Protecting Model Confidentiality for Machine Learning as a Service

by

Jean-Baptiste Truong

A Thesis

Submitted to the Faculty

of the

WORCESTER POLYTECHNIC INSTITUTE

In partial fulfillment of the requirements for the

Degree of Master of Science

in

Computer Science

May 2021

APPROVED:

_____

Professor Robert J. Walls, Major Thesis Advisor

_____

Professor Tian Guo, Thesis Reader

_____

Professor Craig Wills, Head of Department

Abstract

Current model extraction attacks assume that the adversary has access to a surrogate dataset with characteristics similar to the proprietary data used to train the victim model. This requirement precludes the use of existing model extraction techniques on valuable models, such as those trained on rare or hard to acquire datasets. In contrast, we propose data-free model extraction methods that do not require a surrogate dataset. Our approach adapts techniques from the area of data-free knowledge transfer for model extraction. As part of our study, we identify that the choice of loss is critical to ensuring that the extracted model is an accurate replica of the victim model. Furthermore, we address difficulties arising from the adversary's limited access to the victim model in a black-box setting. For example, we recover the model's logits from its probability predictions to approximate gradients. We find that the proposed data-free model extraction approach achieves high-accuracy with reasonable query complexity – $0.99\times$ and $0.92\times$ the victim model accuracy on SVHN and CIFAR-10 datasets given 2M and 20M queries respectively.

Furthermore, this study identifies and proposes techniques to alleviate two key bottlenecks to executing deep neural networks in trusted execution environments (TEEs): page thrashing during the execution of convolutional layers and the decryption of large weight matrices in fully-connected layers. For the former, we propose a novel partitioning scheme, y-plane partitioning, designed to (i) provide consistent execution time when the layer output is large compared to the TEE secure memory; and (ii) significantly reduce the memory footprint of convolutional layers. For the latter, we leverage quantization and compression. In our evaluation, the proposed optimizations incurred latency overheads ranging from $1.09\times$ to $2\times$ baseline for a wide range of TEE sizes; in contrast, an

unmodified implementation incurred latencies of up to 26X when running inside of the TEE.

Acknowledgements

I would like to express my gratitude to my advisors Robert J. Walls and Tian Guo for their tremendous support throughout my Master's studies, and for training me in academic research. Furthermore, I would like to thank Nicolas Papernot for his precious advice and the amazing collaboration these past months. Thanks also to Pratyush Maini, with who it has been a great pleasure to work. Many thanks to Paul J. Buono for teaching me how to play the piano, it's the main reason for my mental sanity during these months of isolation. Infinitely many thanks to all my loved ones, who always supported me during my studies and in my projects. Finally, I want to thank my grandparents, who went through so much to seek a better life. I owe them eternally for the chance they gave me and for where I am today.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Machine learning (ML) and deep learning, in particular, often require large amounts of training data to achieve high performance on a particular task [SGM19]. Curating such data necessitates significant time and monetary investment [HNP09, DDS+09]. Thus, the resulting ML model becomes valuable intellectual property, especially when considering the computing resources and human expertise required [BMR+20, DBK+20]. Often to monetize these models, companies make them available as a service via APIs over the web (MLaaS). These models are also deployed to end-user devices, making their predictions directly accessible to customers. However, the exposure of the model's predictions represents a significant risk as an adversary can leverage this information to steal the model's knowledge [LM05, TZJ+16, CCG+19, PGS+19, OSF19, CSBB+18, MSDH18, JCB+20]. The threat of such model extraction attacks is two-fold: adversaries may use the stolen model for monetary gains or as a reconnaissance step to mount further attacks [PMG+17a, SZB+20, PMG+17b, BMR+20, DBK+20].

Furthermore, deep learning model owners often rely on the others' hardware, such as cloud providers or end-users, for model execution. A malicious hardware owner could directly extract the model from memory. To address this issue, recent works have ex-

plored the use of *trusted execution environments (TEEs)*, i.e., isolated environments which provide a set of security features that allow running verified code safely on untrusted hardware [LLP+19, KKR+20]. While TEEs provide a natural foundation for sensitive computations, their severe memory constraints have important performance implications. In the context of deep learning, where redundant access to large memory areas is frequent, relying solely on existing TEE paging mechanisms results in prohibitively high overheads—upwards of 26X increases in model latency.

In this work, we study deep learning model extraction in the context of MLaaS. We consider two orthogonal avenues of attack. First, as a malicious user: we describe a model extraction attack, which we call *data-free model extraction*, that aims at stealing the remote machine learning model by repeatedly querying the prediciton API. More specificly, the presented attack does not require any prior data nor knowledge about the training data distribution (see Chapter 2).

Second, we describe how a model owner can leverage a TEE to run their deep learning model confidentially on a remote, untrusted machine. We call this property *confidential deep learning*. While the security guaranties for the model are handled by the TEE, a significant overhead needs to be mitigated to efficiently run deep learning inference. This work, thus, describes two major performance bottlenecks—namely, in convolutional and fully-connected layers—and propose mitigations to enable efficient deep learning inference inside a TEE (see Chapter 3).

Chapter 2

Data-Free Model Extraction

While model extraction attacks are in many ways similar to model distillation, it differs in that the victim's proprietary training set is not accessible to the adversary. To stage a model extraction attack, the adversary typically queries the victim using samples from a surrogate dataset with semantic or distributional similarity to the original training set [OSF19]. In the classification setting, the victim's response may be limited to the most-likely label [CCG+18] or include confidence values for different class labels [JCB+20]. The number of queries—i.e., the *query complexity*—is also an important consideration for the adversary. The greater the query complexity, the higher the cost of the attack—unless the victim model is available offline (e.g., deployed on-device).

In this chapter, we first demonstrate that the success of current established practices for model extraction, which often take the form of distillation, depends on the *closeness* of the surrogate distribution to the victim's proprietary training distribution (see Chapter 2.2). This finding has important implications for the practicality of existing model extraction techniques.

To remedy this, we propose techniques for *data-free model extraction* (DFME). In short, we demonstrate the feasibility of extracting ML models without *any* knowledge of

the distribution of the proprietary training data. In practice, gathering a surrogate dataset for the purpose of model extraction can be a very expensive process, both in terms of the time and money required to curate it. In particular, the most valuable models are often those for which it is most challenging to curate an appropriate surrogate dataset, i.e., when the victim model's value arises from its proprietary dataset. Our work builds on recent advances in data-free knowledge distillation, which involve a generative model to synthesize queries that maximize disagreement between the student and teacher models [MS19, FSS$^+$19]. Here, the teacher is the victim model whereas the student is the stolen extracted model. We innovate on two fronts: the choice of loss to quantify student-teacher disagreement and an approach for training the generator without the ability to backpropagate through the teacher to compute its gradients (because we only have black-box access to the victim/teacher predictions in our setting). We observe that it is essential to ensure the stability of the loss computed, and find that the ℓ_1 norm loss is particularly conducive to data-free model extraction. We also demonstrate that using inexpensive gradient approximation (based on the victim model's outputs) is sufficient to train a generative model that produces queries relevant to distill the knowledge of a victim to a student model. In summary, our main contributions are [1]:

- We demonstrate in Chapter 2.2 that successful distillation-based model extraction attacks require the adversary to sample queries from a surrogate dataset whose distribution is *close* to the victim training data.
- In Chapter 2.4, we propose data-free model extraction (DFME) to extract ML models without knowledge of private training data, and only using the victim's black-box predictions. As a by-product of DFME needing to approximate gradients of the victim, this leads us to present a method for recovering per-example logits out

[1]The majority of this work was made in equal contribution with Pratyush Maini. A few pieces are individual contributions: Pratyush Maini is the main contributor of the analysis for data-based model extraction; on my end, I am the main contributor to the logits approximation method.

of the probability vector output by a ML model.

- We validate[2] our DFME technique in Chapter 2.5 on the SVHN and CIFAR10 datasets and successfully extract a model with 0.99x the victim accuracy with only 2M queries for SVHN, and 0.92x the victim accuracy with 20M queries for CIFAR10.
- An ablation study of our approach in Chapter 2.6 provides two key insights: (1) measuring disagreement between the victim and extracted models with the ℓ_1 norm achieves higher extraction accuracy than losses previously considered in the literature; (2) weak gradient estimates yield sufficient signal to train a generator despite only having access to the victim's predictions.

2.1 Related Work

We covered the seminal results in model extraction based on surrogate datasets in the introduction. Here, we discuss data-free knowledge distillation—the technique that underlies our approach to data-free model extraction—as well as the rudiments of generative modeling and gradient approximation required to understand our method.

2.1.1 Data-Free Knowledge Distillation

Knowledge distillation aims to compress, i.e., transfer, the knowledge of a (larger) teacher model to a (smaller) student model [BC14, HVD15]. It was originally introduced to reduce the size of models deployed on devices with limited computational resources. Since then, this line of work has attracted a lot of attention [ZXHL17,GYMT20,RBK$^+$15,ZK17, ZSG$^+$19]. While the model owner usually performs knowledge distillation, the original

[2]Code and models for reproducing our work can be found at https://github.com/cake-lab/datafree-model-extraction

dataset used to train the teacher model may not be available during distillation [MS19], e.g., because the dataset is too large or confidential. Therefore, others have proposed distillation techniques that leverage a surrogate dataset with a similar feature space or distribution [LFS17, OSF19]. Others proposed techniques that altogether remove the need for a surrogate dataset, i.e., data-free knowledge distillation [FSS+19, MS19]. Techniques addressing data-free knowledge distillation have relied on training a generative model to synthesize the queries that the student makes to the teacher [CCEKL20, MS19].

The success of data-free knowledge distillation hints at the feasibility of data-free model extraction. Kariyappa et al. observe this as well in concurrent work [KPQ20]. They also tackle data-free model extraction through the synthesis of queries by a generative model. Key differences include our loss formulation and optimizer choice (see Chapter 2.4). We show in Chapters 2.5 and 2.6 that our approach consistently outperforms theirs.

2.1.2 Generative Models

Model extraction through data-free distillation involves the generation of training data with which the *student* (i.e., adversary) queries the *teacher* (i.e., victim) model. Naively, one could generate these queries randomly [MS19, FSS+19]. In this work, we instead build on a min-max game between two adversaries that try to optimize opposite loss functions. This approach is analogous to the optimization performed in Generative Adversarial Networks (GANs) [GPAM+14] to train the generator and discriminator. Here, we use GANs in a fashion analogous to their application to semi-supervised learning [SGZ+16]: our student and teacher models, in conjunction, play the discriminator's role. The key difference here is that GANs are generally trained to recover an underlying *fixed* data distribution. However, our generator chases a moving target: the distribution of data which is most indicative of the discrepancies between the decision surfaces of the current student

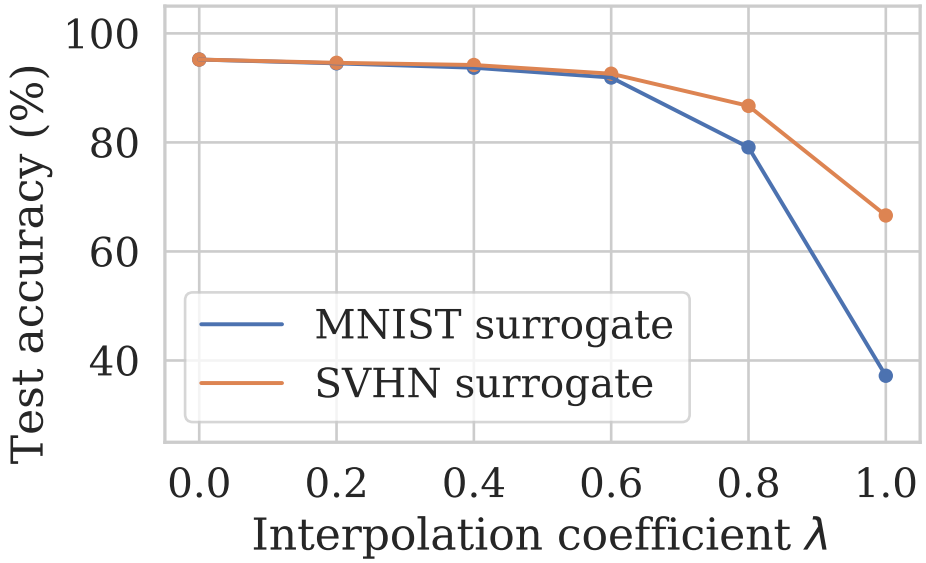

Figure 2.1: Dataset Interpolation with CIFAR10 as target. $\lambda = 0$ implies that the inputs are sampled from the target distribution, while $\lambda = 1$ implies sampling from the surrogate.

model and its teacher model.

2.1.3 Black-box Gradient Approximation

Zeroth-order optimization is a common approach to approximating gradients [WDBS18, NS17, CZS+17, LCK+20]. Such techniques have previously been used to mount attacks against ML models in a *black-box* setting, e.g., to craft adversarial examples [TTC+19, CZS+17, BHLS18]. Various gradient estimation methods solve different trade-offs between query complexity and the quality of the gradient estimate [TTC+19, CZS+17, BHLS18]. We use the forward differences [WWJD12] method for its relatively low query utilization, and systematically study the impact of its main parameter (e.g. the number of random directions) in Chapter 2.4.3.

2.2 How Hard is it to Find a Surrogate Dataset?

To motivate the need for data-free approaches to model extraction, we evaluate if an adversary must ensure that the distribution of its surrogate dataset is *close* to that of the victim's training dataset. We hypothesize that in the absence of this condition, distillation-based model extraction will return a poor approximation of the victim. We perform an analysis on the *closeness* of the distributions along three axes: (1) similarity in feature space, (2) marginal probability distribution of inputs, and (3) class-conditional probability distribution of the inputs. In our experiments, we attempt to steal ML models trained on CIFAR10 [Kri09] and SVHN [NWC+11] using various surrogate datasets that align differently with the axes defined above. We study in details the experimental setting, optimization problem, surrogate datasets and hyperparameters in Appendix A.1.

Our experiments support our hypothesis. For instance, in case of CIFAR10, with a victim model of accuracy 95.5%, extracting it using CIFAR100 [Kri09] as surrogate dataset results in extraction accuracy of 93.5%. This can be largely attributed to the fact that both the CIFAR10 and CIFAR100 datasets are subsets from the same TinyImages [TFF08] dataset. However, on using SVHN as surrogate dataset, the model extraction performance dropped remarkably, attaining a maximum of 66.6% across all the hyperparameters tried. In the extreme scenario when querying the CIFAR10 teacher with MNIST [LBBH98]– a dataset with disjoint feature space both in terms of number of pixels, and number of channels)— accuracy did not improve beyond 37.2%.

On the contrary, we notice that the victim trained on the SVHN dataset is much easier for the adversary to extract. Surprisingly, even when the victim is queried with completely random inputs, the extracted model attains an accuracy of over 84% on the original SVHN test set. We hypothesize that this observation is linked to how the digit classification task, at the root of SVHN, is a simpler task for neural networks to solve, and the underlying

representations (hence, not being as complex as for CIFAR10) can be learnt even when queried over random inputs.

