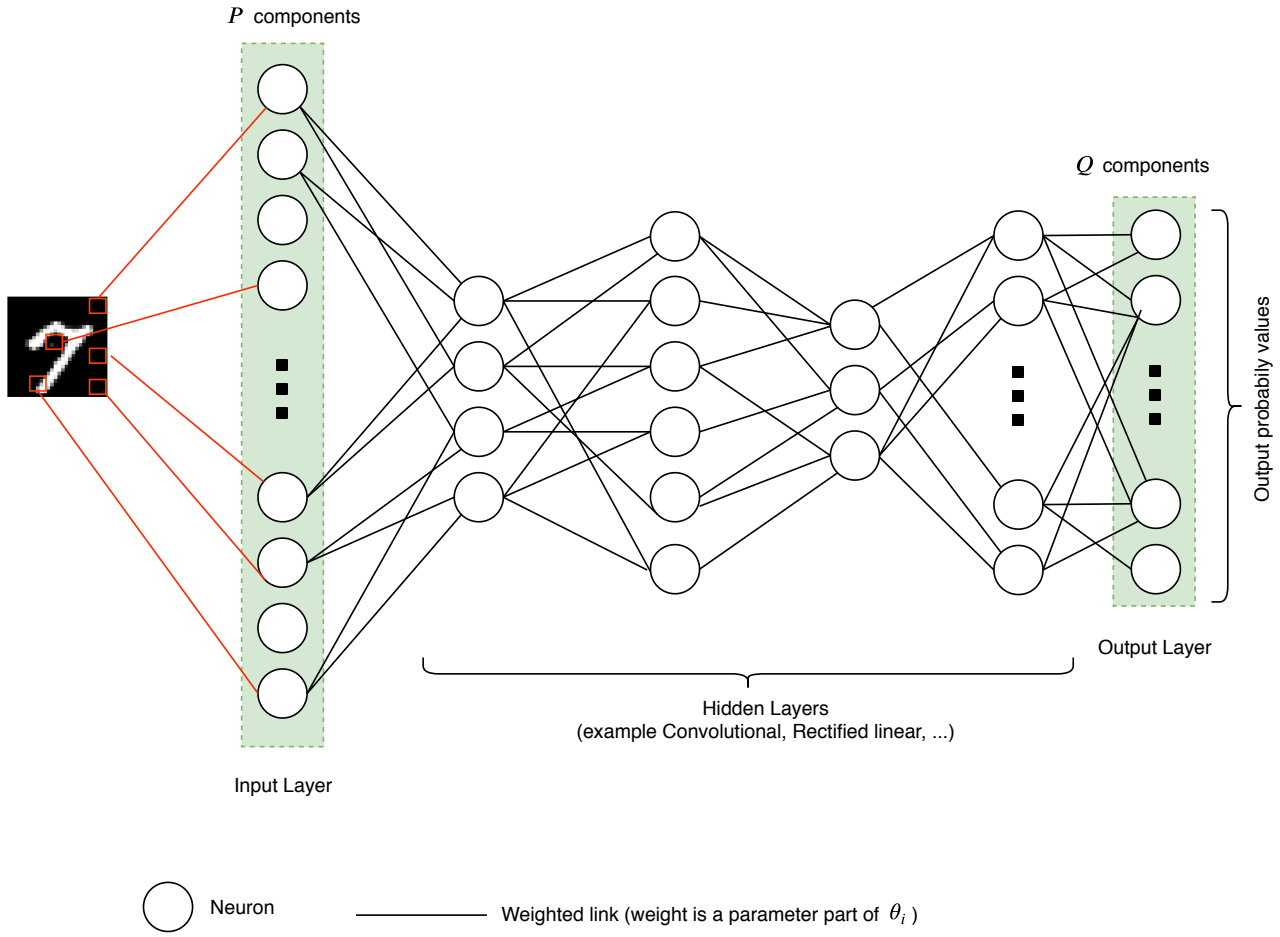


Figure 2.1: General DNN architecture.

the output layer to the input layer of the network.

During the *testing phase* of the DNN model F, it's parameters set θ_F is fixed, i.e., values of all the parameters is no longer changed. These set of parameters are now used by the DNN model F to make predictions on input samples not seen during the training phase. DNNs have shown good generalization ability, when trained on large training datasets, , across diversity of tasks in various domains [21]. The DNN architectures we use in our experiments are defined in Sub-section 5.1.3.

2.2 Active learning

The idea behind active learning [36] is that if a learning algorithm is allowed to choose the data samples to learn from, it will be able to learn the model quickly with very less number of training data samples. It is specially helpful in supervised learning tasks where getting labelled data samples is difficult, time consuming or expensive. For an example, in the task of

classification of images, video, audio, documents etc, it is required that the user label each of the files as "relevant" or "not-relevant", which can be very tedious as number of such files can range from millions to billions.
There are typically three scenarios in which an active learner can query the labels of data samples:

- **Membership Query Synthesis:** In this setting, learner generates/constructs data samples from underlying natural distribution. For example, if underlying distribution is similar to MNIST, learner would construct images similar to digits. It would then query it to the oracle to get the labels.

- **Stream-Based Selective Sampling:** Assumption here is that getting unlabeled data samples is either free or inexpensive. Therefore, learner can sample data points from the actual distribution and can then decide whether to query for it's label or not. This decision is typically made based on informativeness of the sampled data point. Informativeness can be measured by various active learning strategies, one of which is, uncertainty [23].

- **Pool-Based sampling:** In this setting, it is assumed that a large pool of unlabeled samples are available. Some samples are drawn from the pool based on informativeness of the samples. The informative measure is applied on all samples of the pool and then the most informative one(s) are selected. This is the most common scenario in active learning community.
  The main difference between stream-based and pool-based active learning is that stream-based scans data samples sequentially and decides to query for its label on an individual basis, whereas pool-based active learning determines the most informative sample(s) among all the samples present in the pool collectively by ranking them using the informative measure.

We use pool-based active learning scenario in our model extraction framework. Particularly, we use it in each iteration of our extraction algorithm to query a batch of unlabeled samples and add it to a growing subset. Please refer to Sub-section 4.1.1 for more details about the specific active learning techniques used in our extraction algorithm.

2.3 Adversarial sample generation (DeepFool technique)

Moosavi-Dezfooli et al. [25] introduced the DeepFool technique, which is used to generate adversarial samples. This technique is used by one of the active learning technique, DeepFool based Active Learning (DFAL), introduced in Sub-section 4.1.1.

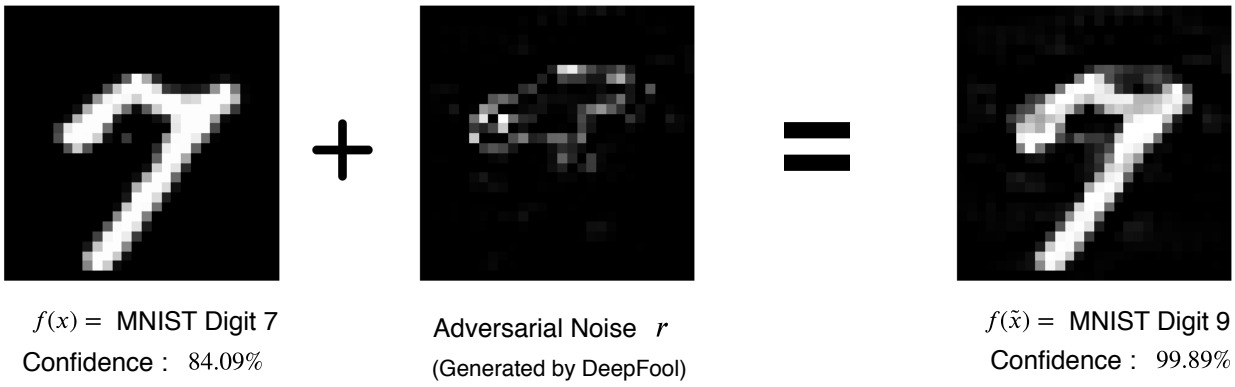


Figure 2.2: Adversarial sample generated using DeepFool [25].

The concept of adversarial example was introduced by Szegedy et al. [44]. They were the first ones to notice that mappings of DNNs are so discontinuous. They designed an optimization procedure to search for a small noise which when added to the natural input can lead the model to output differently. The noise can be small enough that any human can barely notice it. Many different approaches to generate adversarial examples were proposed in literature: the Fast Gradient Sign Method (FGSM) of Goodfellow et al. [11], Jacobian-based Saliency Map Attack (JSMA) of Papernot et al. [30], the C&W attack of Carlini and Wagner [3] and many others [50, 42, 25, 31, 26].

These approaches generally work as follows: given an input sample x and a machine learning model f, they compute an imperceptible noise r. This noise is then added to the original input sample x to get an adversarial sample, $\tilde{x} = x + r$. The objective is that the adversarial sample and original sample must fall into different output classes, i.e, $f(x) \neq f(\tilde{x})$.

DeepFool [25] is one such technique for the generation of adversarial examples. In Figure 2.2, we show one such example where the adversarial noise r (generated using DeepFool) is added on to the original data sample x (MNIST digit 7) to produce adversarial data sample $\tilde{x}$ (MNIST digit 9). In general, DeepFool solves the following problem iteratively:

$$r^* = \arg\min_r \|r\|_2 \text{ s.t. } f(x+r) \neq f(x)$$

In the binary classification setting (i.e. where $rangef = \{-1, 1\}$), it uses a first order approximation of the analytical solution for the linearly-separable case:

$$r_l = -\frac{f(x_l)}{\|\nabla f(x_l)\|_2^2} \nabla f(x_l)$$
$$x_{l+1} = x_l + r_l$$

The process is started by setting $x_0 = x$, and terminates at the lowest index L for which $f(x_L) \neq f(x)$. The total noise is obtained by summing up all the individual noise at each step, $r = \sum_{l=1}^{L} r_l$. This algorithm can be extended to work in the multi-class classification setting. Interested readers can refer to [25] for more details.

2.4 Papernot attack

Papernot et al. [31] introduced a model extraction attack, which we use in Section 5.3 to compare against our model extraction attack (in context of PRADA evasion).

