THE PENNSYLVANIA STATE UNIVERSITY
SCHREYER HONORS COLLEGE

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

# GENERATING CERTIFIABLY ADVERSARIAL ROBUST DEEP NEURAL NETWORKS WITH MINIMAL PREDICTION OVERHEAD

VIVEK ANAND
SPRING 2021

A thesis
submitted in partial fulfillment
of the requirements
for baccalaureate degrees
in Computer Science and Biology
with honors in Computer Science

Reviewed and approved* by the following:

Daniel Kifer
Professor of Computer Science and Engineering
Thesis Supervisor

John Hannan
Associate Department Head of Computer Science and Engineering
Associate Professor of Computer Science and Engineering
Honors Adviser

*Electronic Approvals on file in the Schreyer Honors College.

# Abstract

Over the past decade, Deep Neural Networks (DNNs) have become prevalent in several sensitive and security critical applications such as self-driving cars, facial recognition, anomaly detection, chatbots etc. In these applications, it is imperative that the DNN functions as intended and is robust to malicious adversaries and attacks. However, DNNs have been found to be systemically vulnerable to adversarial attacks. These adversarial attacks result from inputs being carefully manipulated resulting in incorrect outputs by the DNN. These attacks may even be imperceptible to a naked eye such that manual intervention may be insufficient or even infeasible. Till now, most defences to these adversarial attacks require significant computational overhead in both/either training and inference even on powerful graphics processing units (GPUs) resulting in considerable energy and time expended. In this work, to mitigate this overhead, we present a simple algorithm, FAST_DP_PREDICT to reduce computational costs during prediction. Specifically, we minimize the additional i.i.d. samples of the noise required to estimate the mean scoring function of the DNN during the certified prediction mechanism of the baseline PixelDP mechanism. We do this by leveraging the normal approximation to the binomial and adaptive sampling. In our extensive experiments on the CIFAR-10 dataset, we show that FAST_DP_PREDICT unequivocally outperforms the baseline PixelDP prediction mechanism across the board with at least a $9\times$ end to end speedup.

# Table of Contents

**List of Figures** iv

**List of Tables** v

**Acknowledgements** vi

**1 Introduction** 1
- 1.1 Background . . . 2
- 1.2 Related Work . . . 2
  - 1.2.1 Attacks . . . 2
  - 1.2.2 Best Effort Defences . . . 3
  - 1.2.3 Certified Defences . . . 4
- 1.3 Goals of the Thesis . . . 4
- 1.4 Outline . . . 5

**2 Description of Baseline: PixelDP [13]** 6
- 2.1 Preliminaries . . . 7
  - 2.1.1 Basic Machine Learning (ML) Background [13] . . . 7
  - 2.1.2 Adversarial Example Definition [13] . . . 7
  - 2.1.3 Robustness Definition [13] . . . 7
- 2.2 High Level Description . . . 8
- 2.3 Theorems . . . 8
- 2.4 Implementation Details . . . 9
  - 2.4.1 Adding Noise . . . 9
  - 2.4.2 Training . . . 10
  - 2.4.3 Inference . . . 10

**3 FAST_DP_PREDICT Details** 11
- 3.1 Motivation for Approach . . . 11
- 3.2 High Level Description . . . 12
- 3.3 Details . . . 12

**4 Evaluation** 14
- 4.1 Setup and Description . . . 15
  - 4.1.1 PixelDP Setup . . . 15

4.1.2 Training Details . . . . . 15
4.1.3 Prediction Details . . . . . 15
4.1.4 FAST_DP_PREDICT Implementation Details . . . . . 15
4.1.5 System Details . . . . . 15
4.2 Results . . . . . 16
4.3 Analysis . . . . . 16

**5 Conclusion** **18**
5.1 Comparison of FAST_DP_PREDICT to Baseline . . . . . 19
5.2 Implications of FAST_DP_PREDICT . . . . . 19
5.3 Limitations of Model . . . . . 19
5.4 Future Work . . . . . 19

**Bibliography** **21**

# List of Figures

1.1 Addition of small (imperceptible) calibrated noise can change inference of DNN from Panda to Gibbon [2] . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 2

# List of Tables

3.1 Runtimes for predictions of one CIFAR-10 image for original PixelDP prediction mechanism . . . . . 12

4.1 Prediction runtimes for entire CIFAR-10 test set . . . . . 16
4.2 Repeat Statistics for FAST_DP_PREDICT for threshhold of 0.01 . . . . . 16
4.3 Runtimes for predictions of one CIFAR-10 image . . . . . 16

# Acknowledgements

I thank Prof. Kifer for his guidance and direction over the past two years. I also sincerely thank my parents for their constant and unending support for my various endeavors. I thank all the professors, colleagues, friends etc. that have contributed to my growth over my undergraduate studies. I also sincerely thank UGPU07, my partner in crime for endlessly running my experiments and jobs for days on end. Thank you, you deserve a good rest. :)

# Chapter 1

# Introduction

## 1.1 Background

Over the past decade, Deep Neural Networks (DNNs), have become prevalent in several sensitive and security-critical tasks for self-driving cars, face recognition, anomaly detection, etc. In all these applications, it is imperative that the DNN functions as intended, i.e. is robust, despite any malicious attacks. However, DNNs have been found to be vulnerable to both blackbox [1] and whitebox attacks [2]. Some of the most interesting and hard-to-combat attacks are those where inputs are slightly modified by an adversary to result in incorrect outputs. These modified inputs may even be imperceptible from the original (correct) input that manual intervention may be insufficient and/or infeasible. For instance, Figure 1.1 shows one such example where an input panda image, which was correctly classified by a DNN is mis-classified to be a ”gibbon” after adding carefully calibrated noise. While this problem has been tackled by many prior studies since its discovery in 2014, solutions usually provide such robustness at the expense of accuracy.