While these correlations agree with our hypothesis, these experiments can not systematically quantify the *distance* between two distributions (viz. the surrogate and the target). To more systematically understand how the shift away from the target distribution affects extraction performance we interpolated inputs (x_{in}) from the surrogate (x_s) and target (x_t) datasets, s.t. $x_{in} = (1 - \lambda) \cdot x_t + \lambda \cdot x_s$. Figure 2.1 shows the decrease in extraction accuracy as the distribution diverges from target (CIFAR10) for two different surrogate datasets (SVHN and MNIST).

We make two conclusions from our observations: (1) the success of distillation-based model extraction largely depends on the complexity of the task that the victim model aims to solve; and (2) similarity to source domain appears to be critical for extracting ML models that solve complex tasks. We posit that it may be nearly as expensive for the adversary to extract such models with a good surrogate dataset, as is training from scratch. A weaker or non-task specific dataset may have lesser costs, but has high accuracy trade-offs.

2.3 Threat Model

We assume that the attacker has a user access to the prediction API. Therefore, he is able to query the victim model repeatedly and collect the predictions. However, the attack does not have any knowledge about the architecture of the deep learning model served by the API.

The victim model is a deep learning model trained for a classification task. For each query, the API returns a vector for probabilities (one for each class). The goal of the attack is to perform a model extraction without prior data nor knowledge about the training

data distribution. As opposed to pre-existing data-based extraction attacks, this work synthsizes all the queries that are made to the victim model.

Outside the scope of this work are defenses against model extraction attacks. Therefore, we assume that all the queries are processed by the API and receive a valid response. In other words, we assume no detection mecanism is in place.

2.4 Data-Free Model Extraction

The goal of model extraction is to train a student model $\mathcal{S}$ to match the predictions of the victim $\mathcal{V}$ on its private target domain $\mathcal{D}_{\mathcal{V}}$. That is to say, find the student model's parameters θ_S that minimize the probability of errors between the student and victim predictions $\mathcal{S}(x)$ and $\mathcal{V}(x)$ $\forall x \in \mathcal{D}_{\mathcal{V}}$:

$$\arg\min_{\theta_S} \ \mathcal{P}_{x \sim \mathcal{D}_{\mathcal{V}}} \left(\arg\max_i \mathcal{V}_i(x) \neq \arg\max_i \mathcal{S}_i(x) \right) \tag{2.1}$$

Since the victim's domain, $\mathcal{D}_{\mathcal{V}}$, is not publicly available, the proposed data-free model extraction attack minimizes the student's error on a synthesized dataset, $\mathcal{D}_S$. The error is minimized by optimizing a loss function, $\mathcal{L}$, which measures disagreement between the victim and student:

$$\arg\min_{\theta_S} \ \mathbb{E}_{x \sim \mathcal{D}_S} \left[\mathcal{L}(\mathcal{V}(x), \mathcal{S}(x)) \right] \tag{2.2}$$

This section describes how we minimize the number of queries made to the victim model with a novel query generation process, and how we train the student model itself.

2.4.1 Overview

The overall attack setup is inspired by Generative Adversarial Networks [GPAM+14]. A generator ($\mathcal{G}$) model is responsible for crafting some input images, and the student

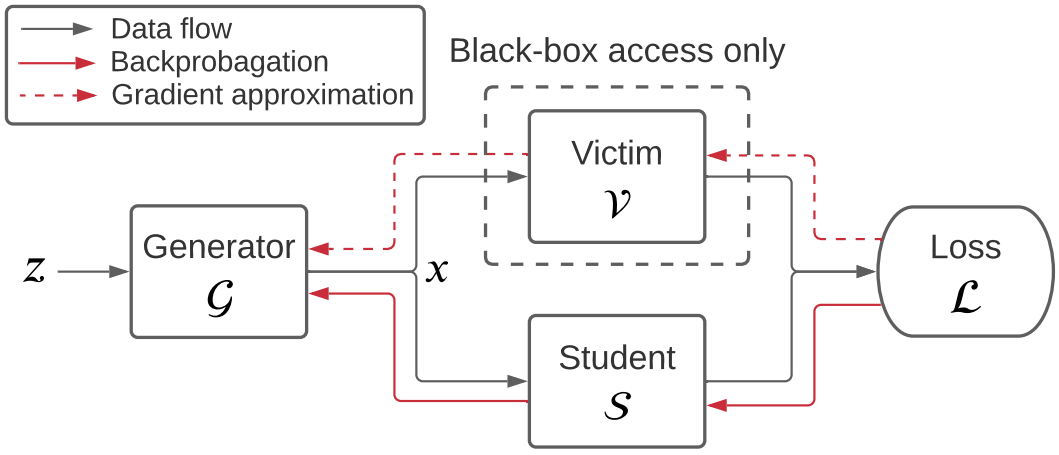

Figure 2.2: Date-Free Model Extraction Attack Diagram

model $\mathcal{S}$ serves as a discriminator while trained to match the victim $\mathcal{V}$ predictions on these images. In this setting, the two adversaries are $\mathcal{S}$ and $\mathcal{G}$, which respectively try to minimize and maximize the disagreement between $\mathcal{S}$ and $\mathcal{V}$.

The data flow is shown as a black arrow in Figure 2.2: a vector of random noise $\mathbf{z}$ is sampled from a standard normal distribution and fed into $\mathcal{G}$ which produces an image $\mathbf{x}$. Then the victim $\mathcal{V}$ and student $\mathcal{S}$ each perform inference on $\mathbf{x}$ to finally compute the loss function $\mathcal{L}$.

During the back-propagation phase (shown with red arrows) gradients from two different sources need to be computed: the gradients of $\mathcal{L}$ with regards to the student's parameters θ_S and the gradient of $\mathcal{L}$ with regards to the generator's parameters θ_G. Because the victim is only accessible as a black-box, it is not possible to propagate gradients through it. The dashed arrow indicates the need for gradient approximation (see Chapter 2.4.3).

Student. Prior work on knowledge distillation showed that a student model $\mathcal{S}$ can learn from a teacher and reach high accuracy even though its architecture is smaller and dif-

ferent [CH19, MS19]. Therefore, in the context of model extraction, the adversary only needs to select a model architecture which has sufficient capacity. This does not require knowledge of the victim architecture but rather generic knowledge of architectural choices made for the task solved by the victim (e.g., a convolutional neural network is appropriate for an object recognition task). In our work, we used a student with ResNet-18-8x architecture for model extraction.

The loss function $\mathcal{L}$ is used to measure the disagreement between $\mathcal{S}$ and $\mathcal{V}$. For this function, we use the ℓ_1 norm loss between victim and student logits (i.e. pre-softmax activations), $l_i(x)$ and $s_i(x)$ respectively. This requires us to recover the logits from the softmax outputs, since the adversary only has access to the later. We introduce an approach for doing so and further elaborate on the choice of $\mathcal{L}$ is detailed in Subsection 2.4.2. It is important to note that the gradient of the loss with regard to the student's weights θ_S does not require gradients of $\mathcal{V}$ since the victim's predictions don't depend on the weights θ_S.

Generator. The generator model $\mathcal{G}$ is used to synthesize images that maximize the disagreement between $\mathcal{S}$ and $\mathcal{V}$. The loss function used for $\mathcal{G}$ is the same as for $\mathcal{S}$ except that the goal is to maximize it. From this setting emerges an adversarial game in which $\mathcal{S}$ and $\mathcal{G}$ compete to respectively maximize and minimize the same function. In other words, the student is trained to match the victim's predictions and the generator is trained to generate difficult examples for the student. The adversarial game can be written as:

$$\min_{\mathcal{S}} \max_{\mathcal{G}} \mathbb{E}_{z\sim\mathcal{N}(0,1)} \left[\mathcal{L}(\mathcal{V}(\mathcal{G}(z)), \mathcal{S}(\mathcal{G}(z)))\right] \tag{2.3}$$

As shown in Figure 2.2, computing the gradient of $\mathcal{L}$ with regard to θ_G requires gradients of $\mathcal{V}$. As we only have access to $\mathcal{V}$ as a black-box, gradient approximation techniques are required. These techniques are discussed in Chapter 2.4.3.

Algorithm 1: Data-Free Model Extraction

Input: Query budget Q, generator iterations $n_{\mathcal{G}}$, student iterations $n_{\mathcal{S}}$, learning rate η, random directions m, step size ϵ
Result: Trained $\mathcal{S}$
while $Q > 0$ **do**
  for $i = 1 \dots n_G$ **do**
    $\mathbf{z} \sim \mathcal{N}(0, 1)$
    $x = \mathcal{G}(z; \theta_{\mathcal{G}})$
    approximate gradient $\nabla_{\theta_{\mathcal{G}}} \mathcal{L}(x)$
    $\theta_{\mathcal{G}} = \theta_{\mathcal{G}} - \eta \nabla_{\theta_{\mathcal{G}}} \mathcal{L}(x)$
  end
  for $i = 1 \dots n_S$ **do**
    $z \sim \mathcal{N}(0, 1)$
    $x = G(z; \theta_{\mathcal{G}})$
    compute $\mathcal{V}(x)$, $\mathcal{S}(x)$, $\mathcal{L}(x)$, $\nabla_{\theta_{\mathcal{S}}} \mathcal{L}(x)$
    $\theta_{\mathcal{S}} = \theta_{\mathcal{S}} - \eta \nabla_{\theta_{\mathcal{S}}} \mathcal{L}(x)$
  end
  update remaining query budget Q
end

Algorithm. Each iteration alternates training the generator $\mathcal{G}$ and student $\mathcal{S}$. To finely tune the balance between $\mathcal{G}$ and $\mathcal{S}$ training, each of these training phases is repeated n_G and n_S times, respectively, before moving on to the next epoch. While setting n_G higher allows $\mathcal{G}$ to train faster and to produce more difficult examples for $\mathcal{S}$, it can also be wasteful if $\mathcal{S}$ does not see enough examples. The trade-off between n_G and n_S is an additional hyperparameter that needs tuning. The additional hyperparameters m and ϵ are related to gradient approximation (see Chapter 2.4.3).

2.4.2 Loss function

Here we discuss different loss functions to measure the disagreement between $\mathcal{V}$ and $\mathcal{S}$. These losses are commonly used in the knowledge distillation literature given the similarity with the model extraction task [CH19,FSS$^+$19]. The choice of the loss function is key to the outcome of the attack since gradients computed through $\mathcal{S}$ and $\mathcal{V}$ can easily

impede the convergence of optimizers, e.g., if they vanish because the wrong loss function is used.

KullbackLeibler (KL) Divergence Most prior work in model distillation optimized over the KL divergence between the student and the teacher [CH19, HVD15, LZG18]. As a result, KL divergence between the outputs of $\mathcal{S}$ and $\mathcal{V}$ is a natural candidate for the loss function to train the student network. For a probability distribution over K classes indexed by i, the KL divergence loss for a single image x is defined as:

$$\mathcal{L}_{\text{KL}}(x) = \sum_{i=1}^{K} \mathcal{V}_i(x) \log \left(\frac{\mathcal{V}_i(x)}{\mathcal{S}_i(x)} \right) \tag{2.4}$$

However, as the student model matches more closely the victim model, the KL divergence loss tends to suffer from vanishing gradients [FSS+19]. Hypothesis 1 suggests that $\mathcal{L}_{\text{KL}}$ can make it difficult to achieve convergence while training $\mathcal{G}$ (refer to Appendix A.4 for justification). Specifically, back-propagating such vanishing gradients through the generator can harm its learning. We confirm this through empirical evaluation as well in Chapter 2.6.1.

Hypothesis 1. *The gradients of the KL divergence loss with respect to the image x should be small compared to the gradients of the ℓ_1 norm loss when $\mathcal{S}$ converges to $\mathcal{V}$:*

$$\|\nabla_x \mathcal{L}_{KL}(x)\| \underset{\mathcal{S}\to\mathcal{V}}{\ll} \|\nabla_x \mathcal{L}_{\ell_1}(x)\|$$

The ℓ_1 norm loss. To prevent gradients from vanishing, we use the ℓ_1 norm loss ($\mathcal{L}_{\ell_1}$) computed with the victim and student logits v_i and s_i where $i \in \{1...K\}$ and K is the number of classes. This was previously found by Fang et al. to prevent gradients from vanishing in knowledge distillation [FSS+19]. Even though $\mathcal{L}_{\ell_1}$ is not differentiable everywhere, it does not suffer from the vanishing gradients issue and yields better results

in practice (see Sec. 2.6.1). Lastly, the probabilities output by $\mathcal{V}$ need to be transformed into logits to be used in $\mathcal{L}_{\ell_1}$. We describe how to perform logit approximation in Appendix A.2, and evaluate in Sec. 2.6.3.

$$\mathcal{L}_{\ell_1}(x) = \sum_{i=1}^{K} |v_i - s_i| \tag{2.5}$$

2.4.3 Gradient Approximation

Because only black-box access is provided for $\mathcal{V}$, the optimizer aims at maximizing a function for which it only has an evaluation oracle. Yet, in order to train $\mathcal{G}$, gradients of the loss with regards to $\mathcal{G}$'s parameters $\nabla_{\theta_G}\mathcal{L}$ must be computed. Thus, we approximate gradients by interacting with the oracle: we maximize $\mathcal{L}$ with zeroth-order optimization.

Images as a Proxy

The number of parameters in $\mathcal{G}$ is typically large (millions of parameters) and it would be very query-expensive for a zeroth-order optimizer to get accurate gradient estimations on this large space. Instead, one can approximate gradients with regards to the input images x, and then back-propagate this gradient through $\mathcal{G}$ [KPQ20]. This way the dimensionality of gradients being approximated is much smaller, which yields more accurate zeroth-order approximations.

Additionally the oracle might only accept images that lie within a pre-defined input domain, for example $[-1, 1]^d$. To force $\mathcal{G}$ to respect this constraint, we use a hyperbolic tangent activation at the end of the generator architecture. Furthermore, zeroth-order gradients approximation methods usually evaluate the function in the neighborhood of a given point, which can result in query images slightly outside the input domain. To avoid this, we approximate gradients with regard to the pre-activation images (i.e. just before the hyperbolic tangent function is applied).

Forward Differences Method

The Forward Differences method approximates gradients by computing directional derivatives $\mathcal{D}_{\mathbf{u}_i} f(x)$ of a function f at a point x along m random directions $\mathbf{u}_i$. The directional derivatives are computed by measuring the variation of f a small step of size ϵ in the direction $\mathbf{u}_i$. They are then averaged to form an estimator of the gradient $\nabla_{\text{FWD}} f(x)$. In a way, each directional derivative brings some amount of information about true gradient. The estimator being more accurate as the number of random directions increases.

$$\nabla_{\text{FWD}} f(x) = \frac{1}{m} \sum_{i=1}^{m} d \frac{f(x + \epsilon \mathbf{u_i}) - f(x)}{\epsilon} \mathbf{u_i} \tag{2.6}$$

The main advantage of this method is that the number of query directions m may be chosen independently of the input space dimensionality, offering a trade-off between query utilization and gradient accuracy. This makes it an appealing candidate for DFME [KPQ20]. The influence of the number of query directions m is further described in Chapter 2.6.2.