With reference to Algorithm 1, we describe the Papernot attack as follows: It needs initial set of training samples S_0, which consists of small set of disjoint samples which are distributed similarly as secret dataset (in other words, it requires Problem Domain (PD) data as thief dataset) to start with. These samples are labeled by querying it to the secret model g to get initial training set D_0. Adversary's thief model $\tilde{g}$ is then trained on D_0. Now, the attack works in an iterative manner till $i \leq N$. In each iteration, it uses Jacobian-based Dataset Augmentation (JbDA) technique to generate k new synthetic samples. These samples are then labeled by querying them to the secret model g and resultant set is then merged with the previous D_{i-1} to get D_i. The thief model $\tilde{g}$ is then trained on D_i from scratch.

Algorithm 1: Papernot's model extraction attack

Input : Initial set of training samples S_0, Secret model g
Parameters: Budget b, Number of iterations N, λ
Output : Thief model $\tilde{g}$

$D_0 \leftarrow \{(x, g(x)) : x \in S_0\}$
$k \leftarrow b \div N$
$\tilde{g} \leftarrow$ Thief_Model_Train(D_0)
foreach $i \in \{1 \dots N\}$ **do**
  $S_i \leftarrow \{(x + \lambda \cdot sgn(J_F[\tilde{g}(x)])) : x \in S_{i-1}\}$
  $D_i \leftarrow D_{i-1} \cup \{(x, g(x)) : x \in S_i\}$
  $S_i \leftarrow S_i \cup S_{i-1}$
  $\tilde{g} \leftarrow$ Thief_Model_Train(D_i)

The heuristic used by Papernot attack for generating new synthetic samples is based on identifying direction in which the thief model's (or partially extracted secret model's) output is varying, around a set of samples S_{i-1}. These directions are identified with the thief model's Jacobian matrix J_F evaluated on data samples $x \in S_{i-1}$. Each $x \in S_{i-1}$, is modified by adding λ (configurable parameter) times sign of the Jacobian matrix $J_F[\tilde{g}(x)]$ (computed on the output vector of the thief model $\tilde{g}$ at point x). The generation of synthetic samples is identical to

generation of adversarial examples using the Fast Gradient Sign Method (FGSM) [11].

Chapter 3

Threat model

In this chapter, we describe the threat model under which our proposed model extraction framework operates.

3.1 Attack surface

We assume that the adversary does not have direct access to the secret model, rather it can only query it in black-box fashion via an API. We assume *query cost* associated with each query to the secret model. As this query cost can be potentially used by the service provider to provide a defense that limits the number of queries from each end-user, we assume that our model extraction framework works on strict *limited query budget*.

3.2 Capabilities

As the adversary has only black-box access to the secret model via an API, it is capable of selecting only which input query to send to the secret model. Since the API which adversary uses to query the secret model is public, it is fair to assume that the adversary knows the format of input-output and how to interpret it. The output format of API can be of two types – top-1 prediction vector and full probability distribution vector. Top-1 prediction vector means that the highest probability output class of the secret model is returned as a one-hot standard basis vector. The full probability distribution vector means the complete output probability distribution of the secret model is returned as it is. An adversary who assumes access to full probability distribution vector is said to have stronger capability compared to the one who assumes access to only the top-1 prediction vector. In this work, all our results are produced by assuming weaker capability of the adversary, i.e., assuming the output of the API to be top-1 prediction vector only. Hence, a stronger model extraction attack.

As we show in Sub-section 5.2.2.1, even if there is a minor difference between network architectures of the secret model and thief model, it still fetches good quality of the extracted model to the adversary. However, as shown by related line of work (*model reverse-engineering* [27, 46, 8, 49, 15, 14]), it is possible for the adversary to know the precise details of the secret model architecture and it's corresponding hyperparameters. Hence, we report our results using the same architecture for both the secret and the thief model.

Lastly, the adversary does not assume any knowledge about the secret dataset. It makes use of freely available unannotated public data to extract the secret model. Note that the public data is first labeled by the secret model before it can be used to train the thief model.

3.3 Adversary's goal

The goal of the adversary is to get a thief model function $\tilde{g}$ which closely approximates the secret model function g, i.e., $\tilde{g} \approx g$. To achieve this, it uses a subset S of thief dataset X_{thief}, i.e., $S \subsetneq X_{thief}$. As there is a cost associated with each query to the secret model, the adversary would want $|S| \ll |X_{thief}|$. It trains the thief model $\tilde{g}$ on the subset S with its corresponding labels obtained by querying the secret model g. A metric introduced by Tramèr et al. [45], which is used to measure similarity quantitatively between the extracted model $\tilde{g}$ and the secret model g is known as *agreement*.

Definition (Agreement): Two models g and $\tilde{g}$ are said to agree on a input sample x, if the label predicted by both of them on x are same, i.e., $g(x) = \tilde{g}(x)$. Formally, it is defined as:

$$\text{Agreement}(g, \tilde{g}, D) = \frac{1}{|D|} \sum_{(x,.) \in D} \mathbb{1}[g(x) = \tilde{g}(x)]$$

where D denotes a dataset and $\mathbb{1}(.)$ is the indicator function. It simply means fraction of samples from D on which both the secret and thief model predicts the same label. Note that we use test split of secret dataset (or problem domain dataset) as D. We do so only for evaluation and is not used during the model extraction process. The secret model is queried with and the thief model is trained on samples only from the NNPD thief dataset.

Chapter 4

Model extraction framework and PRADA evasion

4.1 Model extraction framework

With reference to Figure 4.1, we describe the flow of our model extraction framework as follows:

1. The adversary selects a subset S_0 of samples from the thief dataset. This selection is done in a uniformly random fashion.

2. In i^{th} iteration (where i varies from 0,1,2,...,N), the adversary queries samples from set S_i and collects true labels[1] corresponding to each sample in the set. Thus, makes $D_i = \{(x, g(x)) : x \in S_i\}$.

3. In i^{th} iteration, the thief model is trained from scratch on all collected D_i's till the current one, i.e., $\bigcup_{t=0}^{i} D_t$.

4. The adversary queries remaining samples of the thief dataset against it's own thief model to get approximate labels[1]. In other words, this step curates an auxiliary dataset $\tilde{D}_i$:

$$\tilde{D}_i = \{(x, \tilde{g}(x)) : x \notin S_1 \cup \cdots \cup S_i\}$$

   These approximate label aids active learning techniques (in the next step) to take a decision whether a thief dataset sample is informative or not. Note that as the thief model is in complete access of the adversary, we get these approximate labels $\tilde{y} = \tilde{g}(x)$ as full softmax probability vectors.

[1]We refer *true label* as the output of the secret model, *approximate label* as the output of thief model and *annotation* as categorical information of the thief dataset sample.

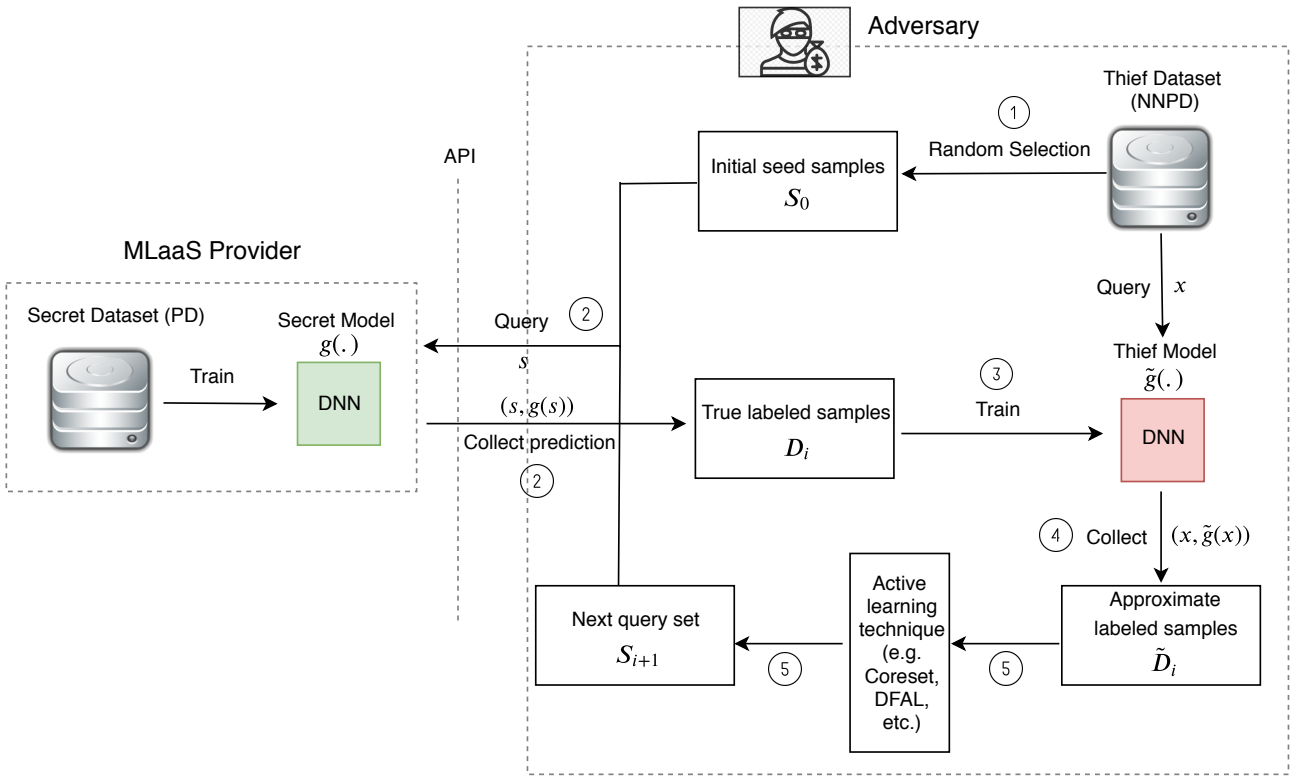


Figure 4.1: Our model extraction framework (explanation of the flow 1-5 is given in Section 4.1)

5. In this step, an active learning subset selection technique is used to choose the most informative k samples S_{i+1} to be queried next, such that $x \in S_{i+1}$ only if $(x, \tilde{y}) \in \tilde{D}_i$.

The steps 2 to 5 goes on in a loop till $i \leq N$. Note that our framework uses NNPD thief dataset (described in introduction) which is freely available public data. The complete procedure is formally described in Algorithm 2. We refer train and validation split of X_{thief} as $X^{\text{valid}}_{\text{thief}}$ and $X^{\text{train}}_{\text{thief}}$ respectively. Training regime followed by the subroutine TRAIN_THIEF_MODEL is described in Sub-section 5.1.3). Details of the active learning techniques used for subset selection in subroutine ACTIVE_LEARNING_TECHNIQUE are described in the following subsections.

4.1.1 Basic active learning subset selection techniques

In each iteration of our model extraction algorithm, the subroutine ACTIVE_LEARNING_TECHNIQUE selects a new set of k thief dataset samples $S_i \subseteq X_{thief}$ to label by querying the secret model g. In general, each subset selection active learning technique takes input as the approximately labeled set $\tilde{D}_i = \{(x_m, \tilde{y}_m)\}$, and return a set S_i. Description of subset selection techniques are as follows:

Algorithm 2: Model extraction using active learning

Input : NNPD thief dataset X_{thief}, Secret model g
Parameters: Budget b, Number of iterations N, Fraction of validation dataset μ, Initial number of samples k_0
Output : Thief model $\tilde{g}$

$S_{\text{valid}} \leftarrow \mu b$ number of random samples from $X^{\text{valid}}_{\text{thief}}$
$D_{\text{valid}} \leftarrow \{(x, g(x)) : x \in S_{\text{valid}}\}$
$S_0 \leftarrow k_0$ number of random samples from $X^{\text{train}}_{\text{thief}}$
$D_0 \leftarrow \{(x, g(x)) : x \in S_0\}$
$k \leftarrow \big((1-\mu)b - k_0\big) \div N$
$\tilde{g} \leftarrow$ TRAIN_THIEF_MODEL(D_0, D_{valid})
foreach $i \in \{1 \dots N\}$ **do**
  $\tilde{D}_i \leftarrow \{(x, \tilde{g}(x)) : x \in X^{\text{train}}_{\text{thief}} \wedge (x, \cdot) \notin D_{i-1}\}$
  $S_i \leftarrow$ ACTIVE_LEARNING_TECHNIQUE$(\tilde{D}_i, D_{i-1}, \tilde{g}, k)$
  $D_i \leftarrow D_{i-1} \cup \{(x, g(x)) : x \in S_i\}$
  $\tilde{g} \leftarrow$ TRAIN_THIEF_MODEL(D_i, D_{valid})

Random technique: In each iteration, this technique simply selects k samples uniformly at random from remaining samples of the thief dataset $X^{\text{train}}_{\text{thief}}$.

Uncertainty technique: It was introduced by Lewis and Gale [23] and is perhaps the simplest active learning technique. It selects data samples which the thief model is least certain about what label to assign. Formally, this technique computes entropy (ω_m) for every data sample $(x_m, \tilde{y}_m) \in \tilde{D}_i$, which is defined as follows:

$$\omega_m = -\sum_j \tilde{y}_{m,j} \log \tilde{y}_{m,j}$$

where j is output class index. In each iteration i, this technique selects k data samples which have highest entropy (ω_m) values (i.e., learning algorithm is most uncertain about which labels to assign) and returns it as set S_i.

Coreset technique: The basic idea in this technique is to construct a core-set of data samples whose label information tell us the labels of other data samples. Core-set construction requires one to construct a set of data samples which can cover the entire dataset. To achieve this we use greedy version of the algorithm proposed by Sener and Savarese [34]. Like uncertainty, this technique also operates in label space (i.e., uses output probability vector of the thief model

$\tilde{g}$). The predicted probability vectors $\tilde{y}_m = \tilde{g}(x_m)$ for samples $(x_m, y_m) \in D_{i-1}$ are considered to be cluster centers. In each iteration, the strategy selects k centers by picking, one at a time, pairs $(x_n, \tilde{y}_n) \in \tilde{D}_i$ such that $\tilde{y}_n$ is the most distant from all existing centers:

$$(x_1^*, \tilde{y}_1^*) = \arg \max_{(x_n, \tilde{y}_n) \in \tilde{D}_i} \min_{(x_m, y_m) \in D_{i-1}} \|\tilde{y}_n - \tilde{y}_m\|_2^2$$
$$(x_2^*, \tilde{y}_2^*) = \arg \max_{(x_n, \tilde{y}_n) \in \tilde{D}_i^1} \min_{(x_m, y_m) \in D_{i-1}^1} \|\tilde{y}_n - \tilde{y}_m\|_2^2$$

where:

$$\tilde{D}_i^1 \leftarrow \tilde{D}_i \setminus \{(x_1^*, \tilde{y}_1^*)\}$$
$$D_{i-1}^1 \leftarrow D_{i-1} \cup \{(x_1^*, f(x_1^*))\}$$

i.e. $(x_1^*, \tilde{y}_1^*)$ is moved to the set of selected centers. This process is repeated to obtain k pairs. The samples $x_1^*, x_2^*, \ldots x_k^*$ corresponding to the chosen pairs are selected.

DFAL technique: The core idea behind this technique is that the data sample x for which we can get it's adversarial counterpart $\tilde{x}$ with very less amount of perturbation are more important samples as they lie closer to the decision boundary. This technique is given by Ducoffe and Precioso [7]. Precisely, it uses Deepfool [25] (explained in Section 2.3) to construct adversarial counter part of each sample $x_m \in \tilde{D}_i$ (denoted by $\tilde{x}_m$) such that $\tilde{g}(x_m) \neq \tilde{g}(\tilde{x}_m)$. Let the absolute difference between x_m and $\tilde{x}_m$ be α_m (i.e. $\alpha_m = \|x_m - \tilde{x}_m\|_2^2$). We compute α_m for each $x_m \in \tilde{D}_i$ and take k samples with lowest perturbation values α_m.

4.1.2 DFAL+Coreset ensemble active learning technique

DFAL, on one hand, has an advantage of picking up very informative samples (i.e., ones which are close to the decision boundary) but on the other hand, it might end up choosing redundant samples (i.e., samples which lie close to only few of the decision boundaries). Similarly, Coreset has the advantage of selecting diverse samples but they might not be that informative (i.e., lie close to any of the decision boundaries).

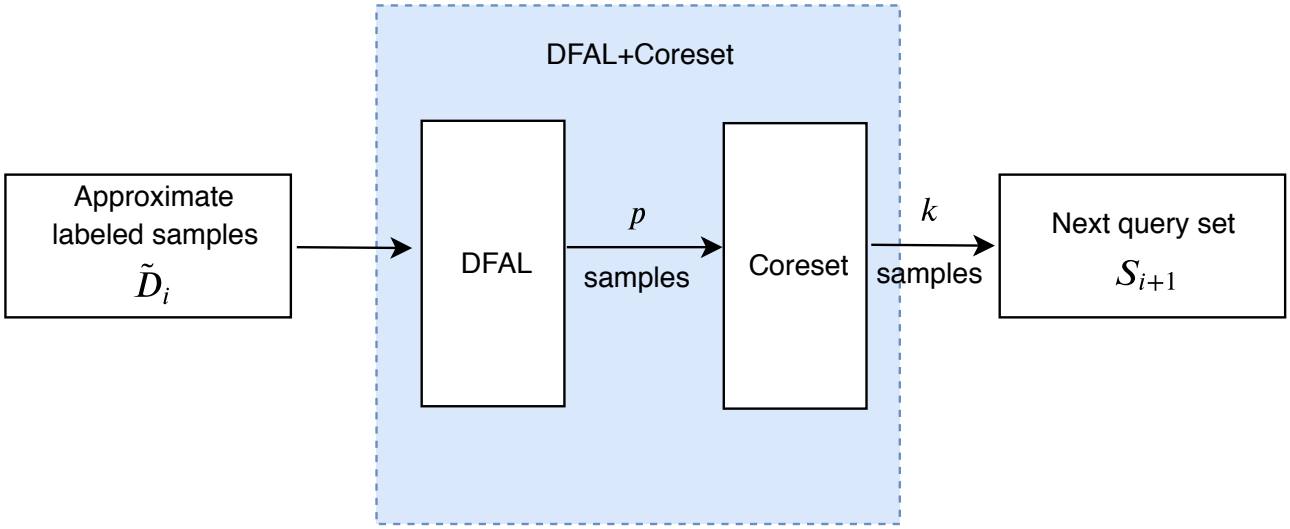


Figure 4.2: Overview of DFAL+Coreset technique

Inspired by this observation, we combined both the techniques in such a way that it could avoid disadvantages of both. We call this ensemble technique as DFAL+Coreset ensemble active learning technique. In this technique, DFAL is first used to select p samples (p is a configurable parameter) and Corset is applied on these p samples to get k samples (which are then queried to the secret model to get labels). In other words, it first selects samples which are close to decision boundary (using DFAL) and then selects samples, among these, which are diverse enough (using Coreset). This gives an advantage of selecting samples which are both close to decision boundary and are diverse enough to cover all the output classes.

To evaluate the efficiency of the new DFAL+Coreset ensemble technique and to check its relative performance in comparison to the basic active learning techniques, we implemented a general active learning Algorithm 3. Note that this algorithm is used to evaluate the DFAL+Coreset technique in the active learning domain, whereas Algorithm 2 is used to evaluate the same technique in model extraction domain. Results for both are presented in next chapter.