Finite differences, an alternative gradient approximation method used when crafting adversarial examples [BHLS18], requires too many queries per gradient estimate to be viable for data-free model extraction.

2.5 Experimental Validation

We evaluate data-free model extraction (DFME) against victim models trained SVHN and CIFAR-10. We show that the resulting student models can reach high accuracy (e.g., 95.2% on SVHN) even when the generator only has access to inaccurate gradient estimates. Later in Chapter 2.6, we perform an ablation study and evaluate the impact of each attack component on the final student model accuracy and on the query budget Q.

2.5.1 Datasets and Architectures

We evaluate the effectiveness of the proposed DFME method on two datasets: SVHN and CIFAR-10. For each dataset, the victim model architecture is a ResNet-34-8x. These victim models were trained during 50 epochs for SVHN and 200 for CIFAR-10 with SGD at an initial learning rate of 0.1, decayed by a factor of 10 at 50% of training.

We use ResNet-18-8x as the architecture for our student model. This is inspired by previous works in knowledge distillation [FSS$^+$19] that show how a smaller student is sufficient to distill the knowledge of a larger teacher. The network was trained with a batch size of 256 with SGD, with an initial learning rate of 0.1, a weight-decay of 5.10^{-4}, and a learning rate scheduler that multiplies the learning rate by a factor 0.3 at $0.1\times$, $0.3\times$, and $0.5\times$ the total training epochs. The default query budget Q is 2M for SVHN, and 20M for CIFAR-10 in our experiments.

The generator used three convolutional layers, interleaved with linear up-sampling layers, batch normalization layers, and ReLU activations for all layers except the last one. The final activation function was the hyperbolic tangent function to output values in the range [-1,1] (see Chapter 2.4.3). It was also trained with a batch size of 256, but using an Adam optimizer with an initial learning rate of 5.10^{-4} which is decayed by a factor 0.3 at 10%, 30%, and 50% of the training.

For gradient approximation we sample $m = 1$ random directions and a step size $\epsilon = 10^{-3}$.

2.5.2 Results

We compare the performance of different extraction attacks in Table 2.1. We measure the ratio between the student's accuracy and the victim's accuracy on the victim's test set. This helps compare the performance of DFME across different datasets. The stu-

dent model's normalized accuracy is reported for each dataset and extraction method evaluated—our approach (DFME), our approach with KL divergence loss (DFME-KL), our approach without logit correction (Log-Probabilities), and concurrent work [KPQ20] (MAZE). Further, we perform DFME with a range of query budgets and reported the accuracy in Figure 2.3.

Without any knowledge of the original training distribution, the proposed DFME method achieved as high as 88.1% (0.92x target) of accuracy with Q = 20M and 89.9% (0.94x target) with Q = 30M. The accuracy of the extracted model exceeds that reported in concurrent work which we refer to as MAZE in our results [KPQ20]. However, in our best-efforts at reproducing their results with the details in the paper,[3] we were unsuccessful in achieving the same reported accuracy, and were only able to achieve an accuracy of 45.6% (0.48x target) at best on the student model. In addition, MAZE reports that they were unable to learn when using extremely few directions (such as $m = 1$) for the gradient approximation with CIFAR10, whereas we find that weak gradient approximations are beneficial to reduce the overall query budget of successful attacks.

We observed similar results for the SVHN victim model: reaching as high as 95.2% (0.99x target) accuracy with only 2M queries. The task for SVHN is much simpler than CIFAR10 given that a model with 84% (0.87x target) accuracy can be extracted from just random noise. The proposed method allows one to achieve far higher accuracy.

One limitation of our study is that the reported query budgets do not include the cost of hyperparameter tuning. This is an important direction for future work as preliminary experiments suggest that extraction accuracy can be sensitive to the choice of hyperparameters.

[3]The authors declined to share their code upon request.

Dataset (budget)	Victim accuracy	DFME	DFME-KL	MAZE* [KPQ20]	Log-Probabilities
CIFAR10 (20M)	95.5%	88.1% (0.92×)	76.7% (0.80×)	45.6% (0.48×)	73.2% (0.77×)
SVHN (2M)	96.2%	95.2% (0.99×)	84.7% (0.88×)	91.1% (0.95×)	94.4% (0.98×)

Table 2.1: Accuracy and normalized accuracy of data-free model extraction methods. Results for 'MAZE' reflect our best-effort reproduction.

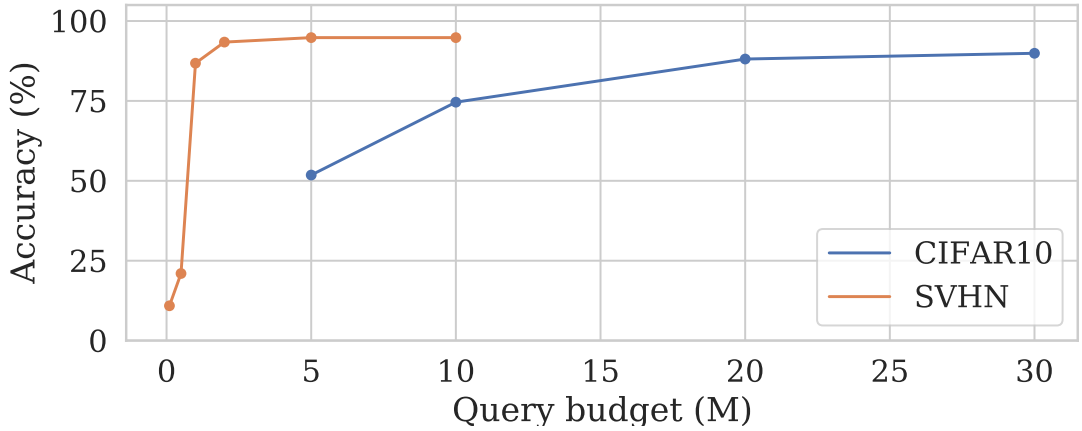

Figure 2.3: Test accuracy wrt query budget, for SVHN and CIFAR10

2.6 Ablation Studies

Our work systematically transitions from a data-free knowledge distillation paradigm [FSS$^+$19, MS19] to a data-free model extraction scenario. The main challenges in this transition were (1) to surpass the need for true gradients for training the student; (2) the lack of access to true victim logits; and (3) the need to restrict the query complexity of the attacks (to reduce the cost of stealing). With this goal, we made specific choices with regards to (a) the loss function; (b) gradient approximation; and (c) logit access. In this section, we detail the impact of each of these choices to the final performance of our proposed DFME method.

2.6.1 Choice of Loss Function

The choice of loss is of paramount importance to a successful extraction. In our DFME approach, the choice of loss involves similar factors to those outlined in research on

GANs: multiple works have discussed the problem of vanishing gradients as the discriminator becomes strong in case of GAN training [AB17, ACB17]. For our DFME approach, we minimize the ℓ_1 distance between the output logits of the student and the teacher. We find that this significantly improves convergence and stability over other possible losses, such as the KL divergence chosen in [KPQ20].

We perform DFME in the same setting to evaluate the difference between the KL divergence and ℓ_1 losses. Below, we draw comparisons based on two metrics: (1) Final accuracy attained by the student at the end of a fixed number of queries as well as the learning curves of the student; and (2) The norm of gradients of the loss with respect to the input image as the training progresses.

Test Accuracy. The key metric of interest for this comparison is the normalized accuracy of the student model at the end of a designated query budget Q of 20M queries for CIFAR10 and 2M queries for SVHN. Table 2.1 shows that using the ℓ_1 loss achieves significantly better test accuracies compared to the KL divergence loss. For instance, on CIFAR10 the accuracy improves from 76.7% to 88.1% when switching from KL divergence to the ℓ_1 loss. We also visualize a learning curve for CIFAR10 in Figure 2.4: the KL divergence objective slows converge and tappers off earlier, even when the student has yet to plateau.

Gradient Vanishing. The KL divergence loss suffers from vanishing gradients, as explained in Chapter 2.4.2. In DFME, these gradients are used to update the generator's parameters and are thus essential to synthesize queries which extract more information from the victim. In Figure 2.5 we empirically demonstrate that as the student accuracy approaches that of the victim model, the gradients of the KL divergence loss with respect to the input image reduce significantly in norm. The same decay is slower and less significant in case of the ℓ_1 loss. We hypothesize that these vanishing gradients are the cause

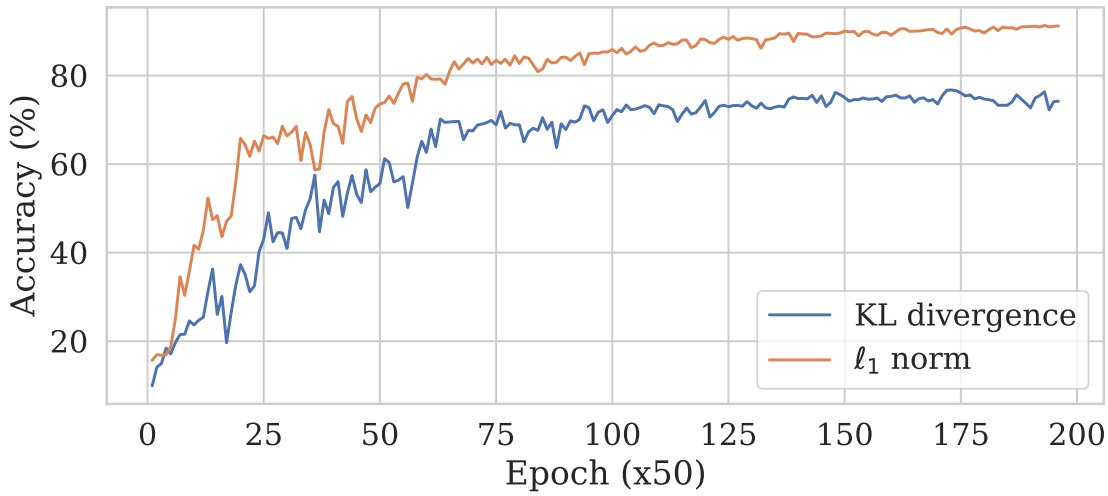

Figure 2.4: Test accuracy as training progresses for ℓ_1 and KL divergence losses.

m	1	3	5	8	10
No. of Queries	10.04	10.02	16.33	13.80	20.00

Table 2.2: Minimum queries (in millions) to reach 85% accuracy on CIFAR10, for different number of gradient approximation steps.

for degraded accuracy when using the KL divergence loss.

2.6.2 Gradient Approximation

Recall that the ℓ_1 loss cannot be back-propagated through the victim since the adversary only has access to it as a black-box. Recent work on data-free model distillation [FSS+19] has claimed that the gradient information from the teacher is 'indispensable at the beginning of adversarial training' because the student alone can not provide useful signal to the generator when randomly initialized. Below we consider: (1) the quality of approximation required; and (2) the overall impact on query complexity. In particular, we compare two approaches for improving the training of our DFME generator: using an increased number of queries to compute more accurate gradient estimates or training generator for longer using poorer gradient estimates.

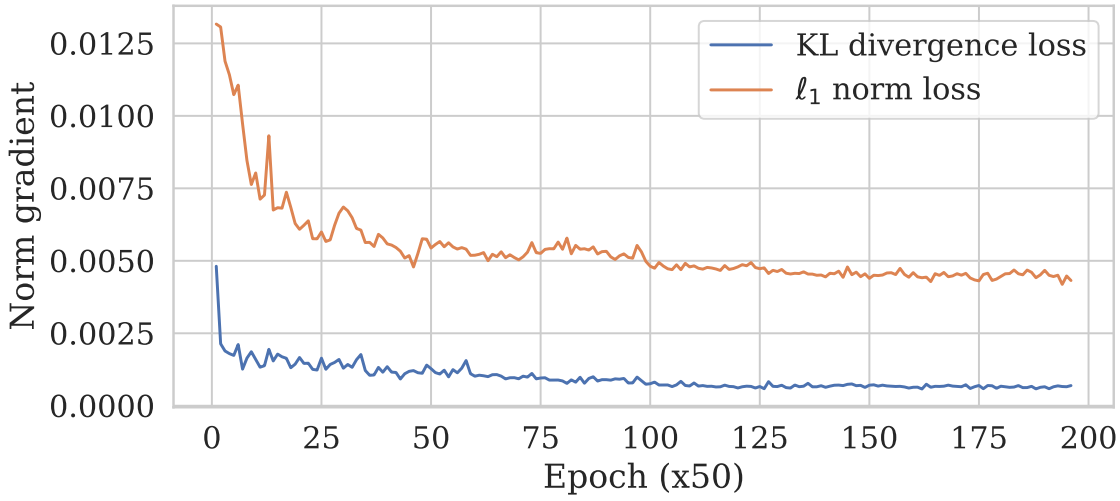

Figure 2.5: Norm of gradients with respect to the input image, for the KL divergence and ℓ_1 norm losses.

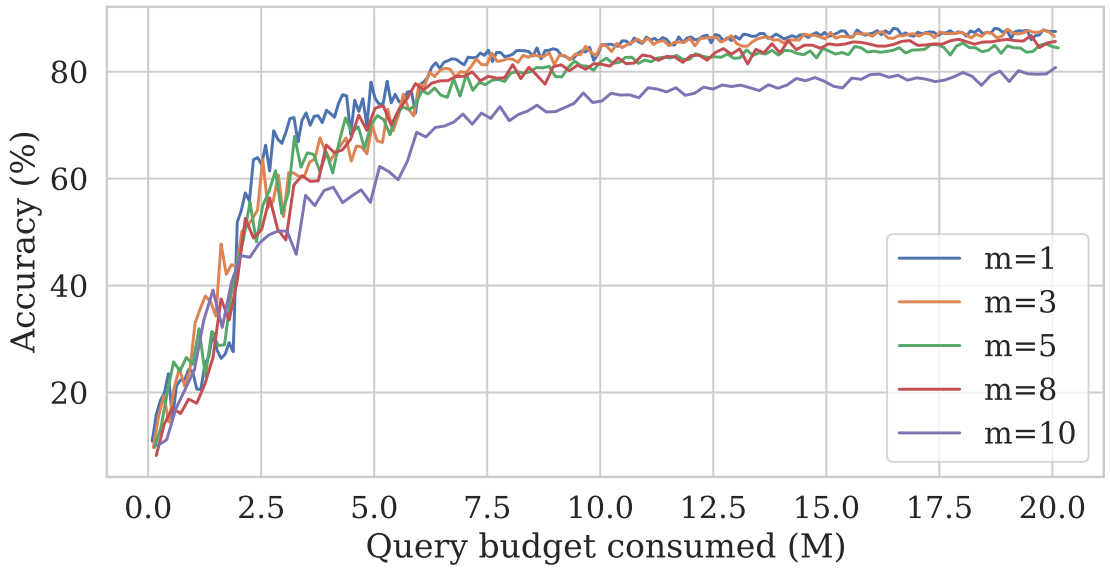

Figure 2.6: Accuracy of the model during training for different number of gradient approximation steps, m.