As the procedure followed by general active learning algorithm is very similar model extraction algorithm, this algorithm turns out to be similar to Algorithm 2 except for some differences. Instead of using secret model g, in active learning domain an *expert* or *oracle* o labels the samples selected by the active learning techniques. The general active learning algorithm thrives to select a small subset of the original Problem Domain (PD) dataset which when used to train a machine learning model with, will produce a comparable test accuracy as compared to when it is trained with the complete PD dataset. Thus, Algorithm 3 assume access to PD dataset (Unlike in model extraction domain). Note that the objective in model extraction domain is different, hence access to PD dataset there is considered a stronger assumption. Lastly, in this

Algorithm 3: General active learning algorithm

Input : Problem domain unlabeled dataset X_{PD}, Oracle o
Parameters: Budget b, Number of iterations N, Fraction of validation dataset μ, Initial number of samples k_0
Output : Trained model ϕ

$S_{\text{valid}} \leftarrow \mu b$ number of random samples from $X_{\text{PD}}^{\text{valid}}$
$D_{\text{valid}} \leftarrow \{(x, o(x)) : x \in S_{\text{valid}}\}$
$S_0 \leftarrow k_0$ number of random samples from $X_{\text{PD}}^{\text{train}}$
$D_0 \leftarrow \{(x, o(x)) : x \in S_0\}$
$k \leftarrow \big((1-\mu)b - k_0\big) \div N$
$\phi \leftarrow \textsc{Model\_Train}(D_0, D_{\text{valid}})$
foreach $i \in \{1 \dots N\}$ **do**
  $\tilde{D}_i \leftarrow \{(x, \phi(x)) : x \in X_{\text{PD}}^{\text{train}} \wedge (x, \cdot) \notin D_{i-1}\}$
  $S_i \leftarrow \textsc{Active\_Learning\_Technique}(\tilde{D}_i, D_{i-1}, \phi, k)$
  $D_i \leftarrow D_{i-1} \cup \{(x, o(x)) : x \in S_i\}$
  $\phi \leftarrow \textsc{Model\_Train}(D_i, D_{\text{valid}})$

case we evaluate performance of active learning techniques using *test accuracy*, computed on test split of the PD dataset.

Training regime followed by the subroutine Model_Train is described in Sub-section 5.1.3. Subroutine Active_Learning_Technique follows similar description as given in Sub-sections 4.1.1 and 4.1.2.

4.2 PRADA evasion

Juuti et al. [16] proposed PRADA (Protecting against DNN Model Stealing Attacks) which is a generic approach to detect model extraction attacks. It is generic because it does not assume to have information about the secret model or the secret dataset it was trained on. Compared to prior work on adversarial machine learning defenses (e.g., [12], [24]), it is different in its goal as it does not take a decision based on whether individual queries are malicious or not rather it analyses a range of queries and then makes a decision. Thus, this technique tries to know the relation between successive queries rather than focusing on each query.

Algorithm 4 is a simplified version of PRADA's model extraction detection framework. It considers a stream S of samples x queried by an end-user to the secret model g. It calculates minimum distance $d_{min}(x_i)$ of sample x_i with all the previous samples $x_0, x_1, \dots, x_{i-1}$ of the same class. To keep track of the class of each sample, it stores them in G_c, which is different for different class c. All the minimum distances $d_{min}(x_i)$ are stored in a set D. This D is used to model the distribution of distances between queried samples and identify samples that are

unusually close to or far away from previously queried samples.

Algorithm 4: Simplified PRADA algorithm

Input : Secret model g, Stream of samples from end-user S

Parameters: Set for each class G_c, Set of minimum distances D, Detection threshold δ

Output : $IsAdversary$

$G_c \leftarrow \emptyset, D \leftarrow \emptyset, IsAdversary \leftarrow False$
foreach $x \in S$ **do**
  $c \leftarrow g(x)$
  if $G_c == \emptyset$ **then**
    $G_c \cup \{x\}$
  else
    $d \leftarrow \emptyset$
    foreach $y \in G_c$ **do**
      $d \cup \{dist(x, y)\}$
    $d_{min} \leftarrow min(d)$
    $D \cup \{d_{min}\}$
  if $|D| > 100$ **then**
    if $W(D) < \delta$ **then**
      $IsAdversary \leftarrow True$
    else
      $IsAdversary \leftarrow False$

They believe that if the distribution followed by D deviates beyond a threshold δ from normal distribution then the end-user is a potential adversary. To check this, they use Shapiro-Wilk test for *normality test*, which is as follows:

$$W(D) = \frac{(\sum_{i=1}^{n} a_i d_{(i)})^2}{\sum_{i=1}^{n}(d_i - \bar{d})^2}$$

where $D = \{d_i\}_{i=1}^{n}$, and $d_{(i)}$ refers to the i^{th} order statistic of D, and the values of a_i are functions of the i^{th} expected order statistics of i.i.d. normally distributed random variables. When $W(D) < \delta$, PRADA rejects the null hypothesis and claims that an attack has been detected.

We believe that their detection algorithm works on the assumption that the adversary either generates synthetic combinations or perturbations of benign samples, which causes the distribution of D to deviate from the Normal distribution. As our model extraction framework with the DFAL+Coreset technique does not require the generation of synthetic samples, rather it directly selects samples from natural NNPD dataset, we show in Section 5.3 that it evades PRADA's model extraction detection framework.

Chapter 5

Experimental evaluation

5.1 Experimental setup

5.1.1 Datasets

The datasets on which we perform our experiments can be found in Table 5.1. We use MNIST [20] dataset consisting of gray scale images of handwritten digits (0 to 9 digits, 10 classes), Fashion-MNIST [48] dataset consisting of gray scale images of fashion products ranging over 10 different classes, CIFAR-10 [19] dataset consisting of small color images of 10 different categories and GTSRB [41] dataset consisting of color images of german traffic sign boards spanning over 43 different categories. Experiments in active learning domain (Sub-section 5.2.1) are performed using aforementioned datasets only, whereas, experiments performed in model extraction domain (Sub-section 5.2.2 and Section 5.3) uses these datasets only to train the secret models.

We use a subset of the ILSVRC2012-14 dataset [33] as thief dataset in our model extraction experiments (Sub-section 5.2.2 and Section 5.3). It is a Natural Non-Problem Domain (NNPD) dataset which is used to extract functionality of the secret models and train thief model. Details

Table 5.1: Shows details of the PD datasets used in our experiments. # means *number of* and K means 1000.

Dataset	Dimensions	# Train	# Val	# Test	# Classes
MNIST	$28 \times 28 \times 1$	48K	12K	10K	10
F-MNIST	$28 \times 28 \times 1$	48K	12K	10K	10
CIFAR-10	$32 \times 32 \times 3$	40K	10K	10K	10
GTSRB	$32 \times 32 \times 3$	$\sim$ 31K	$\sim$ 8K	$\sim$ 12K	43

Table 5.2: Shows details of the NNPD thief dataset used in model extraction experiments. # means *number of* and K means 1000. Note that ImageNet subset do not have predefined folds, but for reference the fractions used for training and validation have been mentioned.

Thief Dataset	Dimensions	# Train	# Val	# Test	# Classes
ImageNet subset	$64 \times 64 \times 3$	100K	50K	–	–

of the ILSVRC2012-14 dataset used is present in Table 5.2.

5.1.2 DDN architectures

For experiments in active learning domain (Sub-section 5.2.1), we use following architectures: For MNIST, Fashion-MNIST tasks we use simple LeNet architecture [20] and for CIFAR-10, GTSRB tasks we use VGG-16 [40].

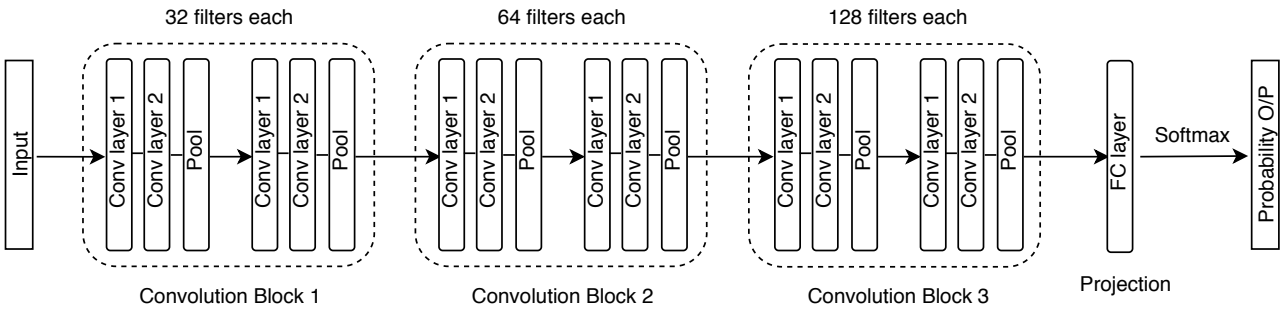


Figure 5.1: Network architecture used in model extraction experiments

For experiments in model extraction domain (Sub-section 5.2.2 and Section 5.3), we use architecture presented in Figure 5.1 for both the secret and thief model. It is a multilayered Convolutional Neural Network (CNN) containing three convolution blocks followed by fully connected and softmax layer to get output probability vector. Each convolution block consist of two repeated units. In each unit, there are two convolution layer (of 3×3 kernel size with stride 1) and one pooling layer (of 2×2 kernel size with stride 2). Each convolution layer output goes through ReLU activation and batch normalization layer. Pooling layer output passes through dropout layer. Convolution layer in the three blocks use 32, 64 and 128 filters respectively.

5.1.3 Training regime

For experiments in active learning domain (Sub-section 5.2.1), we follow following training regime: we run all our experiments with initial batch size or number of samples k_0 as 0.1 times

the total budget (b). The fraction (μ) of problem domain dataset used as validation split is 0.2. For the DFAL+Coreset ensemble technique, we use $p = b$, i.e., first DFAL technique selects b samples and then that goes as an input to Coreset technique to filter k samples out of it. We choose number of iterations $N = 10$ across all our experiments. For training the model we use Adam optimizer [18] with default learning rate of 0.001.

For experiments in model extraction domain (Sub-section 5.2.2 and Section 5.3), we follow following training regime: We use Adam optimizer [18] with default parameter values (learning rate = 0.001) for training the Thief model. It is trained, in each iteration, starting from the same random initialization for at most 1000 epochs with an early stopping criteria (patience of 100 epochs). Batch size used is 150. L_2 regularizer is used for all the model parameters with a loss term multiplier of 0.001. A dropout of 0.1 is used for all the model parameters, for every datasets except CIFAR-10. For CIFAR-10, it is 0.2. Model is evaluated at the end of each epoch and it's F_1 measure is recorded on validation split of the thief dataset (ImageNet). Model with best F_1 value is selected as $\tilde{g}$ in that iteration. We use the value of k_0 as $0.1 \times b$, μ as 0.2. Specific to DFAL+Coreset ensemble technique, we use $p = b$, i.e., DFAL filter outs b samples followed by Coreset which selects k samples.