Model	MTL (2.3e-6)	MC (2.3e-6)	LP (1.3)
Resnet-34-8x	-1.24e-5	1.24e-5	4.98
Densenet 121	5.53e-7	1.88e-6	3.88
VGG 16	1.97e-5	1.97e-5	3.93

Table 2.3: Mean of true logits (MTL) for 3 victim architectures; reconstruction error (MAE) between approximate and true logits when using mean correction (MC) and log-probabilities (LP).

Number of gradient approximation steps. When gradients used to update the generator are approximated with the forward differences method, a larger number of random directions m allows one to compute more accurate gradients. However, in a model extraction setting, each additional gradient approximation step comes at the cost of increased query complexity. In practice, with a fixed query budget Q, changing the number of random directions directly impacts the proportion of queries used to train each network. This ratio of queries r used to train the student is given by:

$$r = \frac{n_S}{n_S + (m+1)n_G} \tag{2.7}$$

In our setting (i.e. $n_G = 1$, $n_S = 5$) choosing m equal to 1 or 10 respectively results in 71% and 31% of the query budget being used to directly train the student, while the remainder is spent to get better gradient estimates to train the generator. Despite using a majority of queries to train the generator, the setting with $m = 10$ achieves comparable accuracy. In Table 2.2, we observe how choosing lower values of m achieves 85% test accuracy in much fewer queries.

Amortizing the cost. We hypothesize that since early in the training the discriminator (or student) provides only little signal, it is beneficial for the generator to initially rely on weak signals of gradient approximation. Effectively, this helps amortize the cost of gradient approximation over multiple epochs, and effectively pushes the expense to a

later stage when the discriminator (or student) provides stronger signal. Figure 2.6 shows that relative to the query budget utilization, different values of m perform similarly.

This also suggests a hybrid strategy where the adversary first extracts a (somewhat poor) student model through distillation from a surrogate dataset. Indeed, we show in Chapter 2.2 that surrogate datasets drawn from a different distribution (than the victim model's training data) enable distillation-based model extraction—albeit to a lower accuracy than in-distribution surrogate datasets. This poor initial student can then be improved by synthesizing queries with the data-free model extraction's generator to bring the student model closer to the victim model's performance.

Case $\mathbf{m = 1}$. In the extreme case where the number of gradient approximation steps for forward differences (m) is set to 1, the approximated gradient is colinear to the random direction sampled for the approximation, but always points in the direction that helps maximize the loss (i.e. its projection onto the true gradient is positive). The cosine similarity with the true gradient is, thus, very small. To validate this effect, we additionally experiment with $m = 1$ where the approximate gradient was randomly flipped to the wrong direction with half probability. As hypothesized, the student accuracy did not improve beyond 20% in our experiments on CIFAR10. This suggests that computing gradients that are extremely inaccurate makes it possible to train the student as long as these gradients are in the correct direction.

2.6.3 Impact of Logits Correction

A model extraction attack should be applicable to the nature of predictions offered by MLaaS APIs. Most APIs provide per-class probability distribution rather than the true logits, since probabilities are more easily interpreted by the end user. To perform model extraction successfully we thus need to recover the logits from the victim's prediction

probabilities. We show that is is possible to do so and recover approximate logits whose Mean Average Error (MAE) with the true logits is low, on three different victim architecture.

The MAE reported in Table 2.3 are negligible compared to true logits which take values in the order of magnitude of 1. Therefore, the adversary can use these approximate logits in lieu of the true logits. In comparison, approximating true logits with plain log-probabilities resulted in a MAE in the order of magnitude of the true logits themselves. Using the log-probabilities with such a large error makes the student training harder—it did not yield accuracy above 75%.

This method is effective because the mean of the true logits is nearly 0 (see Table 2.3). Therefore, subtracting the mean from the log-probabilities is equivalent to subtracting the additive constant $C(x)$ itself.

2.7 Summary

In this chapter, we demonstrate that data-free model extraction is not only practical but also yields accurate copies of the victim model. This means that model extraction attacks are a credible threat to the intellectual property of models released intentionally or not to the public, even when no data is available. By leveraging a well-chosen loss, i.e. the ℓ_1 loss, we achieved higher accuracy than concurrent work [KPQ20]. As a side-product of this attack, we proposed an accurate method to recover logits out of the probability vector.

Chapter 3

Confidential Deep Learning

In this chapter, we characterize two bottlenecks that impact TEE performance and consider methods to address them. For the *page thrashing bottleneck* described in Chapter 3.5, we propose a data partitioning scheme, *y-plane partitioning*, that allows for efficient computation of convolutional layers in TEEs with as little as 28MB of secure memory. Additionally, in Chapter 3.6, we identify a previously unexplored performance bottleneck, the *decryption bottleneck*, that arises from parameter decryption and propose a mitigation strategy based on compression and quantization. We used SGX-based TEEs on Microsoft Azure cloud servers [Gor18] to measure the impact of these bottlenecks and evaluate the proposed solutions. For the most extreme case (shown in Table 3.1), the bottlenecks increased model latency to 26X over the unmodified baseline, while the proposed optimizations reduced model latency from 26X to 1.09X.

The optimizations proposed in this study significantly reduce the per-layer memory footprint for a model, which is a limiting factor for prior work such as Vessels [KKR+20]. Further, we demonstrate that the proposed y-plane partitioning scheme offers complimentary design tradeoffs, with different strengths and weaknesses, to channel partitioning [LLP+19]. Our evaluation suggests that a combination of y-plane and channel parti-

		Inside TEE	
	Outside TEE (s)	**Optimized (s)**	**Unmodified (s)**
28MB+1vCPU	3.174	3.468 (1.09X)	84.639 (26.73X)
56MB+2vCPU	1.858	2.430 (1.31X)	28.599 (15.39X)
112MB+4vCPU	1.112	1.868 (1.68X)	11.256 (10.12X)
168MB+8vCPU	0.808	1.667 (2.06X)	4.377 (05.42X)

Table 3.1: **Model Latency of VGG-16.** The "Optimized" column records the latency improvements after applying the optimizations proposed in Chapters 3.5 and 3.6. Each row represents an SGX enclave configuration; for example, 28MB+1vCPU means the enclave has 28MB of secure memory and 1 virtual CPU core. Numbers are averaged over 30 runs.

tioning provides the smallest memory footprint for convolutional layers. The choice of scheme depends on the size of the layer's output versus the size of the weights. Finally, reducing memory footprint improves model latency and allows for greater concurrency, allowing more TEEs to coexist on the same system [KKR+20]. We leave an exploration of model concurrency for future work.

In summary, we make the following contributions:

- the introduction of a novel *y-plane partitioning* scheme that complements channel partitioning, alleviating the page thrashing bottleneck and reducing the memory footprint of convolution layers;
- a characterization of the previously unexplored decryption bottleneck in fully-connected layers;
- an evaluation of quantization and compression as a means to address the decryption bottleneck and reduce the memory footprint of fully-connected layers.

3.1 Background

Trusted Execution Environments. While the exact capabilities of TEE implementations

vary, some of the more common security features include *(i)* isolation, i.e., confidentiality and integrity of the code and data located inside the TEE, and *(ii)* remote attestation, the ability to verify the state of the TEE remotely. These properties are why recent works have explored TEEs as a means to protect the confidentiality of both user [LLP+19] and model [KKR+20] data when running deep learning inference on untrusted hardware.

An *SGX enclave* is a TEE implementation provided by Intel's software guard extensions (SGX) [SGX15]. SGX enclaves include an area of *secure memory*, called the *processor reserved memory (PRM)*, which is isolated from the rest of the system. This secure memory is only accessible from code within the enclave. The secure memory size is usually small relative to the rest of the system; typically, far less than what deep learning models require for inference. For example, the enclaves used for this study offered between 28MB and 168MB of memory, whereas VGG-16 [SZ14] requires over 1GB of memory. To support programs with higher memory requirements, SGX provides paging mechanisms to encrypt and swap memory pages between secure and main memory. When code running inside the enclave attempts to access a virtual memory address on a page that is not currently in the enclave, a *page fault* is raised. SGX transparently services this page fault: evicting an older page and transferring, decrypting, and checking the requested page's integrity. We refer to this as *secure paging*.

Convolutional Neural Networks. *Convolutional neural networks (CNNs)* are a type of deep neural network that contain neurons organized into *layers*, including the eponymous convolutional layers. CNNs are commonly used for vision tasks but are garnishing attention in other domains. The process of using a CNN model for classification is referred to as *inference* or *model execution* and the time taken to perform this inference is called *model latency*.

Layer execution refers to the process of transforming the *inputs* (i.e., the output of the previous layer) and *parameters (e.g., weights)* into the *outputs* for an individual layer. The

precise computation performed depends on the layer's type. We collectively refer to the model's static parameters (e.g., weights) and any values calculated at runtime as *model data*.

Broadly, CNNs use three different types of hidden layers: *fully-connected*, *convolutional*, and *pooling layers*. The inputs and outputs of convolutional and pooling layers are 3D arrays which resemble stacks of 2D images called *channels* (e.g. the RGB channels of an image). In fully-connected layers, the inputs and outputs are simple 1D vectors.

3.2 Related Work

Deep Learning and Trusted Execution Environments. In Vessels [KKR+20], Kim et al. optimize the memory usage of neural networks in TEEs by analyzing the dependency graph of the model's layers and then allocating a memory pool in which only the required data is stored at any given time. The rationale is that the sequential nature of neural networks' architecture allows reusing most memory buffers, avoiding unnecessary paging. Furthermore, as all of the computations are done in a pre-allocated memory area, a single machine can host multiple enclaves to compute different models concurrently. As long as the different enclaves do not fill up the secure memory, the contention is minimized. The limiting factor for such a system is the size of the memory pool, which relies primarily on the size of the largest layers.

Partitioning is one mechanism to reduce the per-layer memory requirements. For example, we propose a convolution-layer partitioning scheme, y-plane partitioning, in Chapter 3.5. Another example is Occlumency's *channel partitioning*. Occlumency [LLP+19] is an inference framework implemented on top of SGX that uses channel partitioning to divide the computation and memory requirements of convolutional layers. We compare Occlumency's channel partitioning scheme to y-plane partitioning and explain why

a combination of both schemes provides the best performance in Chapter 3.5.4.

Grover et al. proposed Privado [TGS+18], a system designed to remove any input-dependent memory accesses, thereby preventing information leakage from the TEE. Chiron [HSS+18] uses SGX enclaves to train machine learning models, protecting the confidentiality of the user's training data, the model's architecture, and the training procedure. Neither work attempts to address the performance challenges described in this paper.

Encryption for Deep Learning. Cryptographic techniques offer an alternative to trusted execution environments for maintaining user privacy [GBDL+16,MLS+20]. These techniques rely on homomorphic encryption to process encrypted data on a server. Such systems usually have high inference latencies, which they make up for with high throughput. Thus, these systems are more appropriate for processing large batches of input data. Further, existing cryptographic systems like CryptoNets [GBDL+16] do not protect model weights from disclosure—protecting the model confidentiality in Cryptonets would significantly degrade performance.

3.3 Threat model

Our work focuses on performance rather than security when running a deep learning model in a Trusted Execution Environment. For context we describe the thread model of TEE model confidentiality:

The model owner is concerned with protecting the intellectual property of their trained deep learning model. The hardware provider supplies the system on which the trusted execution environment is created. For example, in the case of cloud-based inference, the hardware provider is the company that maintains the cloud infrastructure. We assume the TEE supports secure paging.

The model owner is concerned that the hardware provider will attempt to steal model

data *during execution*. Further, as the hardware provider may have control over the system's operating system, the model owner must assume that a malicious hardware provider will succeed if that data is ever placed unencrypted in memory accessible outside of the TEE.

3.4 Methodology

We conducted our experiments on virtual machines provided by Microsoft's Azure cloud computing infrastructure. The four tested enclave configurations represent all of the configurations offered by Microsoft Azure at the time of writing. We refer to each VM using its enclave size and number of virtual CPUs; for example, *28MB+1vCPU* refers the VM configuration with 28MB of secure memory and 1 virtual CPU. All configurations ran Ubuntu 18.04 and used an Intel Xeon E-2288G CPU. Unless otherwise specified, the number of execution threads for each system was equal to the number of virtual CPUs—this is why the baseline model latency varies, for example.

Our evaluation methodology emphasizes the *per-layer* performance of convolutional neural networks (CNNs). Focusing on individual layers offers two distinct benefits. First, it allows us to examine each of the components in isolation. Second, it helps us determine the performance implications for a variety of CNN architectures. For example, we observed that the performance benefits offered by quantization and compression for the large fully-connected layers in VGG-16 directly translated to performance benefits for the large fully-connected layer in AlexNet—though we elide the AlexNet numbers for space. Consequently, we focus primarily on the VGG architecture as VGG models contain a variety of fully-connected and convolutional layers that range in size, shape, and memory requirements.

We use Darknet as the baseline inference framework due to its portability. In par-

ticular, Darknet uses C and lacks external dependencies, making it possible to port the framework to SGX with relatively minor changes. In contrast, PyTorch is a more popular framework, but moving model execution entirely into the TEE would require significant engineering efforts.

3.5 The Page Thrashing Bottleneck

The mismatch between enclave size and convolutional-layer memory requirements manifests as inefficient paging patterns during model inference, i.e., a *page thrashing bottleneck*. In this section, we characterize this phenomenon. We then propose y-plane partitioning as a means to mitigate this bottleneck. We compare y-plane partitioning to a prior scheme and argue that the combination offers the best latency and smallest memory footprint.

3.5.1 Characterization

As observed by Lee et al. [LLP+19], the challenge for convolutional layers is that the memory access pattern during execution leads to page thrashing, i.e., the constant transfer of pages into and out of the TEE.[1] Every transfer between secure and main memory adds significant overhead.

Darknet—as many usual deep learning frameworks—uses the im2col transformation to speed up convolutional layer execution. This transformation expands the 3D input array into a large 2D matrix and organizes the weights into a different 2D matrix. These transformations allow the convolution operations to be computed using a large matrix-matrix multiplication. They thus benefit from highly optimized *general matrix to matrix multiplication (GEMM)* functions provided by BLAS libraries (e.g. OpenBLAS [XQC12]).

[1] Denning [Den68] defines thrashing as "excessive overhead and severe performance degradation or collapse caused by too much paging."

This method's inherent tradeoff is that the im2col transformation duplicates the inputs, resulting in a transformed im2col matrix that is significantly larger than the original input array.