Training regime used for the thief model of the papernot attack [31] (formally defined in Section 2.4) used in our PRADA experiments (Section 5.3) is as follows: we use a value of $\lambda = 0.1$, as recommended in the paper. Values for total number of iterations (N), initial number of samples per class, total budget (b) for MNIST, Fashion-MNIST, CIFAR-10 are 7, 15, 9.6K respectively and for GTSRB it is 6, 10, 13.76K respectively.

All of our experiments are run on a server with a 24-core Intel Xeon Gold 6150 CPUs and NVIDIA GeForce GTX 1080Ti GPUs.

5.2 Results and analysis of the DFAL+Coreset ensemble technique

5.2.1 In active learning domain

Results of the experiments performed using Algorithm 3 is shown in Table 5.3. Each sub-table indicates results on a specific dataset. In any such sub-table, row corresponds to an active learning technique and column corresponds to a specific query budget. An entry in the table corresponds to test accuracy achieved by the model when a specific active learning technique and budget (b) is used. We report each entry after completion of all $N = 10$ iterations of the algorithm. For comparison, we also report the test accuracy the model can achieve when complete dataset is used.

Table 5.3: Each entry corresponds to test accuracy of the model for a particular active learning technique and budget (b) after completion of all $N = 10$ iterations of the Algorithm 3.

	Test Accuracy(%)				
MNIST	100	500	1000	1500	2000
Random	77.53	95.09	96.28	96.77	97.16
Uncertainty	78.90	95.54	96.90	96.92	97.94
Coreset	81.96	96.43	98.04	98.48	**98.95**
DFAL	84.44	96.64	**98.14**	98.51	98.90
DFAL+Coreset	**86.05**	**96.80**	97.99	**98.57**	98.61
Using full dataset:					98.53

	Test Accuracy(%)				
Fashion-MNIST	100	500	1000	1500	2000
Random	68.92	79.62	80.98	83.69	84.03
Uncertainty	65.72	77.79	84.03	85.21	86.18
Coreset	71.91	78.96	81.85	83.71	84.21
DFAL	69.94	79.18	82.42	84.47	85.53
DFAL+Coreset	**73.36**	**82.36**	**84.56**	**85.73**	**87.18**
Using full dataset:					92.99

	Test Accuracy(%)				
CIFAR-10	1000	5000	10000	15000	20000
Random	41.61	71.74	77.39	79.21	81.45
Uncertainty	41.12	70.90	79.41	81.07	83.20
Coreset	45.12	**73.01**	**80.80**	**83.02**	83.25
DFAL	43.47	72.14	78.65	81.09	84.05
DFAL+Coreset	**45.59**	71.03	79.51	82.13	**84.30**
Using full dataset:					84.09

	Test Accuracy(%)				
GTSRB	100	500	1000	2500	5000
Random	10.14	71.66	87.32	94.15	96.94
Uncertainty	10.59	72.90	91.01	97.05	97.74
Coreset	11.40	70.48	91.81	**97.32**	**97.80**
DFAL	10.83	78.14	90.77	95.15	96.77
DFAL+Coreset	**15.69**	**80.01**	**91.89**	97.11	97.19
Using full dataset:					98.00

Following are the observations we make from the table:

- Out of 20 experiments, DFAL+Coreset ensemble active learning technique won 13 times, which is a clear majority. Followed by Coreset and DFAL which won for 6 and 1 times respectively.
- The ensemble DFAL+Coreset technique turns out to be a better alternative as it improved over DFAL 17 out 20 times. Similarly, it improved over Coreset 13 out of 20 experiments.
- As we increase the budget to be labeled by the oracle o (i.e. going from left to right in the Table 5.3), test accuracy of the model almost always increases.

The above observations indicate the potential DFAL+Coreset ensemble active learning technique bears. It turns out to be a majority winner and improved over the individual DFAL and Coreset techniques in majority of our experiments.

5.2.2 In model extraction domain

Results of all our experiments performed using Algorithm 2 are shown in Table 5.4. Except for random, all other active learning techniques are run in an iterative manner. As the choice of samples in each iteration is not influenced by the thief model ($\tilde{g}$) (in random technique), rather it is chosen in uniformly at random, we chose to run it in one shot. The numbers reported in the table are test agreement (as defined in Section 3.3), recorded at the end of last iteration, between the secret and the thief model on test split of Problem Domain (PD) dataset (defined in Chapter 1). Note that the PD dataset is not used during extraction of the secret model but only to report the test agreement after the secret model is extracted. We show our results on all four datasets and for different query budgets: 10K, 15K, 20K, 25K, 30K (where K = 1000). For comparison, we report the test agreement achieved by thief model when complete thief dataset (120K samples) is used.

Our observation across all the experiments of the Table 5.4 are following:

- The DFAL+Coreset ensemble technique improves over the DFAL technique in 15 (of 20) experiments. Similarly, it improves over the Coreset technique in 13 (of 20) experiments.
- The DFAL+Coreset ensemble active learning technique wins for 13 (of 20) experiments, which is a clear majority. Followed by Coreset and DFAL techniques which wins for 5 and 2 (of 20) experiments respectively.
- As it is evident from the table, when we increase query budget test agreement between the secret and the thief model increases.

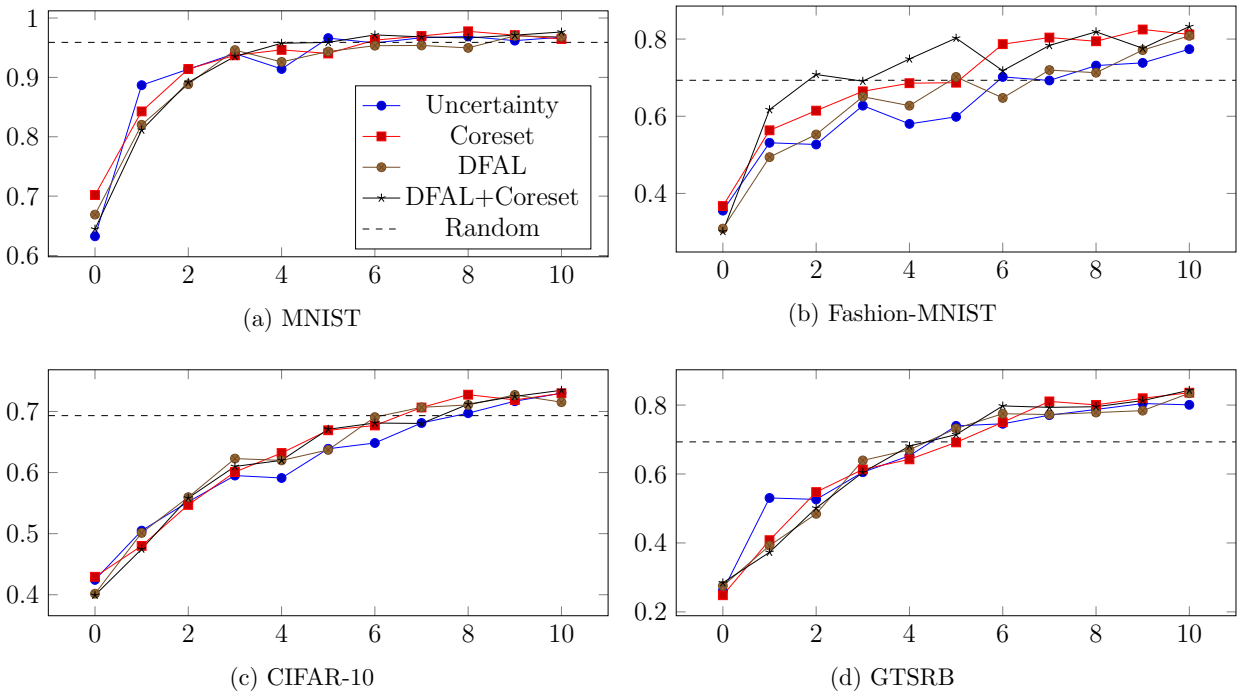
Table 5.4: Each entry corresponds to test agreement between the secret and the thief model after all $N = 10$ iterations of the Algorithm 2 has been completed. Here, K denotes 1000.

	Test Agreement(%)				
MNIST	10K	15K	20K	25K	30K
Random	91.64	95.19	95.90	97.48	97.36
Uncertainty	94.64	97.43	96.77	97.29	97.38
Coreset	**95.80**	95.66	96.47	**97.81**	97.95
DFAL	95.75	95.59	96.84	97.74	97.80
DFAL+Coreset	95.40	**97.64**	**97.65**	97.60	**98.18**
Using the full thief dataset (120K):					98.54
Using uniform noise samples (100K):					20.56

	Test Agreement(%)				
Fashion-MNIST	10K	15K	20K	25K	30K
Random	62.36	67.61	69.32	71.76	71.57
Uncertainty	71.18	72.19	77.39	77.88	82.63
Coreset	71.37	77.03	81.21	79.46	82.90
DFAL	67.61	69.89	80.84	80.28	81.17
DFAL+Coreset	**73.51**	**81.45**	**83.24**	**80.83**	**83.38**
Using the full thief dataset (120K):					84.17
Using uniform noise samples (100K):					17.55

	Test Agreement(%)				
CIFAR-10	10K	15K	20K	25K	30K
Random	63.75	68.93	71.38	75.33	76.82
Uncertainty	63.36	69.45	72.99	74.22	76.75
Coreset	**64.20**	70.95	72.97	74.71	78.26
DFAL	62.49	68.37	71.52	**77.41**	77.00
DFAL+Coreset	61.52	**71.14**	**73.47**	74.23	**78.36**
Using the full thief dataset (120K):					84.99
Using uniform noise samples (100K):					10.62

	Test Agreement(%)				
GTSRB	10K	15K	20K	25K	30K
Random	67.72	77.71	79.49	82.14	83.84
Uncertainty	67.30	73.92	80.07	83.61	85.49
Coreset	70.89	**81.03**	83.59	**85.81**	85.93
DFAL	**72.71**	79.44	83.43	84.41	83.98
DFAL+Coreset	70.79	79.55	**84.29**	85.41	**86.71**
Using the full thief dataset (120K):					93.68
Using uniform noise samples (100K):					45.53



(a) MNIST (b) Fashion-MNIST

(c) CIFAR-10 (d) GTSRB

Figure 5.2: Plots show iterative improvement of test agreement between the secret and the thief model, for all the active learning techniques, after each iteration of the Algorithm 2. Random technique is indicated as a straight line parallel to X-axis.