The above scheme has unintended consequences when used naively in the TEE. First, im2col's expansion of the input array—a factor of 9 in VGG-16—causes many memory pages to be evicted from the TEE only to be brought back into the TEE during the matrix-matrix multiplication. Second, as the im2col matrix cannot fit entirely in TEE memory, the pattern of memory accesses to this matrix has important performance implications. For example, in Darknet the output matrix is computed row by row, resulting in an unfavorable memory access pattern that triggers cascading evictions and page faults. In particular, computing one row of the output requires a lookup of the entire transformed im2col matrix, and this lookup process is repeated for all rows of the output.

In our experiments with Darknet and VGG-16, we observed that convolutional layers cause more page evictions, by multiple orders of magnitude, when run in a 28MB enclave versus the 168MB enclave. Layer 8 triggers 1.8 million page evictions in the 28MB enclave, but only 1,700 in the 168MB enclave.

3.5.2 Partitioning

Partitioning addresses the thrashing bottleneck by applying the im2col transformation to a subset of the input array. The subset, i.e., partition, can be processed efficiently using the limited secure memory of the TEE.

We evaluate two partitioning schemes: *y-plane partitioning* and *channel partitioning*. The former is a contribution of this work, and the latter was previously used as part of Occlumency [LLP+19]. While channel partitioning splits the input by channels, *orthogonal* to the depth direction, *y-plane partitioning* uses planes *parallel* to the depth direction; Figure 3.1 provides a visual representation of y-plane partitioning.

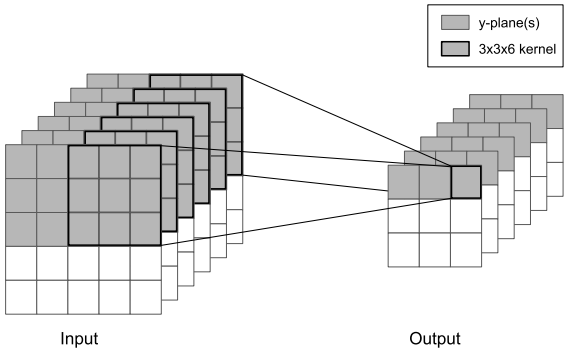

Figure 3.1: **Illustration of Y-Plane Partitioning.** A 5x5x6 input is convolved with a 3x3x6 kernel. This figure highlights the computation of 1 output value. Three input y-planes are required to compute one output y-plane.

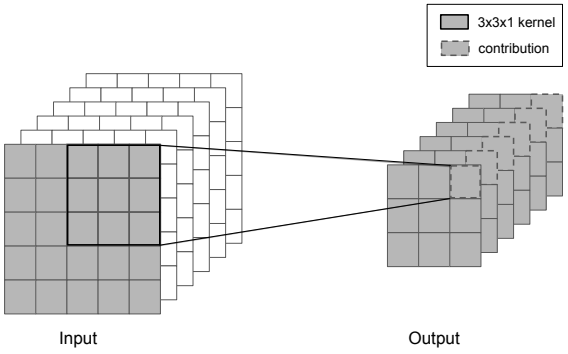

Figure 3.2: **Illustration of Channel Partitioning.** This figure highlights the contribution of 1 input channel to 1 value on each output channel. Each input channel contributes to the entire output.

Layer	Input (MB)	28MB+1vCPU Latency (s)	28MB+1vCPU Evictions (#)	56MB+2vCPU Latency (s)	56MB+2vCPU Evictions (#)	112MB+4vCPU Latency (s)	112MB+4vCPU Evictions (#)	168MB+8vCPU Latency (s)	168MB+8vCPU Evictions (#)
Outside TEE									
2	123	0.547	-	0.340	-	0.219	-	0.168	-
5	61	0.455	-	0.274	-	0.152	-	0.106	-
8	31	0.416	-	0.237	-	0.127	-	0.085	-
Partitioning in TEE									
2	123	0.399	8,270	0.223	3,503	0.162	3,251	0.153	3,184
5	61	0.296	1,999	0.177	1,716	0.120	1,718	0.101	1,712
8	31	0.292	1,374	0.172	1,364	0.102	1,362	0.084	1,360
Unmodified in TEE									
2	123	17.899	**1,859,815**	10.989	**1,881,017**	6.153	**1,858,259**	0.746	30,286
5	61	17.664	**1,838,563**	10.651	**1,848,772**	0.682	8,799	0.381	1,729
8	31	17.556	**1,827,409**	1.121	4,909	0.538	1,361	0.323	1,700

Table 3.2: **Latency and Page Evictions for Convolution Layers using Y-plane Partitioning.** Only the three convolutional layers with the largest inputs are shown. Input size was measured after the im2col transformation.

At a high level, both schemes first split the input into partitions and compute the contribution of that partition to the output by *(i)* applying im2col on each partition, *(ii)* computing a matrix-matrix multiplication with the corresponding subset of the weight matrix, and *(iii)* adding the result to the output buffer. Y-plane and channel partitioning offer different design tradeoffs, with complementary strengths and weaknesses. In particular, we find that y-plane partitioning is more memory-efficient when the layer output is large, while channel partitioning is better when the weight matrix is large.

Y-Plane Partitioning. As illustrated in Figure 3.1, *y-planes* are the concatenation of one row from each channel of a 3D array. For this scheme, a partition is a group of contiguous y-planes; both the layer's inputs and outputs are logically divided into y-plane partitions. Each output y-plane is computed from a small and contiguous subset of the input y-planes. The convolution kernel size and stride determine the relationship between input and output y-planes.

Each round of computation involves three elements: *(i)* an output partition composed of contiguous y-planes, *(ii)* the corresponding subset of input y-planes, and *(iii)* the entire weight array. This repeated access to the entire weight array makes the weights size the limiting factor of y-plane partitioning.

Channel Partitioning. Channel partitioning, illustrated in Figure 3.2, divides the input into partitions of one or more channels [LLP+19], using a partition of the weights to calculate each contribution to the output. Note that the output is not partitioned and needs to be accessed during every round of computation to add the input-weight partition pairs' contribution. Thus, the output size is the limiting factor.

In practice, deep neural networks contain many convolutional layers, and the output and weight sizes of each layer vary. This observation, along with the aforementioned differences between y-plane and channel partitioning, suggests that a combination could yield the best results. Such a scheme would use the best partitioning scheme for the given

layer. Further, the cost of switching from y-plane to channel partitioning (and vice versa) is negligible. We explore this idea in Chapter 3.5.4.

3.5.3 Performance of Y-plane Partitioning

Table 3.2 illustrates the page thrashing bottleneck in convolutional layers, showing the impact of enclave size on latency and page evictions for the unmodified baseline running outside of the TEE, inside of the TEE, and inside of the TEE with y-plane partitioning. We make several observations that are consistent with prior work [LLP+19].

First, with y-plane partitioning, the convolutional layer latency decreased significantly and remained stable for all secure memory sizes. Second, page thrashing in Darknet was triggered when the size of the im2col-transformed input exceeded the enclave size—as measured by the drastic difference in page evictions. For example, layer 2, with its 123MB input, saw approximately 1.8 million page evictions for all three enclaves with less than 123MB of secure memory, but only 30 thousand evictions for the enclave with 168MB of memory.

Third, Darknet's per-layer latency varied dramatically, ranging from more than 17 seconds when thrashing occurred and less than 1 second when thrashing did not occur. As the total number of floating point operations remained constant, this difference resulted from thrashing.

3.5.4 Combining Y-Plane and Channel Partitioning

Different factors limit Y-Plane and Channel partitioning. Below we demonstrate those differences using a model with layer sizes that far exceed the available secure memory. We show that a combination of y-plane and channel partitioning allows us to execute this model without thrashing, whereas either scheme would fail if used in isolation.

			Y-Plane		Channel	
	Weights (MB)	Output (MB)	Latency (s)	Evictions (#)	Latency (s)	Evictions (#)
1	0.01	**49.44**	0.544	48,883	1.387	**123,145**
2	0.14	**49.44**	1.877	62,964	21.552	**1,809,386**
4	0.28	**24.72**	0.891	22,001	9.216	**868,540**
5	0.56	**24.72**	1.638	29,444	18.204	**1,716,377**
7	1.13	12.47	0.770	5,310	1.205	5,301
8	2.25	12.47	1.542	10,283	2.392	7,680
10	4.50	1.64	0.288	1,606	0.166	1,579
11	9.00	1.64	0.583	2,916	0.330	2,727
13	9.00	0.44	0.239	2,434	0.107	2,420
14	9.00	0.44	0.237	2,419	0.107	2,422
16	17.58	0.06	0.112	4,543	0.066	4,602
17	**34.33**	0.06	0.467	**35,689**	0.126	8,901
18	**68.66**	0.12	0.925	**70,979**	0.252	17,885
19	**68.66**	0.06	0.925	**70,878**	0.253	17,762

Table 3.3: **Per-Layer Latency and Page Evictions for VGG-Large.**

Methodology. To scale up the model, we preserved most layer parameters (stride, padding, etc.), types, structure, and order of VGG-16. We only scale up two parameters: *(i)* the input resolution, which has an impact on the input and output size in all the layers, and *(ii)* the number of kernels in the first layer, which impacts the inputs, outputs, and weights size in all the layers. We chose an input resolution of 450×450 and 64 kernels in the first layer, so that some layers have either their output or weights larger than the enclave size. We call this model *VGG-Large*.

Results. Table 3.3 shows the per-layer page evictions and inference latency for y-plane partitioning and channel partitioning. We make four observations of these results. First, when the output is large compared to the secure memory size, as is the case in the first few layers, channel partitioning will start thrashing. In contrast, y-plane partitioning divides the output and, consequently, saw up to 58X fewer page evictions than channel partitioning.

Second, in the last few layers the weights are larger than secure memory, and y-plane partitioning shows up to 4.0X more page evictions than channel partitioning. This behavior is expected as y-plane partitioning does not divide the weights, but channel partitioning does.

Third, each scheme out-performed the other for a subset of the layers. In other words, using y-plane and channel partitioning in conjunction allows for efficient computations for models that neither y-plane nor channel partitioning could handle without page thrashing. To completely avoid page thrashing with VGG-Large, an enclave of at least 68 MB (resp. 50 MB) would be needed to run this model with y-plane-only (resp. channel-only) partitioning; while the hybrid scheme can run it with just 28MB. This experiment also shows that a large model can be ran with a significantly reduced memory footprint; even if it can fit in memory. This result is useful in practice as systems that provide concurrency for secure deep learning inference, like Vessels [KKR$^+$20], are limited by the memory footprint of individual models. Therefore, this hybrid scheme is likely to allow for greater concurrency, enabling more models to share the available secure memory efficiently. Of course, our observations are incomplete, and it is essential to consider other factors, such as the specifics of the target model and other potential sources of concurrency-based contention. We leave such explorations for future work.

Lastly, for the layers that can fit both the output and weights in secure memory, channel and y-plane partitioning are comparable in terms of latency and number of page evictions. Further experiments showed that, for these intermediate layers, the slight difference between both schemes is due to the GEMM (matrix multiplication) implementation. When using standard GEMM libraries such as OpenBLAS [XQC12], this difference disappeared. Thus, we do not claim that one scheme is supperior to the other for layers that fit in secure memory.

			Compression Ratio				
Accuracy	**Base.**	**Quant.**	**32:10**	**32:5**	**32:4**	**32:3**	**32:2**
Top-1 (%)	70.4	70.4	70.4	70.2	68.1	68.5	26.7
Top-5 (%)	89.8	89.8	89.8	89.8	89.1	89.0	53.6

Table 3.4: **Model Accuracy with Quantization and Compression.** A compression ratio of 32:10 means that a buffer of 32 bytes is compressed into 10 bytes. We omit ratios from 32:9 to 32:6 as they produced the same results as ratio 32:10.

3.6 The Decryption Bottleneck

Partitioning alleviates the page thrashing bottleneck. Without thrashing, the transfer of model parameters into the enclave becomes the dominating performance factor due to the overhead of page decryption and integrity checking. This issue, which we call the *decryption bottleneck*, is especially problematic for fully-connected layers with large weight matrices. We explore quantization and compression as possible solutions, reducing the number of pages that need to be transferred.

3.6.1 Characterization

For ease of exposition, we refer to the collection of components that handle secure paging—i.e., the eviction, encryption/decryption, and integrity checking of pages—as the *decryption link*.

In fully-connected layers, loading the weights into secure memory is expensive. For example, in our experiments, we observed that the first fully-connected layer of VGG-16 (i.e., layer 19) took 0.028 seconds to execute with Darknet normally, but 1.131 seconds ($\sim$40X) to execute with Darknet when run inside a trusted execution environment. Our experiments show that the difference in execution time was due entirely to the additional 1.102 seconds needed for loading in the weights from main memory—the 0.028 seconds needed for the layer computations was trivial by comparison.

Layer	Input (MB)	28MB+1vCPU Latency (s)	28MB+1vCPU Evictions (#)	56MB+2vCPU Latency (s)	56MB+2vCPU Evictions (#)	112MB+4vCPU Latency (s)	112MB+4vCPU Evictions (#)	168MB+8vCPU Latency (s)	168MB+8vCPU Evictions (#)
Outside TEE									
19	392	0.030	-	0.029	-	0.030	-	0.029	-
21	64	0.005	-	0.005	-	0.005	-	0.005	-
23	16	0.001	-	0.001	-	0.001	-	0.001	-
Quant./Comp. in TEE									
19	392	0.542	55,707	0.498	55,258	0.497	16,167	0.464	16,168
21	64	0.090	9,194	0.094	11,157	0.076	2,660	0.075	2,660
23	16	0.023	2,090	0.020	2,500	0.020	715	0.022	715
Unmodified in TEE									
19	392	0.987	101,113	1.158	102,328	1.213	101,270	1.124	103,536
21	64	0.162	16,481	0.191	16,516	0.203	16,458	0.185	16,523
23	16	0.039	4,090	0.046	4,165	0.050	4,030	0.045	4,041

Table 3.5: **Latency for Fully-Connected Layers using Quantization and Compression.** Numbers averaged over 30 runs.

Assuming we cannot modify the hardware to improve secure paging performance, and because the decryption link is already saturated, we turn toward techniques to *reduce the amount of data* that must be transferred over that link. Specifically, we analyze the use of two techniques, quantization and compression, to reduce the size of the weights for fully-connected layers. Further, as these techniques require additional computation, multi-threading can be used to keep the decryption link saturated.

Quantization is the process of converting the set of possible weight values (e.g., 32-bit floats) into a smaller discrete set of values (e.g., 16-bit floats). Some information is lost in this conversion, potentially affecting the model's accuracy, but the total memory requirements are halved. The weights are stored quantized and are converted back to 32-bit floats once decrypted. The cost of converting the values back to 32-bits floats was negligible in our experiments.

Similarly, compression also reduces data transfer requirements. We only consider lossy compression here as the compression factor for lossless compression was too small to be useful in our experiments. The amount of information lost is directly related to the compression factor, which can be tuned for many compression algorithms. The computational cost of decompression is higher than quantization, but the workload can be split more easily between virtual CPUs.

3.6.2 Performance of Quantization and Compression

Table 3.5 shows the execution latency for fully-connected layers. For the 28MB+1vCPU and 56MB+2vCPU enclave configurations, we observe roughly half as many page evictions as unmodified Darknet, and execution took roughly half of the time. This performance difference is due to the quantization scheme, which halves the size of the weight matrix. In separate experiments, we observed that adding more than two threads failed to yield further improvement for quantization, suggesting that two threads are sufficient to

saturate the decryption link.

Compression benefits more than quantization from the larger number of virtual CPUs offered by the 112MB+4vCPU and 168MB+8vCPU configurations. When using compression, the number of page evictions decreased to roughly 16% of unmodified Darknet. Once the decryption link saturated with 6 threads, the compression scheme proved more efficient than quantization in these enclaves.

Lastly, we observe no drop in accuracy from quantization, as shown in Table 3.4. Results will vary by model, and the impact of quantization on accuracy is an active area of research in the AI community [Kri18, ZYG+17, DR95]. More aggressive quantization strategies could yield even higher performance. For compression in all but the most extreme compression rate, the top-1 accuracy was within 2% of baseline and the top-5 accuracy was within 0.8%.

3.7 Summary

In summary, we studied the use of partitioning, quantization, and compression to improve the memory efficiency of deep learning inference in trusted execution environments. Partitioning addresses the page thrashing bottleneck, and a combination of the proposed y-plane partitioning scheme and channel partitioning allows for the smallest memory footprint. Quantization and compression reduce the impact of the decryption bottleneck with little impact on model accuracy.

Chapter 4

Conclusions

We proposed a data-free model extraction attack that *(i)* does not require any prior data nor knowledge about the training data distribution, and *(ii)* is able to extract a student model that achieves high accuracy on CIFAR10 and SVHN. This attack was made possible by using a well-chosen loss function and optimizer, as well as a method for approximating the logits out of the probability vector.

This work has few limitations such as the high query budget of the attack. However, additional efforts should lower this query budget and we leave such optimizations for future work. More importantly, the cost of hyperparameter tuning is not included in the query budget and can be quite high due to the sensitivity of the attack to the choice of hyperparameters. We leave a detailed study of hyperparameter tuning for future work.

The second part of this study takes place in a context of remote deep learning inference on untrusted hardware, equiped with a Trusted Execution Environment (TEE). By analyzing two main performance bottleneck for deep learning inference in a TEE, we proposed two mitigations based on partitioning, quantization, and compression. While partitioning was already proposed in previous work as a solution to the page thrashing bottleneck, the new y-plane partitioning described in this works complementarily with pre-existing

channel partitioning. By using both partitioning methods together, the memory footprint of individual models is reduced. Such optimization is useful for *(i)* running large deep learning models in very small TEEs or *(ii)* allowing more models to be computed with the same amount of memory in a concurrent execution setting.

The primary limitation of Chapter 3 is the limited number of models we consider. While convolution and fully-connected are common to a wide range of deep learning models, the benefits of the aforementioned optimizations depend on model specifics. For example, partitioning will not reduce the inference latency for the layers that already fit in memory. Even so, partitioning allows for a configurable memory footprint. This configurability is especially important in the context of concurrent inference, i.e., multiple enclaves running on a single server. We believe a full study of partitioning and model concurrency is an interesting direction for future work.

It is also important to consider other hardware capabilities when configuring the optimizations, such as secure memory size and decryption speed. For example, one would need to adjust the size of each partition to ensure they fit within secure memory. In the current implementation, a partition can be as small as a single y-plane, which for VGG-16, is at most a few hundred kilobytes. As another example, one might also want to tune the compression and quantization factors based on the decryption speed and available CPU resources.

Appendix A

Appendix

A.1 How Hard is it to Find a Surrogate Dataset?

To motivate the need for data-free approaches to model extraction, we show here that an adversary relying on a surrogate dataset must ensure that its distribution is close to the one of the victim's training set. Otherwise, model extraction will return a poor approximation of the victim.

Consider a *victim* machine learning model, $\mathcal{V}$, trained on a proprietary dataset, $\mathcal{D}_{\mathcal{V}}$. The victim model reveals its predictions through either a prediction API (as is common in MLaaS) or through the deployment of the model on devices accessible to adversaries. The adversary, $\mathcal{A}$, attempts to steal $\mathcal{V}$ by querying it with a surrogate dataset, $\mathcal{D}_{\mathcal{S}}$. This surrogate dataset is assumed to be publicly available or easier to access because it does not need to be labeled.

We now perform a systematic study of the features that characterize the *closeness* of $\mathcal{D}_{\mathcal{S}}$ when compared to $\mathcal{D}_{\mathcal{V}}$. Let the private and surrogate datasets $\mathcal{D}_{\mathcal{V}}$ and $\mathcal{D}_{\mathcal{S}}$ be characterized by $\mathcal{D} = \{\mathcal{X}, P(X), \mathcal{Y}, P(Y|X)\}$ [Rud17]. The private and surrogate datasets can vary in three ways, which we will illustrate in the following with object recognition tasks:

	Victim	CIFAR10	CIFAR100	SVHN	MNIST	SVHN_{skew}	Random
CIFAR10	95.5%	95.2%	93.5%	66.6%	37.2%	-	10.0%
SVHN	96.2%	96.0%	-	96.3%	89.5%	96.1%	84.1%

Table A.1: Model Extraction accuracy across various surrogate datasets. Victim models were trained on the CIFAR10 and SVHN datasets, and the source accuracies are reported under the heading 'Victim'

1. **A:** $(\mathcal{X}_{\mathcal{V}} \neq \mathcal{X}_{\mathcal{S}})$. When the inputs of $\mathcal{D}_{\mathcal{V}}$ and $\mathcal{D}_{\mathcal{S}}$ belong to different feature spaces (i.e. domain). In computer vision, for example, this can be a scenario wherein the input data (e.g., images, videos) for the datasets contain a different number of channels or pixels.

2. **B:** $(P(X_{\mathcal{V}}) \neq P(X_{\mathcal{S}}))$. While the input domain of both the surrogate and private datasets is the same, their marginal probability distribution is different. For example, when the semantic nature is different for the two datasets (images of animals, digits, etc).

3. **C:** $(P(Y_{\mathcal{V}}|X_{\mathcal{V}}) \neq P(Y_{\mathcal{S}}|X_{\mathcal{S}}))$. We consider a setting where the semantic distribution $(P(X))$ is the same, but the class-conditional probability distributions of the victim and surrogate training sets are different, e.g. when the two datasets have class imbalance.

A.1.1 Optimization Problem

The logit distribution of the victim network can often have a strong affinity toward the true label. To address this issue, Hinton et al. suggested scaling the logits to make the probability distributions more informative [HVD15].

$$\mathcal{V}_i(x) = \frac{\exp\left(v_i(x)/\tau\right)}{\sum_j \exp\left(v_j(x)/\tau\right)}; \quad \mathcal{L} = -\tau^2 \text{KL}\left(\mathcal{V}(x), \mathcal{S}(x)\right)$$

where $\tau > 1$ is referred to as the temperature scaling parameter. In the knowledge distillation literature, a combination of both the cross entropy and knowledge distillation loss are used to query the teacher [CH19]. However, model extraction we rely only on the KL divergence loss because the queries are made from a surrogate dataset which may not have any semantic binding to the true class.

A.1.2 Experimental Setting

Hyperparameters To search over a meaningful hyperparameter space for the temperature co-efficient τ, we refer to prior work in knowledge distillation such as [CH19, HVD15, LZG18] to confine the search over $\tau \in \{1, 3, 5, 10\}$. We used the SGD optimizer, and train the CIFAR10 students for 100 epochs, while the SVHN students were trained for 50 epochs. We experimented with two different learning schedules: (1) cyclic learning rate [Smi17]; and (2) step-decay learning rate. For step-decay, we reduced the learning rate by a factor of 0.2 at 30%, 60%, and 80% of the training process. In both cases, the maximum learning rate was set to 0.1.

Experimental validation. To illustrate our argument, we next detail the relation between a task-specific surrogate dataset and the accuracy of state-of-the-art model extraction techniques. The victim models under attack are ResNet-18-8x models, their accuracy is reported in Table A.1. Further details on the victim models training are provided in Chapter 2.5). We find that querying from the original dataset yields the most query-efficient and accurate extraction results. This is not surprising given that this setup corresponds to the original knowledge distillation setting. Our observations are made on both CIFAR10 and SVHN:

- **CIFAR10.** We benchmarked model extraction on 4 surrogate datasets, each reflecting a different property detailed above: CIFAR10 [Kri09], CIFAR100 [Kri09]

(BC), SVHN [NWC+11] (AB) and MNIST [LBBH98] (AB). To ensure a fair comparison, we bound the maximum number of distinct samples queried by 50,000 while performing model extraction.

- **SVHN.** We evaluated model extraction by querying from SVHN, SVHN$_{skew}$ (C), MNIST (A) and CIFAR10 (B) as surrogate datasets. Similar to the case for CIFAR10, we cap the maximum number of distinct samples queried to 50,000.

Finally, as a control for our experiments, we also studied the extraction accuracy of the models when trained using totally random queries.

Dataset adaptation. The SVHN, CIFAR10, and CIFAR100 datasets contain 32×32 color images. To query networks trained on CIFAR10 and SVHN with images from the MNIST dataset, which contains grayscale images of size 28×28, we re-scaled the image and repeated the same input across all three RGB channels. In case of random input generation, we sample input tensors from a normal distribution with mean 0 and variance 1. Note that the teacher networks were trained on normalized datasets in the first place. Finally, in case of SVHN$_{skew}$, we supplied images from only the first 5 classes of the dataset to skew the distribution of the modified dataset.

Results. We present the results for accuracy of extracted models across various surrogate datasets for CIFAR10 and SVHN in Table A.1. Recall that both the CIFAR10 and CIFAR100 datasets are subsets from the same TinyImages [TFF08] dataset. We find that the identical source distribution was extremely useful in making relevant queries to the CIFAR10 teacher. The accuracy of the extracted model reached 93.5%, just below the 95.5% accuracy of the teacher model. However, when we used the SVHN surrogate dataset to query the CIFAR10 teacher, with a different source distribution, the model extraction performance dropped remarkably, attaining a maximum of 66.6% across all of the

hyperparameters tried. In the most extreme scenario when querying the CIFAR10 teacher with MNIST–a dataset with disjoint feature space both in terms of number of pixels, and number of channels)—model accuracy did not improve beyond 37.2%.

On the contrary, we notice that the victim trained on the SVHN dataset is much easier for the adversary to extract. Surprisingly, even when the victim is queried with completely random inputs, the extracted model attains an accuracy of over 84% on the original SVHN test set. Further, nearly all surrogate datasets are able to achieve greater than 90% accuracy on the test set. We hypothesize that this observation is linked to how the digit classification task, at the root of SVHN, is a simpler task for neural networks to solve, and the underlying representations (hence, not being as complex as for CIFAR10) can be learnt even when queried over random inputs.

Given the current understanding of model extraction, we make two conclusions: (1) the success of model extraction largely depends on the complexity of the task that the victim model aims to solve; and (2) similarity to source domain is critical for extracting machine learning models that solve complex tasks. We posit that it may be nearly as expensive for the adversary to extract a CIFAR10 machine learning model with a good surrogate dataset, as is training from scratch. A weaker or non-task specific dataset may have lesser costs, but has high accuracy trade-offs.

A.2 Recovering logits from probabilities

The main difficulty with computing $\mathcal{L}_{\ell_1}$ is that it requires access to $\mathcal{V}$'s logits v_i, but we only have access to the probabilities of each class (i.e., after the softmax is applied to the logits). In a first approximation, the logits can be recovered by computing the log-probabilities but the resulting approximate logits are computed up to an additive constant $C(x)$ to which we don't have access in a black-box setting. This additive constant is the

same for all logits but is different from one image to another. Related works on adversarial examples [BHLS18, CZS+17] use losses that are the difference of two logits, effectively canceling out the additive constant. In our case, the logits need to be used individually which makes the ℓ_1 loss more difficult to compute in our setting.

To overcome this issue, we propose to approximate the true logits of each image x in two steps. First, compute the logarithm of the probability vector $V(x)$.

$$\tilde{v}_i(x) = \log \mathcal{V}_i(x) = v_i + C(x) \tag{A.1}$$

Then, compute the approximate true logits $v_i^*(x)$ by subtracting the log-probability vector with its own mean:

$$\begin{aligned} v_i^*(x) &= \tilde{v}_i(x) - \frac{1}{K}\sum_{j=1}^{K} \tilde{v}_j(x) \\ &= v_i(x) - \frac{1}{K}\sum_{j=1}^{K} v_j(x) \approx v_i(x) \end{aligned} \tag{A.2}$$

The second equality holds because the mean of the log-probability vector $\tilde{v}_i(x)$ is equal to the mean of the true victim logits $v_i(x)$ plus the mean of the additive constant (i.e. the $C(x)$ itself). By analyzing the mean values of the true logits from various pre-trained models—which proves to be negligible in comparison to the logit values themselves, we provide empirical evidence in Chapter 2.6.3 that this recovers a highly accurate approximation of the true logits $v_i^*(x)$.

A.3 Examples of Synthetic Images

Figure A.1 shows 4 images from the generator towards the end of the attack on CIFAR-10. We do not observe any similarities with the images from the original training dataset.