- Using uniform noise samples (as used by Tramèr et al. [45]) as thief dataset is not a good choice. In our experiments for all four datasets, it gives low test agreement between the secret and the thief model even when we query 100K samples. The reason which we observe is that in all of our experiments (which involves using uniform noise as thief dataset) there are many output class labels which are predicted extremely rarely while some are quite dominant. For example, in case of MNIST dataset, digit 6 is predicted for 96.04% of the time, followed by digit 4 and 5 with 2.74% and 1.21% prediction; rest of the digits were never predicted even once. This issue diminishes if we use NNPD thief datasets such as ImageNet. On an average, by using full thief dataset (i.e., 120K samples of ImageNet) the test agreement improves by 4.85× over full uniform noise samples (100K samples). Moreover, an improvement of 4.70× is retained even when using 30K samples which are selected using the DFAL+Coreset ensemble active learning technique.

Above observations infer (the similar conclusion as concluded by previous Sub-section 5.2.1) that DFAL+Coreset ensemble active learning technique in general shows increased potential in extracting information from the secret model over the individual DFAL and Coreset techniques.

We also show test agreement on each iteration, as the Algorithm 2 progresses, for 20K

budget (on all the active learning techniques and datasets) in Figure 5.2. As evident from the plots, DFAL+Coreset ensemble active learning technique shows similar performance, at the end of each iteration of the algorithm, as other active learning techniques and it improves with each subsequent iterations.

5.2.2.1 Influence of thief model architecture

To check the influence of varying DNN architecture on our model extraction algorithm, we consider following three architectures:

- **Lower complexity (LC) architecture:** This architecture has two convolution blocks. Each block has two repeated units of – two convolution layers followed by a pooling layer. The convolution layers in each block has 32 and 64 filters respectively.
- **Base complexity (BC) architecture:** This architecture has three convolution blocks. Each block has two repeated units of – two convolution layers followed by a pooling layer. The convolution layers in each block has 32, 64 and 128 filters respectively. This is the architecture we use in all our main experiments of model extraction (Sub-section 5.2.2 and Section 5.3)
- **Higher complexity (HC) architecture:** This architecture has four convolution blocks. Each block has two repeated units of – two convolution layers followed by a pooling layer. The convolution layers in each block has 32, 64, 128 and 256 filters respectively.

We consider all possible combination of the above architectures applied to both the secret model and the thief model. We show results of experiments on such combinations in Table 5.5.

As evident from the table, the test agreements along the principal diagonal (i.e., when both the secret and the thief model complexities match) are generally high. These results confirm the findings of [16]. We speculate that the degradation in test agreement from using a less or more complex architecture is due to underfitting or overfitting respectively. A less complex architecture may not have the required complexity to learn a function of a more complex secret model. Similarly, A more complex architecture may readily overfit the constructed dataset using the less complex secret model, leading to poor generalization and test agreement.

Even though test agreements are higher when both the secret and thief model architectures are identical, it is still reasonably high even when there is a mismatch in model complexities. An adversary can use *model reverse-engineering* approaches (as explained in Section 6.3) to recover information about the architecture and hyperparameters used by the secret model. Using this information it can construct a thief model of similar architectural complexity leading to an extracted model of high test agreement.

Table 5.5: The agreement on the secret test set when architectures of different complexity are used as the secret model and thief model. Each row corresponds to a secret model architecture, while each column corresponds to a thief model architecture.

(a) MNIST dataset

	Thief model		
Secret model	LC	BC	HC
Lower Complexity (LC)	**98.73**	98.15	97.63
Base Complexity (BC)	97.21	**98.81**	98.10
Higher Complexity (HC)	96.75	98.05	**98.36**

(b) Fashion-MNIST dataset

	Thief model		
Secret model	LC	BC	HC
Lower Complexity (LC)	**87.15**	80.15	75.26
Base Complexity (BC)	81.50	**84.17**	79.88
Higher Complexity (HC)	79.83	73.35	**84.01**

(c) CIFAR-10 dataset

	Thief model		
Secret model	LC	BC	HC
Lower Complexity (LC)	**78.34**	76.83	74.48
Base Complexity (BC)	80.66	81.57	**81.80**
Higher Complexity (HC)	74.34	**79.17**	78.82

(d) GTSRB dataset

	Thief model		
Secret model	LC	BC	HC
Lower Complexity (LC)	**95.02**	92.30	86.88
Base Complexity (BC)	90.08	**91.42**	91.28
Higher Complexity (HC)	80.95	**86.50**	84.69

5.2.2.2 The convergence of thief model training

We observe in our experiments that at the end of each iteration of the model extraction algorithm, the train loss becomes close to zero which indicates that the labels of the samples picked by active learning strategies do not change as the algorithm progresses.

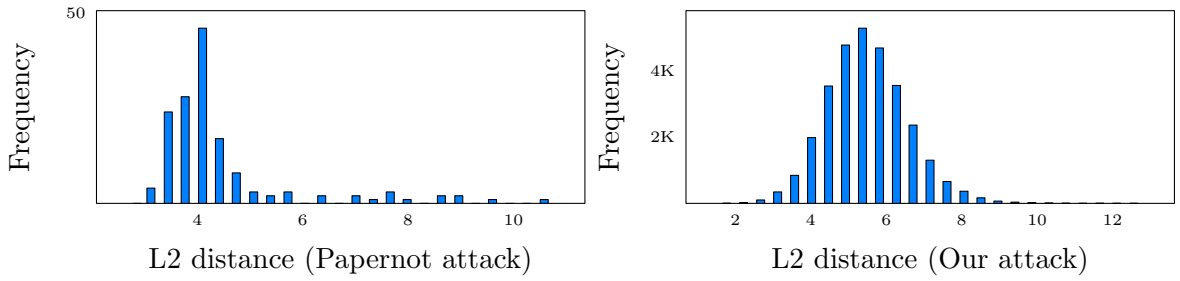
5.3 Results and analysis on PRADA evasion

We run PRADA against our model extraction attack with the DFAL+Coreset ensemble active learning technique and the Papernot attack [31] (as explained in Section 2.4). We run it, against both the attacks, for all four datasets. Results are plotted as histograms of D (explained in Section 4.2) and are shown in Figure 5.3. For Papernot attack, D is plotted only for samples up to the point of detection.
Following are the observations we make:

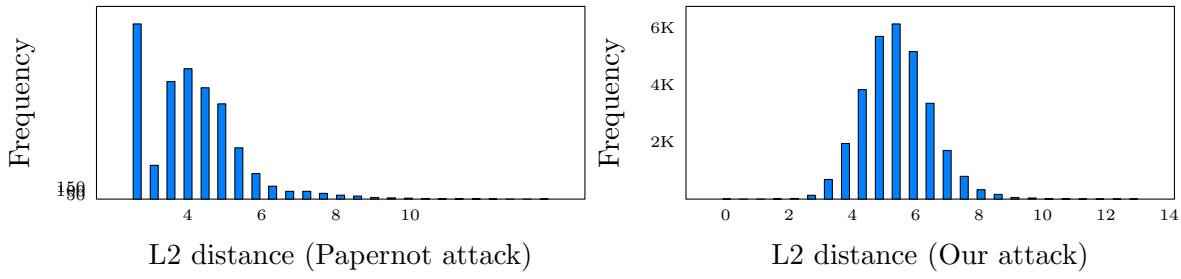
- For MNIST, Fashion-MNIST, CIFAR-10 and GTSRB tasks: Papernot attack is detected by PRADA in 210, 491, 380 and 710 queries respectively to the secret model. Thus, we see its plots deviated from normal distribution.

- For all the aforementioned tasks, our attack with the DFAL+Coreset ensemble technique is not detected even when using a query budget of $30,000$, i.e., even when we fire $30,000$ queries to the secret model (as opposed to Papernot attack where query budget assigned is approximately $10,000$). Thus, all of our attack's plots roughly follow normal distribution.

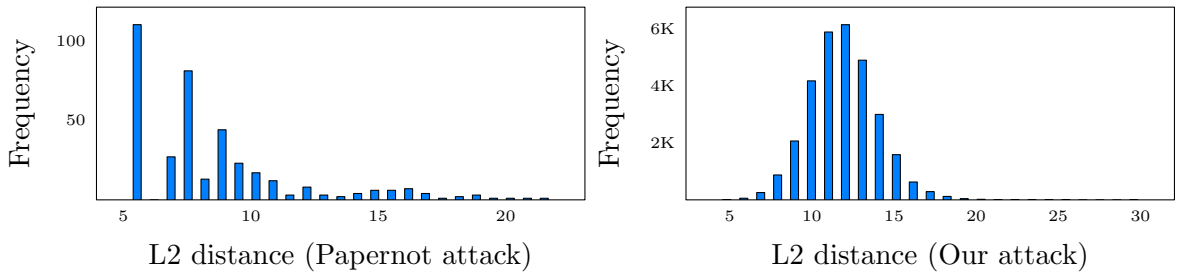
We believe that Papernot's attack gets detected due to the reason that their attack crafts adversarial examples of a small set of original samples (that are then queried to the secret model), which leads D to deviate from normal distribution. On the contrary, as our attack with the DFAL+Coreset ensemble technique selects samples from only Natural NNPD data (and does not craft any synthetic samples), we are able to evade model extraction detection by PRADA.



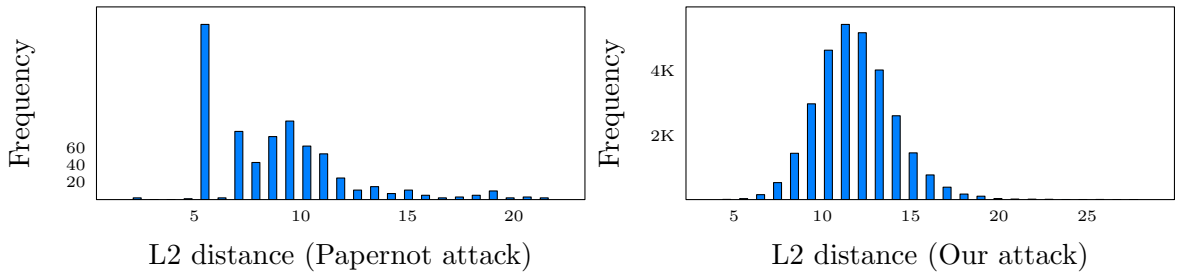
(a) MNIST



(b) Fashion-MNIST



(c) CIFAR-10



(d) GTSRB

Figure 5.3: Plots demonstrate that unlike papernot attack, our attack go undetected (i.e. follows normal distribution) against PRADA model extraction detection framework.

Chapter 6

Related work

In this chapter, we categorize and discuss about the related work in three broad domains – model extraction, active learning and model reverse-engineering.

6.1 Model extraction

We further categorize prior work on model extraction into attacks and defenses depending on their applicability.

6.1.1 Attacks

Tramèr et al. [45] was the first work, which introduced the notion of model extraction. Their work is very similar to ours as they also perform model extraction in strict query budget constraint. They proposed several attacks to extract simple ML models such as: *equation solving attack* for one-layer logistic regression model, *path finding attack* for decision trees. Both the attacks are very efficient and requires only few queries to perform model extraction but are limited to only simple ML models. They proposed a model extraction method targeting shallow feedforward neural network. However, as we showed in Sub-section 5.2.2, their approach of using random uniform noise as thief dataset does not extend to deeper networks.

Sethi and Kantardzic [35] proposed a general framework to evade security systems with ML-based core, e.g., a CAPTCHA system that uses mouse speed and click time features to detect benign users from bots. They call their approach as Seed-Explore-Exploit Framework, where they use model extraction to help in the generation of adversarial examples which can evade the secret model. Their proposed framework has three phases: in *seed phase*, they start with one legitimate and one malicious class sample and then in the *exploration phase* they use Gram-Schmidt process to generate orthogonal samples, near the mid-point of two randomly selected samples of opposite classes from seed set. At the end of *exploration phase*, the generated

dataset is used to train a thief ML model. This model is then used in exploitation phase to craft adversarial samples to evade detection by the security system.

Chandrasekaran et al. [4] draws a parallel between active learning and model extraction. They show that the process of active learning and model extraction is quite similar. Based on this observation, they propose an attack which uses Query Synthesis (QS) active learning to extract ML models such as decision trees. QS active learning is used to generate queries de novo, which are independent of the secret dataset distribution. They implement two QS active learning techniques and leverage them to extract binary classifiers (d-dimensional halfspaces). In contract to their approach, we use pool-based active learning for model extraction.

Shi et al. [37] considers deep learning models as thief models to extract functionality of basic ML models. In particular, they show that their extraction approach works on basic ML models such as: SVM and naïve bayes secret models. Both of these models are trained to perform text classification. Thus, they show that deep learning can reliably infer functionality of basic ML models. They also show that the reverse is not true, i.e., SVM and naïve bayes models cannot be used as a thief model to extraction functionality of deep learning classifiers.

Shi et al. [38] leverages active learning in conjunction with problem domain data to perform model extraction on shallow feed-forward neural network. In the follow-up work by the same authors [39], they propose a model extraction attack that uses Generative Adversarial Network (GAN) trained on small subset of problem domain dataset. They show that this trained GAN can be used to generate informative samples to query the secret model with. In both of these works, they perform their experiments on classification tasks in text domain. Also, both the works, uses extracted model to launch evasion attack (i.e. to generate adversarial samples using the extracted model which can fool the secret model) and causative attack (i.e. to send mislabeled samples, generated using the extracted model, to the secret model which uses them as a feedback to train itself. Thus, degrading the quality of the secret model).

6.1.2 Defenses

Lee et al. [22] proposed a defense which involves applying a perturbation noise to the output softmax probability vector of the secret model. They empirically show that this defense would lead the adversary to make much more queries before it train a thief model with comparable performance. However, as our model extraction attack operates under a stricter Top-1 prediction constraint, such a defense would still leave the secret model vulnerable against our attack.

Quiring et al. [32] shows that the attacks and defenses developed by two separate research communities of model extraction and digital watermarking are very similar. To this end, they

show that defenses against digital watermarking attack can be used to defend against model extraction. This defense assumes secret model to be a decision tree only and does not apply on DNNs.

Hanzlik et al. [13] designed MLCapsule, a framework which can be used to provide guarded offline deployment of MLaaS models. They leveraged a hardware security feature provided by Intel known as Intel SGX for its implementation. This on one hand, allows MLaaS provider to serve their ML models in an offline manner with same security guarantees which are possible with deployment in a cloud platforms and on the other hand, it gives additional benefits to the end-users who do not trust to send their confidential data to the service provider for prediction. MLCapsule uses PRADA [16] to provide a defense against model extraction attacks. As we show in Section 5.3, our model extraction attack evades detection by PRADA and hence, it can successfully extract MLaaS models deployed using MLCapsule.

6.2 Active learning

There are various active learning techniques which are applicable to basic machine learning models such as naïve Bayes and SVMs. For more details on it, we refer interested reader to a survey work on active learning by Settles [36].

Active learning techniques which are proposed specifically for deep neural networks include the following:

Sener and Savarese [34] proposed an active learning technique based on core-set selection, i.e., choosing a set of data samples such that a model learned over the selected subset is competitive for the remaining data samples of the complete training dataset. They theoretically show that the problem of core-set selection can be approximated to solving the K-center problem [47]. While solution to K-center problem proved to be a good initialization point, they further improved their solution by formulating and solving a mixed integer program which ensured that the number of outliers does not exceed a threshold. They used traditional active learning techniques such as uncertainty [23] as baseline and showed significant improvement over them when training deep Convolutional Neural Networks (CNNs).

Ducoffe and Precioso [7] proposed a margin-based active learning technique which selects data samples lying close to decision boundary of the CNNs. Their technique works on the assumption that distance of a data sample from the decision boundary of a CNN can be approximated by calculating minimum perturbation required to get an adversarial counter-part of the data sample. In other words, they exploit information provided by adversarial data sample to estimate approximate distance to the decision boundary. In particular, they use DeepFool [25] for generation of adversarial data samples. They empirically show that their

technique outperforms classical active learning technique such as uncertainty [23] while still being competitive to that of [34] for image classification tasks on CNNs.

6.3 Model reverse-engineering

As we show in Sub-section 5.2.2.1, the agreement between the secret model and the thief model is comparable even when their network architectures do not match. However, it improved when they match. Hence, it is in best interest of the adversary to infer the architecture details of the secret model which is shown to be possible by following approaches in model reverse-engineering literature:

Oh et al. [27] proposed a method to infer confidential attributes such as the number of convolution layers, filter size, value of dropout, activation function, batch size, optimization algorithm and training dataset of the secret model from sequence of input-output queries. They train a meta-model to do this. Input to the meta-model is softmax output probability vector of the secret model and it predicts, with statistically significant confidence, confidential attributes of the secret model as an output. To curate a dataset for training the meta-model on, they first randomly generate and train neural networks of varying architectural complexities and then queries them to get the input-output pairs.

Wang and Gong [46] proposed *hyperparameter stealing attacks* which are applicable on variety of popular machine learning models such as ridge regression, logistic regression, support vector machine and neural network. Their proposed framework is based on the observation that the model parameters learnt by an ML algorithm are often minima of the corresponding objective function. They evaluate effectiveness of their method both theoretically and empirically.

Yan et al. [49] proposed a cache-based side channel attack to extract DNN secret model architectures on general purpose processors. Their method is based on the observation that the secret DNN model's predictions relies heavily on tiled GEMM (Generalized Matrix Multiply). Moreover, the secret DNN model's architecture parameters determines the number of GEMM calls and some attributes of the GEMM function. As such a information is present in the cache, it can be leaked through cache side-channel attacks.

Duddu et al. [8] proposed a model reverse-engineering attack which uses timing side channel to predict secret model architecture and hyperparameters, i.e., it uses execution time of the forward pass of the secret model, averaged over multiple queries, to infer confidential attributes of the secret model. The averaged execution time is given as an input to a pre-trained regressor which predicts the depth of the secret model, which is then used as an input to a reinforcement learning algorithm to predict an optimal architecture which has very close test accuracy as the

secret model.

Hu et al. [15] proposed a method which uses off-chip memory address traces and PCIe events to predict the secret model architecture details. They use off-chip memory address traces and PCIe events information to first predict kernel features such as read and write data volume of memory requests, which is then used to construct layer topology and predict the secret model architecture.

Chapter 7

Conclusion and Future work

In the domain of model extraction, researchers have used active learning techniques to reduce the number of queries required to extract the secret model. However, as it is evident from the experiments, no one active learning technique is well-suited for different datasets and under different query budget constraints. Given the plethora of active learning techniques at the adversary's disposal and the black-box nature of the model under attack, the choice of the technique to be used is difficult but integral: the chosen technique is a strong determinant of the quality of the extracted model. To this end, we introduced a new ensemble active learning technique, *DFAL+Coreset*. Its efficacy is demonstrated by comparing and contrasting with existing active learning techniques. We have also demonstrated that the models extracted using the DFAL+Coreset ensemble technique exhibit more consistent performance than the existing active learning techniques, with it consistently producing extracted models that have a higher agreement with the confidential MLaaS model. Finally, we have also shown that the model extraction attack using the DFAL+Coreset ensemble active learning technique is not detected by the state-of-the-art model extraction detection method, PRADA.

Directions for Future Research. Our proposed DFAL+Coreset ensemble active learning technique shows increased potential in extracting information from the secret model over the individual DFAL and Coreset techniques. However, in a few of our experiments either Coreset or DFAL won over the combined technique. Next step could be to investigate the reason behind it and extend it in such a way that it could take out advantages of both the individual techniques in all the cases.

As we have tried one good combination of existing active learning techniques, the DFAL+Coreset ensemble technique, the next step could be to experiment with more such combinations.