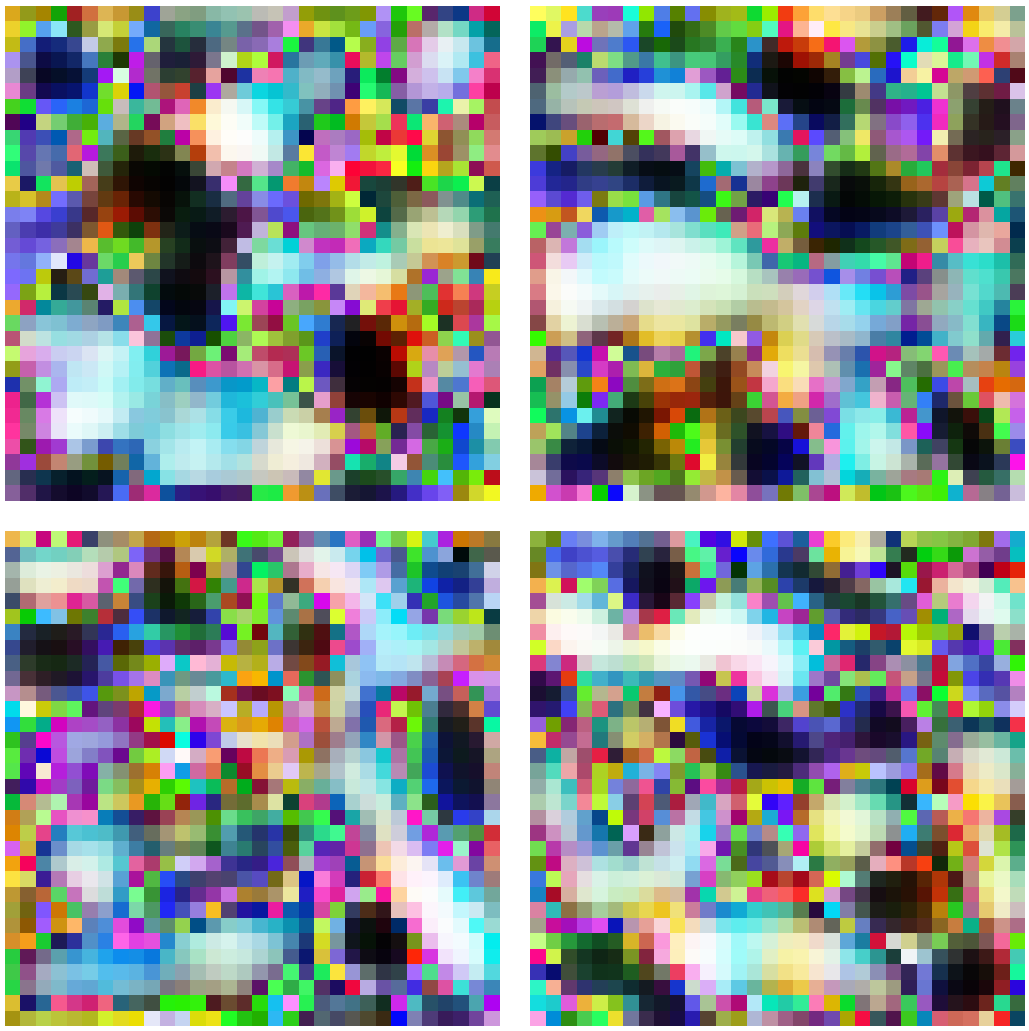

Figure A.1: Four synthetic images from the generator.

A.4 Hypothesis 1: Justification

A.4.1 Preliminary results

Lemma 1. *If $\mathcal{S}(x) \in (0,1)^K$ is the softmax output of a differentiable function (e.g. a neural network) on an input x and s is the corresponding logits vector, then the Jacobian matrix $J = \frac{\partial \mathcal{S}}{\partial s}$ has an eigenvalue decomposition and all its eigenvalues are in the interval [0, 1].*

Proof. By definition:

$$\forall i \in \{1 \dots K\},\ \mathcal{S}_i = \frac{\exp(s_i)}{\sum_{k=1}^{K} \exp(s_k)}$$

For some $i, j \in \{1 \dots K\}$, if $i \neq j$:

$$\begin{aligned}\frac{\partial \mathcal{S}_i}{\partial s_j} &= -\exp(s_j)\frac{\exp(s_i)}{\left(\sum_{k=1}^{K} \exp(s_k)\right)^2} \\ &= -\mathcal{S}_i \mathcal{S}_j\end{aligned}$$

if $i = j$:

$$\begin{aligned}\frac{\partial \mathcal{S}_i}{\partial s_j} &= \frac{\exp(s_i)(\sum_{k=1}^{K} \exp(s_k)) - \exp(s_j)\exp(s_i)}{\left(\sum_{k=1}^{K} \exp(s_k)\right)^2} \\ &= \frac{\exp(s_i)}{\sum_{k=1}^{K} \exp(s_k)} - \frac{\exp(s_i)^2}{(\sum_{k=1}^{K} \exp(s_k))^2} \\ &= \mathcal{S}_i(1 - \mathcal{S}_i)\end{aligned}$$

Therefore, $\forall x$,

$$J = \frac{\partial \mathcal{S}}{\partial s} = \begin{bmatrix} \mathcal{S}_1(1-\mathcal{S}_1) & -\mathcal{S}_1\mathcal{S}_2 & \dots & -\mathcal{S}_1\mathcal{S}_K \\ -\mathcal{S}_1\mathcal{S}_2 & \mathcal{S}_2(1-\mathcal{S}_2) & \dots & -\mathcal{S}_2\mathcal{S}_K \\ \vdots & \vdots & & \vdots \\ -\mathcal{S}_1\mathcal{S}_K & -\mathcal{S}_2\mathcal{S}_K & \dots & \mathcal{S}_K(1-\mathcal{S}_K) \end{bmatrix}$$

The matrix J is real-valued symmetric, therefore it has an eigen-decomposition with real eigenvalues. $\exists \lambda_1, \lambda_2, \dots, \lambda_K \in \mathbb{R}, X_1, X_2, \dots, X_K \neq 0$ such that:

$$\forall i \in \{1 \dots K\}, JX_i = \lambda_i X_i$$

Let us prove that all eigenvalues are in the interval [0, 1]. Suppose for a contradiction that one eigenvalue λ is strictly negative. Let the associated eigenvector be:

$$X = [x_1, x_2, \dots, x_K]^T$$

The i-th component of the vector JX is:

$$\begin{aligned}[JX]_i &= \mathcal{S}_i x_i - \mathcal{S}_i \sum_{k=1}^{K} x_k \mathcal{S}_k \\ &= \mathcal{S}_i x_i - \mathcal{S}_i \langle X, \mathcal{S} \rangle\end{aligned}$$

where $\langle \cdot, \cdot \rangle$ is the standard inner product.

Since X is an eigenvector we have,

$$JX = \lambda X$$

So $\forall\ i$,

$$\begin{aligned}\mathcal{S}_i x_i - \mathcal{S}_i \langle X, \mathcal{S} \rangle &= \lambda x_i \\ x_i(\mathcal{S}_i - \lambda) &= \mathcal{S}_i \langle X, \mathcal{S} \rangle\end{aligned}$$

Since $X \neq 0$, $\exists\ i_0$ such that $x_{i_0} \neq 0$. Furthermore, λ is strictly negative so:

$$x_{i_0}(\mathcal{S}_{i_0} - \lambda) = \mathcal{S}_{i_0} \langle X, \mathcal{S} \rangle \neq 0$$

Therefore, the inner product on the right hand side is non-zero.

In addition, $\lambda < 0$ implies that $\mathcal{S}_i - \lambda > \mathcal{S}_i > 0$ so $\forall\ i, x_i$ and $\langle X, \mathcal{S} \rangle$ have the same sign. There are two cases left.

If $\langle X, \mathcal{S} \rangle > 0$, then $\forall\ i, x_i > 0$ and:

$$x_i(\mathcal{S}_i - \lambda) > x_i\mathcal{S}_i$$
$$\mathcal{S}_i\langle X, \mathcal{S} \rangle > x_i\mathcal{S}_i$$

By summing on all i we obtain:

$$\sum_{i=1}^{K} \mathcal{S}_i\langle X, \mathcal{S} \rangle > \sum_{i=1}^{K} x_i\mathcal{S}_i$$
$$\langle X, \mathcal{S} \rangle \sum_{i=1}^{K} \mathcal{S}_i > \langle X, \mathcal{S} \rangle$$
$$\langle X, \mathcal{S} \rangle > \langle X, \mathcal{S} \rangle$$

Which is an **absurdity**.

If $\langle X, \mathcal{S} \rangle < 0$, then $\forall\ i, x_i < 0$ and:

$$x_i(\mathcal{S}_i - \lambda) < x_i\mathcal{S}_i$$
$$\mathcal{S}_i\langle X, \mathcal{S} \rangle < x_i\mathcal{S}_i$$

The same summation and reasoning yields an **absurdity**. We just proved that all the eigenvalues of J are non-negative.

Lastly, the trace of the Jacobian matrix $tr(J)$ equals the sum of all eigenvalues. Com-

puting the trace yields:

$$\begin{aligned} tr(J) = \sum_{i=1}^{K} \lambda_i &= \sum_{i=1}^{K} \mathcal{S}_i(1 - \mathcal{S}_i) \\ &= \sum_{i=1}^{K} \mathcal{S}_i - \sum_{i=1}^{K} \mathcal{S}_i^2 \\ &= 1 - \sum_{i=1}^{K} \mathcal{S}_i^2 < 1 \end{aligned}$$

Since $\lambda_i \geq 0$ and $\sum_{i=1}^{K} \lambda_i < 1$, all eigenvalues must be in the interval [0, 1], which concludes the proof.

□

Lemma 2. *In the same setting as for Lemma 1, if J is the Jacobian matrix $\frac{\partial \mathcal{S}}{\partial s}$ then for any vector Z we have:*

$$\|JZ\| \leq \|Z\|$$

Proof. Let $\lambda_1, \lambda_2, \ldots, \lambda_K$ be the eigenvalues of J and $X_1, X_2, \ldots, X_K$ be the associated eigenvectors. We can decompose Z with the orthonormal eigenvector basis:

$$Z = \sum_{i=1}^{K} \alpha_i X_i$$

Computing the product JZ yields:

$$JZ = \sum_{i=1}^{K} \lambda_i \alpha_i X_i$$

The norm of the product is:

$$\begin{aligned}\|JZ\| = (JZ)^T(JZ) &= \sum_{i=1}^{K} \lambda_i^2 \alpha_i^2 \\ &\leq \sum_{i=1}^{K} \alpha_i^2\end{aligned}$$

because $\forall\ i, |\lambda_i| \leq 1$ (see Lemma 1). Since the eigenvector basis is orthonormal we have

$$\|JZ\| \leq \sum_{i=1}^{K} \alpha_i^2 = \|Z\|$$

□

Lemma 3. *Let $\mathcal{S}(x)$ and $\mathcal{V}(x)$ be the softmax output of two differentiable functions (e.g. neural networks) on an input x, with respective logits $s(x)$ and $v(x)$. When $\mathcal{S}$ converges to $\mathcal{V}$, then $\frac{\partial \mathcal{S}}{\partial s}$ converges to $\frac{\partial \mathcal{V}}{\partial v}$.*

Proof. Recall that

$$\frac{\partial \mathcal{S}}{\partial s} = \begin{bmatrix} \mathcal{S}_1(1-\mathcal{S}_1) & -\mathcal{S}_1\mathcal{S}_2 & \dots & -\mathcal{S}_1\mathcal{S}_K \\ -\mathcal{S}_1\mathcal{S}_2 & \mathcal{S}_2(1-\mathcal{S}_2) & \dots & -\mathcal{S}_2\mathcal{S}_K \\ \vdots & \vdots & & \vdots \\ -\mathcal{S}_1\mathcal{S}_K & -\mathcal{S}_2\mathcal{S}_K & \dots & \mathcal{S}_K(1-\mathcal{S}_K) \end{bmatrix}$$

If $\mathcal{V}_i(x) - \mathcal{S}_i(x) = \epsilon_i(x)$, then:

$$\begin{aligned}\mathcal{S}_i(1-\mathcal{S}_i) &= (\mathcal{V}_i + \epsilon_i)(1 - \mathcal{V}_i - \epsilon_i) \\ &= \mathcal{V}_i(1-\mathcal{V}_i) + \epsilon_i(1-\mathcal{V}_i) - \epsilon_i^2 \\ &= \mathcal{V}_i(1-\mathcal{V}_i) + o(1)\end{aligned}$$

and

$$\begin{aligned} -\mathcal{S}_i\mathcal{S}_j &= -(\mathcal{V}_i + \epsilon_i)(\mathcal{V}_j + \epsilon_j) \\ &= -\mathcal{V}_i\mathcal{V}_j - \mathcal{V}_i\epsilon_j - \mathcal{V}_j\epsilon_i - \epsilon_i\epsilon_j \\ &= -\mathcal{V}_i\mathcal{V}_j + o(1) \end{aligned}$$

Therefore, we can write:

$$\frac{\partial \mathcal{S}}{\partial s} = \frac{\partial \mathcal{V}}{\partial v} + \bar{\epsilon}(x)$$

where $\bar{\epsilon}(x)$ converges to the null matrix as $\mathcal{S}$ converges to $\mathcal{V}$. In other words we can write:

$$\frac{\partial \mathcal{S}}{\partial s} \underset{\mathcal{S}\to\mathcal{V}}{\approx} \frac{\partial \mathcal{V}}{\partial v}$$

□

A.4.2 Justification of the hypothesis.

Hypothesis 1 states that for two differentiable functions with softmax output $\mathcal{S}$ and $\mathcal{V}$, and respective logits s and v, the gradients of the KL divergence loss $\mathcal{L}_{KL}$ with respect to the input should be small compared to the gradients of the ℓ_1 norm loss $\mathcal{L}_{\ell_1}$ as $\mathcal{S}$ converges to $\mathcal{V}$. $\forall x \in [-1, 1]^d$:

$$\|\nabla_x \mathcal{L}_{\text{KL}}(x)\| \underset{\mathcal{S}\to\mathcal{V}}{\ll} \|\nabla_x \mathcal{L}_{\ell_1}(x)\|$$

Proof. First, we note that:

$$\sum_{i=1}^{K} \mathcal{S}_i = 1$$

implies

$$\sum_{i=1}^{K} \frac{\partial \mathcal{S}_i}{\partial x} = \mathbf{0}$$

And the same holds for $\mathcal{V}$ because both are probability distributions.

Then we compute the gradients for both loss functions:

For the ℓ_1 norm loss:

$$\nabla_x \mathcal{L}_{\ell_1}(x) = \sum_{i=1}^{K} sign(v_i - s_i) \left(\frac{\partial v_i}{\partial x} - \frac{\partial s_i}{\partial x} \right)$$

For the KL divergence loss:

$$\begin{aligned}
\nabla_x \mathcal{L}_{\text{KL}}(x) &= \sum_{i=1}^{K} \frac{\partial \mathcal{V}_i}{\partial x} \log \mathcal{V}_i + 1 \frac{\partial \mathcal{V}_i}{\partial x} - \frac{\partial \mathcal{V}_i}{\partial x} \log \mathcal{S}_i - \frac{\partial \mathcal{S}_i}{\partial x} \frac{\mathcal{V}_i}{\mathcal{S}_i} \\
&= \sum_{i=1}^{K} \frac{\partial \mathcal{V}_i}{\partial x} + \sum_{i=1}^{K} \frac{\partial \mathcal{V}_i}{\partial x} \log \frac{\mathcal{V}_i}{\mathcal{S}_i} - \frac{\partial \mathcal{S}_i}{\partial x} \frac{\mathcal{V}_i}{\mathcal{S}_i} \\
&= \sum_{i=1}^{K} \frac{\partial \mathcal{V}_i}{\partial x} \log \frac{\mathcal{V}_i}{\mathcal{S}_i} - \frac{\partial \mathcal{S}_i}{\partial x} \frac{\mathcal{V}_i}{\mathcal{S}_i}
\end{aligned}$$

When $\mathcal{S}$ converges to $\mathcal{V}$, we can write

$$\mathcal{V}_i(x) = \mathcal{S}_i(x)(1 + \delta_i(x))$$

where $\delta_i(x) \underset{\mathcal{S} \to \mathcal{V}}{\to} 0$.

Since $\log 1 + x \approx x$ when x is close to 0 we can write:

$$\begin{aligned}
\nabla_x \mathcal{L}_{\text{KL}}(x) &= \sum_{i=1}^{K} \frac{\partial \mathcal{V}_i}{\partial x} \log \frac{\mathcal{V}_i}{\mathcal{S}_i} - \frac{\partial \mathcal{S}_i}{\partial x} \frac{\mathcal{V}_i}{\mathcal{S}_i} \\
&\approx \sum_{i=1}^{K} \frac{\partial \mathcal{V}_i}{\partial x} \delta_i - \frac{\partial \mathcal{S}_i}{\partial x} (1 + \delta_i) \\
&\approx \sum_{i=1}^{K} \delta_i \left(\frac{\partial \mathcal{V}_i}{\partial x} - \frac{\partial \mathcal{S}_i}{\partial x} \right) + \sum_{i=1}^{K} \frac{\partial \mathcal{S}_i}{\partial x} \\
&\approx \sum_{i=1}^{K} \delta_i \left(\frac{\partial \mathcal{V}_i}{\partial x} - \frac{\partial \mathcal{S}_i}{\partial x} \right) \\
&\approx \sum_{i=1}^{K} \delta_i \frac{\partial \mathcal{V}}{\partial v} \left(\frac{\partial v_i}{\partial x} - \frac{\partial s_i}{\partial x} \right) \text{ (Lemma 3)} \\
&\approx \frac{\partial \mathcal{V}}{\partial v} \sum_{i=1}^{K} \delta_i \left(\frac{\partial v_i}{\partial x} - \frac{\partial s_i}{\partial x} \right)
\end{aligned}$$

Using Lemma 2, the norm is upper bounded by:

$$\|\nabla_x \mathcal{L}_{\text{KL}}(x)\| \leq \left\| \sum_{i=1}^{K} \delta_i \left(\frac{\partial v_i}{\partial x} - \frac{\partial s_i}{\partial x} \right) \right\| \tag{A.3}$$

For the ℓ_1 norm, however, the norm is:

$$\|\nabla_x \mathcal{L}_{\ell_1}(x)\| = \left\| \sum_{i=1}^{K} sign(v_i - s_i) \left(\frac{\partial v_i}{\partial x} - \frac{\partial s_i}{\partial x} \right) \right\| \tag{A.4}$$

From equation A.3, we can observe that each term is negligible compared to its counterpart in equation A.4: for all i we have:

$$\left\| \delta_i \left(\frac{\partial v_i}{\partial x} - \frac{\partial s_i}{\partial x} \right) \right\| \leq \epsilon \left\| \left(\frac{\partial v_i}{\partial x} - \frac{\partial s_i}{\partial x} \right) \right\|$$

And also $\forall i$:

$$\left\| sign(v_i - s_i) \left(\frac{\partial v_i}{\partial x} - \frac{\partial s_i}{\partial x} \right) \right\| = \left\| \left(\frac{\partial v_i}{\partial x} - \frac{\partial s_i}{\partial x} \right) \right\|$$

Therefore, by summing these terms on the index i we can expect the KL divergence gradient to be small in magnitude compared to those of the ℓ_1 norm. However, it does not seem possible to prove this result rigorously without further assumptions on the data distribution or the mode of convergence of $\mathcal{S}$.

□

Bibliography

[AB17] Martin Arjovsky and Lon Bottou. Towards principled methods for training generative adversarial networks, 2017.

[ACB17] Martin Arjovsky, Soumith Chintala, and Lon Bottou. Wasserstein gan, 2017.

[BC14] Jimmy Ba and Rich Caruana. Do deep nets really need to be deep? In *Advances in neural information processing systems*, pages 2654–2662, 2014.

[BHLS18] Arjun Nitin Bhagoji, Warren He, Bo Li, and Dawn Song. Practical black-box attacks on deep neural networks using efficient query mechanisms. In *European Conference on Computer Vision*, pages 158–174. Springer, 2018.

[BMR+20] Tom B Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*, 2020.

[CCEKL20] Yoojin Choi, Jihwan Choi, Mostafa El-Khamy, and Jungwon Lee. Data-free network quantization with adversarial knowledge distillation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, pages 710–711, 2020.

[CCG+18] V. Chandrasekaran, K. Chaudhuri, I. Giacomelli, S. Jha, and Songbai Yan. Model extraction and active learning. *ArXiv*, abs/1811.02054, 2018.

[CCG+19] Varun Chandrasekaran, Kamalika Chaudhuri, Irene Giacomelli, Somesh Jha, and Songbai Yan. Exploring connections between active learning and model extraction, 2019.

[CH19] Jang Hyun Cho and Bharath Hariharan. On the efficacy of knowledge distillation, 2019.

[CSBB+18] Jacson Rodrigues Correia-Silva, Rodrigo F. Berriel, Claudine Badue, Alberto F. de Souza, and Thiago Oliveira-Santos. Copycat cnn: Stealing knowledge by persuading confession with random non-labeled data. *2018 International Joint Conference on Neural Networks (IJCNN)*, Jul 2018.

[CZS+17] Pin-Yu Chen, Huan Zhang, Yash Sharma, Jinfeng Yi, and Cho-Jui Hsieh. Zoo: Zeroth order optimization based black-box attacks to deep neural networks without training substitute models. In *Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security*, pages 15–26, 2017.

[DBK+20] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020.

[DDS+09] J. Deng, W. Dong, R. Socher, L. Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 248–255, 2009.

[Den68] Peter J. Denning. Thrashing: Its causes and prevention. In *Proceedings of the December 9-11, 1968, Fall Joint Computer Conference, Part I*, AFIPS '68 (Fall, part I), page 915922, New York, NY, USA, 1968. Association for Computing Machinery.

[DR95] Gunhan Dundar and Kenneth Rose. The effects of quantization on multilayer neural networks. *IEEE Transactions on Neural Networks*, 1995.

[FSS+19] Gongfan Fang, Jie Song, Chengchao Shen, Xinchao Wang, Da Chen, and Mingli Song. Data-free adversarial distillation. *arXiv preprint arXiv:1912.11006*, 2019.

[GBDL+16] Ran Gilad-Bachrach, Nathan Dowlin, Kim Laine, Kristin Lauter, Michael Naehrig, and John Wernsing. Cryptonets: Applying neural networks to encrypted data with high throughput and accuracy. In *International Conference on Machine Learning (ICML)*, 2016.

[Gor18] Jim Gordon. Microsoft azure confidential computing with intel sgx, 2018.

[GPAM+14] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks, 2014.

[GYMT20] Jianping Gou, Baosheng Yu, Stephen John Maybank, and Dacheng Tao. Knowledge distillation: A survey, 2020.

[HNP09] Alon Halevy, Peter Norvig, and Fernando Pereira. The unreasonable effectiveness of data. *IEEE Intelligent Systems*, 24:8–12, 2009.

[HSS+18] Tyler Hunt, Congzheng Song, Reza Shokri, Vitaly Shmatikov, and Emmett Witchel. Chiron: Privacy-preserving machine learning as a service. *arXiv preprint arXiv:1803.05961*, 2018.

[HVD15] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.

[JCB+20] Matthew Jagielski, Nicholas Carlini, David Berthelot, Alex Kurakin, and Nicolas Papernot. High accuracy and high fidelity extraction of neural networks. In *29th {USENIX} Security Symposium ({USENIX} Security 20)*, 2020.

[KKR+20] Kyungtae Kim, Chung Hwan Kim, Junghwan" John" Rhee, Xiao Yu, Haifeng Chen, Dave Tian, and Byoungyoung Lee. Vessels: efficient and scalable deep learning prediction on trusted processors. In *Proceedings of the 11th ACM Symposium on Cloud Computing*, pages 462–476, 2020.

[KPQ20] Sanjay Kariyappa, Atul Prakash, and Moinuddin Qureshi. Maze: Data-free model stealing attack using zeroth-order gradient estimation, 2020.

[Kri09] Alex Krizhevsky. Learning multiple layers of features from tiny images. Technical report, 2009.

[Kri18] Raghuraman Krishnamoorthi. Quantizing deep convolutional networks for efficient inference: A whitepaper. *arXiv preprint arXiv:1806.08342*, 2018.

[LBBH98] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[LCK+20] Sijia Liu, Pin-Yu Chen, Bhavya Kailkhura, Gaoyuan Zhang, Alfred Hero, and Pramod K. Varshney. A primer on zeroth-order optimization in signal processing and machine learning, 2020.

[LFS17] Raphael Gontijo Lopes, Stefano Fenu, and Thad Starner. Data-free knowledge distillation for deep neural networks, 2017.

[LLP+19] Taegyeong Lee, Zhiqi Lin, Saumay Pushp, Caihua Li, Yunxin Liu, Youngki Lee, Fengyuan Xu, Chenren Xu, Lintao Zhang, and Junehwa Song. Occlumency: Privacy-preserving remote deep-learning inference using sgx. In *The 25th Annual International Conference on Mobile Computing and Networking (MobiCom)*, 2019.

[LM05] Daniel Lowd and Christopher Meek. Adversarial learning. In *Proceedings of the Eleventh ACM SIGKDD International Conference on Knowledge Discovery in Data Mining*, KDD '05, page 641647, New York, NY, USA, 2005. Association for Computing Machinery.

[LZG18] Xu Lan, Xiatian Zhu, and Shaogang Gong. Knowledge distillation by on-the-fly native ensemble, 2018.

[MLS+20] Pratyush Mishra, Ryan Lehmkuhl, Akshayaram Srinivasan, Wenting Zheng, and Raluca Ada Popa. Delphi: A cryptographic inference service for neural networks. In *29th USENIX Security Symposium (USENIX Security)*, 2020.

[MS19] Paul Micaelli and Amos J Storkey. Zero-shot knowledge transfer via adversarial belief matching. In *Advances in Neural Information Processing Systems*, pages 9551–9561, 2019.

[MSDH18] Smitha Milli, Ludwig Schmidt, Anca D. Dragan, and Moritz Hardt. Model reconstruction from model explanations, 2018.

[NS17] Yurii Nesterov and Vladimir Spokoiny. Random gradient-free minimization of convex functions. *Foundations of Computational Mathematics*, 17(2):527–566, 2017.

[NWC+11] Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, and Andrew Y. Ng. Reading digits in natural images with unsupervised feature learning. In *NIPS Workshop on Deep Learning and Unsupervised Feature Learning 2011*, 2011.

[OSF19] Tribhuvanesh Orekondy, Bernt Schiele, and Mario Fritz. Knockoff nets: Stealing functionality of black-box models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.

[PGS+19] Soham Pal, Yash Gupta, Aditya Shukla, Aditya Kanade, Shirish Shevade, and Vinod Ganapathy. A framework for the extraction of deep neural networks by leveraging public data, 2019.

[PMG+17a] Nicolas Papernot, Patrick McDaniel, Ian Goodfellow, Somesh Jha, Z. Berkay Celik, and Ananthram Swami. Practical black-box attacks against machine learning. In *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security*, ASIA CCS '17, page 506519, New York, NY, USA, 2017. Association for Computing Machinery.

[PMG+17b] Nicolas Papernot, Patrick McDaniel, Ian Goodfellow, Somesh Jha, Z Berkay Celik, and Ananthram Swami. Practical black-box attacks against machine learning. In *Proceedings of the 2017 ACM on Asia conference on computer and communications security*, pages 506–519, 2017.

[RBK+15] Adriana Romero, Nicolas Ballas, Samira Ebrahimi Kahou, Antoine Chassang, Carlo Gatta, and Yoshua Bengio. Fitnets: Hints for thin deep nets, 2015.

[Rud17] Sebastian Ruder. Transfer Learning - Machine Learning's Next Frontier. http://ruder.io/transfer-learning/, 2017.

[SGM19] Emma Strubell, Ananya Ganesh, and Andrew McCallum. Energy and policy considerations for deep learning in nlp, 2019.

[SGX15] Intel, Software Guard Extensions (SGX). https://software.intel.com/content/www/us/en/develop/topics/software-guard-extensions.html, 2015.

[SGZ+16] Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen. Improved techniques for training gans. In *Advances in neural information processing systems*, pages 2234–2242, 2016.

[Smi17] Leslie N. Smith. Cyclical learning rates for training neural networks, 2017.

[SZ14] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[SZB+20] Ilia Shumailov, Yiren Zhao, Daniel Bates, Nicolas Papernot, Robert Mullins, and Ross Anderson. Sponge examples: Energy-latency attacks on neural networks. *arXiv preprint arXiv:2006.03463*, 2020.

[TFF08] Antonio Torralba, Rob Fergus, and William T. Freeman. 80 million tiny images: A large data set for nonparametric object and scene recognition. *IEEE Trans. Pattern Anal. Mach. Intell.*, 30(11):19581970, November 2008.

[TGS+18] Shruti Tople, Karan Grover, Shweta Shinde, Ranjita Bhagwan, and Ramachandran Ramjee. Privado: Practical and secure dnn inference. *arXiv preprint arXiv:1810.00602*, 2018.

[TTC+19] Chun-Chen Tu, Paishun Ting, Pin-Yu Chen, Sijia Liu, Huan Zhang, Jinfeng Yi, Cho-Jui Hsieh, and Shin-Ming Cheng. Autozoom: Autoencoder-based zeroth order optimization method for attacking black-box neural networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 742–749, 2019.

[TZJ+16] Florian Tramèr, Fan Zhang, Ari Juels, Michael K Reiter, and Thomas Ristenpart. Stealing machine learning models via prediction apis. In *25th USENIX Security Symposium (USENIX Security)*, 2016.

[WDBS18] Yining Wang, Simon Du, Sivaraman Balakrishnan, and Aarti Singh. Stochastic zeroth-order optimization in high dimensions. In *International Conference on Artificial Intelligence and Statistics*, pages 1356–1365, 2018.

[WWJD12] Andre Wibisono, Martin J Wainwright, Michael Jordan, and John C Duchi. Finite sample convergence rates of zero-order stochastic optimization methods. *Advances in Neural Information Processing Systems*, 25:1439–1447, 2012.

[XQC12] Zhang Xianyi, Wang Qian, and Zaheer Chothia. Openblas. *URL: http://xianyi.github.io/OpenBLAS*, 2012.

[ZK17] Sergey Zagoruyko and Nikos Komodakis. Paying more attention to attention: Improving the performance of convolutional neural networks via attention transfer, 2017.

[ZSG+19] Linfeng Zhang, Jiebo Song, Anni Gao, Jingwei Chen, Chenglong Bao, and Kaisheng Ma. Be your own teacher: Improve the performance of convolutional neural networks via self distillation, 2019.

[ZXHL17] Ying Zhang, Tao Xiang, Timothy M. Hospedales, and Huchuan Lu. Deep mutual learning, 2017.

[ZYG+17] Aojun Zhou, Anbang Yao, Yiwen Guo, Lin Xu, and Yurong Chen. Incremental network quantization: Towards lossless cnns with low-precision weights. In *International Conference on Learning Representations (ICLR)*, 2017.