Bibliography

[1] Moez Baccouche, Franck Mamalet, Christian Wolf, Christophe Garcia, and Atilla Baskurt. Sequential deep learning for human action recognition. In Albert Ali Salah and Bruno Lepri, editors, *Human Behavior Understanding*, pages 29–39, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg. ISBN 978-3-642-25446-8. 1

[2] Osbert Bastani, Carolyn Kim, and Hamsa Bastani. Interpretability via model extraction. *2017 Workshop on Fairness, Accountability, and Transparency in Machine Learning*, 2017. 2

[3] Nicholas Carlini and David Wagner. Towards evaluating the robustness of neural networks. In *2017 IEEE Symposium on Security and Privacy (S&P)*, pages 39–57. IEEE, 2017. 11

[4] Varun Chandrasekaran, Kamalika Chaudhuri, Irene Giacomelli, Somesh Jha, and Songbai Yan. Exploring connections between active learning and model extraction. *CoRR*, abs/1811.02054, 2018. URL http://arxiv.org/abs/1811.02054. 36

[5] D. Ciregan, U. Meier, and J. Schmidhuber. Multi-column deep neural networks for image classification. In *2012 IEEE Conference on Computer Vision and Pattern Recognition*, pages 3642–3649, June 2012. doi: 10.1109/CVPR.2012.6248110. 1

[6] 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 *2018 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8, July 2018. 1, 3, 4

[7] Melanie Ducoffe and Frédéric Precioso. Adversarial active learning for deep networks: a margin based approach. *CoRR*, abs/1802.09841, 2018. URL http://arxiv.org/abs/1802.09841. 5, 19, 37

[8] Vasisht Duddu, Debasis Samanta, D. Vijay Rao, and Valentina E. Balas. Stealing neural networks via timing side channels. *CoRR*, abs/1812.11720, 2018. URL http://arxiv.org/abs/1812.11720. 15, 38

[9] Xiaoyue Feng, Hao Zhang, Yijie Ren, Penghui Shang, Yi Zhu, Yanchun Liang, Renchu Guan, and Dong Xu. The deep learning–based recommender system "pubmender" for choosing a biomedical publication venue: Development and validation study. *J Med Internet Res*, 21(5):e12957, May 2019. ISSN 1438-8871. doi: 10.2196/12957. URL http://www.jmir.org/2019/5/e12957/. 1

[10] Matt Fredrikson, Somesh Jha, and Thomas Ristenpart. Model inversion attacks that exploit confidence information and basic countermeasures. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, pages 1322–1333. ACM, 2015. 1

[11] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*, 2014. 11, 13

[12] Kathrin Grosse, Praveen Manoharan, Nicolas Papernot, Michael Backes, and Patrick D. McDaniel. On the (statistical) detection of adversarial examples. *CoRR*, abs/1702.06280, 2017. URL http://arxiv.org/abs/1702.06280. 21

[13] Lucjan Hanzlik, Yang Zhang, Kathrin Grosse, Ahmed Salem, Max Augustin, Michael Backes, and Mario Fritz. MLCapsule: Guarded offline deployment of machine learning as a service. *CoRR*, abs/1808.00590, 2018. URL http://arxiv.org/abs/1808.00590. 37

[14] 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. *arXiv preprint arXiv:1810.03487*, 2018. 15

[15] Xing Hu, Ling Liang, Lei Deng, Shuangchen Li, Xinfeng Xie, Yu Ji, Yufei Ding, Chang Liu, Timothy Sherwood, and Yuan Xie. Neural network model extraction attacks in edge devices by hearing architectural hints. *arXiv preprint arXiv:1903.03916*, 2019. 15, 39

[16] Mika Juuti, Sebastian Szyller, Alexey Dmitrenko, Samuel Marchal, and N. Asokan. PRADA: Protecting against DNN model stealing attacks. In *2019 IEEE European Symposium on Security and Privacy (EuroS&P)*, 2019. 5, 21, 31, 37

[17] Yoon Kim. Convolutional neural networks for sentence classification. In *EMNLP*, 2014. 1

[18] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *ICLR*, 2015. URL https://arxiv.org/abs/1412.6980. 26

[19] Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images. Technical report, Citeseer, 2009. 5, 24

[20] Yann LeCun, Léon Bottou, Yoshua Bengio, Patrick Haffner, et al. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998. 5, 24, 25

[21] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553):436, 2015. 9

[22] Taesung Lee, Benjamin Edwards, Ian Molloy, and Dong Su. Defending against model stealing attacks using deceptive perturbations. *CoRR*, abs/1806.00054, 2018. URL http://arxiv.org/abs/1806.00054. 36

[23] David D. Lewis and William A. Gale. A sequential algorithm for training text classifiers. In *Proceedings of the 17th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 3–12, 1994. URL http://dl.acm.org/citation.cfm?id=188490.188495. 10, 18, 37, 38

[24] Dongyu Meng and Hao Chen. Magnet: a two-pronged defense against adversarial examples. *CoRR*, abs/1705.09064, 2017. URL http://arxiv.org/abs/1705.09064. 21

[25] Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, and Pascal Frossard. DeepFool: a simple and accurate method to fool deep neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2574–2582, 2016. vii, 10, 11, 12, 19, 37

[26] Anh Nguyen, Jason Yosinski, and Jeff Clune. Deep neural networks are easily fooled: High confidence predictions for unrecognizable images. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 427–436, 2015. 11

[27] Seong Joon Oh, Max Augustin, Mario Fritz, and Bernt Schiele. Towards reverse-engineering black-box neural networks. In *ICLR*, 2018. URL https://openreview.net/forum?id=BydjJte0-. 15, 38

[28] Tribhuvanesh Orekondy, Bernt Schiele, and Mario Fritz. Knockoff nets: Stealing functionality of black-box models. In *CVPR*, 2019. 1, 3, 4

[29] Soham Pal, Yash Gupta, Aditya Shukla, Aditya Kanade, Shirish K. Shevade, and Vinod Ganapathy. ActiveThief: Model extraction using active learning and unannotated public data. In *AAAI*, 2020. 6

[30] Nicolas Papernot, Patrick McDaniel, Somesh Jha, Matt Fredrikson, Z Berkay Celik, and Ananthram Swami. The limitations of deep learning in adversarial settings. In *2016 IEEE European Symposium on Security and Privacy (EuroS&P)*, pages 372–387. IEEE, 2016. 11

[31] Nicolas Papernot, Patrick D. McDaniel, Ian J. Goodfellow, Somesh Jha, Z. Berkay Celik, and Ananthram Swami. Practical black-box attacks against machine learning. In *AsiaCCS*, 2017. 1, 3, 8, 11, 12, 26, 33

[32] Erwin Quiring, Daniel Arp, and Konrad Rieck. Fraternal twins: Unifying attacks on machine learning and digital watermarking. *arXiv preprint arXiv:1703.05561*, 2017. 36

[33] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015. 3, 4, 24

[34] Ozan Sener and Silvio Savarese. Active learning for convolutional neural networks: A core-set approach. In *ICLR*, 2018. URL https://openreview.net/forum?id=H1aIuk-RW. 5, 18, 37, 38

[35] Tegjyot Singh Sethi and Mehmed M. Kantardzic. Data driven exploratory attacks on black box classifiers in adversarial domains. *Neurocomputing*, 289:129–143, 2018. 35

[36] Burr Settles. Active learning literature survey. Technical report, University of Wisconsin-Madison Department of Computer Sciences, 2009. 4, 9, 37

[37] Yi Shi, Yalin E. Sagduyu, and Alexander Grushin. How to steal a machine learning classifier with deep learning. *2017 IEEE International Symposium on Technologies for Homeland Security (HST)*, pages 1–5, 2017. 36

[38] Yi Shi, Yalin E. Sagduyu, Kemal Davaslioglu, and Jason H. Li. Active deep learning attacks under strict rate limitations for online API calls. In *2018 IEEE International*

Symposium on Technologies for Homeland Security (HST), pages 1–6, Oct 2018. doi: 10.1109/THS.2018.8574124. 36

[39] Yi Shi, Yalin E. Sagduyu, Kemal Davaslioglu, and Jason H. Li. Generative adversarial networks for black-box API attacks with limited training data. In *2018 IEEE International Symposium on Signal Processing and Information Technology (ISSPIT)*, pages 453–458, Dec 2018. doi: 10.1109/ISSPIT.2018.8642683. 36

[40] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014. 25

[41] Johannes Stallkamp, Marc Schlipsing, Jan Salmen, and Christian Igel. Man vs. computer: Benchmarking machine learning algorithms for traffic sign recognition. *Neural Networks*, 32:323–332, 2012. 5, 24

[42] Jiawei Su, Danilo Vasconcellos Vargas, and Kouichi Sakurai. One pixel attack for fooling deep neural networks. *IEEE Transactions on Evolutionary Computation*, 2019. 11

[43] Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. Sequence to sequence learning with neural networks. *CoRR*, abs/1409.3215, 2014. URL http://arxiv.org/abs/1409.3215. 1

[44] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. In *International Conference on Learning Representations*, 2014. URL http://arxiv.org/abs/1312.6199. 11

[45] Florian Tramèr, Fan Zhang, Ari Juels, Michael K. Reiter, and Thomas Ristenpart. Stealing machine learning models via prediction APIs. In *USENIX Security Symposium*, 2016. 1, 2, 3, 5, 15, 30, 35

[46] Binghui Wang and Neil Zhenqiang Gong. Stealing hyperparameters in machine learning. *2018 IEEE Symposium on Security and Privacy (S&P)*, pages 36–52, 2018. 15, 38

[47] Gert W. Wolf. Facility location: Concepts, models, algorithms and case studies. series: Contributions to management science. *Int. J. Geogr. Inf. Sci.*, 25(2):331–333, February 2011. ISSN 1365-8816. doi: 10.1080/13658816.2010.528422. URL https://doi.org/10.1080/13658816.2010.528422. 37

[48] Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-MNIST: a novel image dataset for benchmarking machine learning algorithms. *CoRR*, abs/1708.07747, 2017. URL http://arxiv.org/abs/1708.07747. 5, 24

[49] Mengjia Yan, Christopher Fletcher, and Josep Torrellas. Cache telepathy: Leveraging shared resource attacks to learn DNN architectures. *arXiv preprint arXiv:1808.04761*, 2018. 15, 38

[50] Xiaoyong Yuan, Pan He, Qile Zhu, Rajendra Rana Bhat, and Xiaolin Li. Adversarial examples: Attacks and defenses for deep learning. *IEEE Transactions on Neural Networks and Learning Systems*, 2019. 11

[51] B. Zhao, H. Lu, S. Chen, J. Liu, and D. Wu. Convolutional neural networks for time series classification. *Journal of Systems Engineering and Electronics*, 28(1):162–169, Feb 2017. ISSN 1004-4132. doi: 10.21629/JSEE.2017.01.18. 1