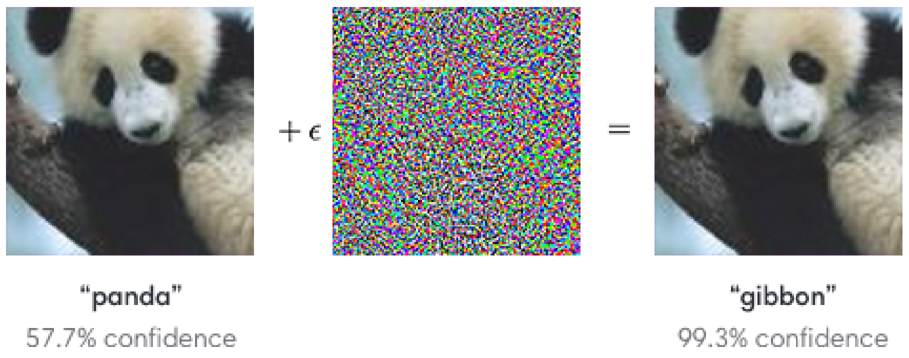

Figure 1.1: Addition of small (imperceptible) calibrated noise can change inference of DNN from Panda to Gibbon [2]

## 1.2 Related Work

### 1.2.1 Attacks

Adversarial attacks were first discovered in [3] which proposed solving an optimization problem to find the smallest change in the input that changes the output label. As this problem was intractable, they proposed relaxing it using the gradient of the loss.

The first general purpose and efficient attack was the Fast Gradient Sign Method (FGSM) proposed by [2] which modifies the input by subtracting the scaled sign of the gradient of the loss.

Subsequent works by [4] [5] use a finer iterative optimizer that performs FGSM for multiple iterations with a smaller stepsize and then clip/project the gradient to the appropriate feasible set. [5] additionally, uses the momentum memory and a scheme to build an ensemble of models to construct attacks.

[6] use an attack that optimizes over the set of potential adversarial distributions instead of simply sampling independently from one of them.

One of the most successful attacks [7], used a similar optimization based approach from [3].

The main difference is that they reformulate the optimization problem seen [3] by absorbing the custom adversarial loss into the objective function. This technique achieves a 100% success rate on ImageNet, CIFAR-10, MNIST etc. while also compromising defences such as L-BFGS [3] and Model Distillation [7].

[8] demonstrate a method to generate black-box attacks against a remote model without *any* knowledge of the model's details. They train a local model to substitute for the target DNN usin the label predictions of the remote model. Subsequently, the attacker leverages the local model to generate adversarial examples for the remote model.

The Elastic-Net attack [9] considers the problem of finding appropriate adversarial examples as an elastic-net regularized optimization problem.

Another set of attacks [10][11] utilizes Generative Adversarial Networks (GANs) to learn the adversarial distribution to then sample adversarial examples from.

### 1.2.2 Best Effort Defences

Most of the adversarial defences fall into the *best-effort* category, i.e. defences that may work well in practice but do not have rigorous certificates or guarantees of efficacy.

#### Adversarial Training

Of these best effort defences, most rely on Adversarial Training in which the models are trained on adversarial examples in addition to their original training set.

One of the first methods to utilize adversarial training was in [2] which used the FGSM produced examples during training. However, such a model was still found to be vulnerable to other attacks [12].

The Projected Gradient Descent (PGD) method found in [4] was demonstrated empirically to likely be a universal first-order $L_\infty$ attack. Consequentially, examples generated by PGD were used to train the model [12]. However, though this defence was shown to work well in practice [12] [13], training was significantly slower due to a high computational cost in generating the PGD examples.

To reduce the computational cost brought by the PGD training method, [14] sampled multiple adversarial examples from pre-trained models to increase the diversity of examples used for adversarial training. Sometimes, it was shown that performance was even better when compared to PGD-training [14].

#### Randomization based defences

Another of set of defences use randomization schemes to improve adversarial performance.

[15] won 2nd place in the NIPS 2017 adversarial examples challenge [16] by using random resizing and padding. [17] added random noise before each convolutional layer and then ensemble predictions to improve robustness. However, both these methods were compromised by white-box methods that leveraged the model gradients and jacobians [18].

Denoising the inputs from the adversarial noise is a straightforward method to remove the effects of the adversarial example. This method was examined through bit reduction and image blurring in [19], GAN based input cleansing [20], and Autoencoder based denoising [21]. However, though they are effective evidenced by the evaluation in their own papers, they have been found vulnerable to white-box and black-box attacks as evidenced by results from [22] [23].

### 1.2.3 Certified Defences

Certified Defences are defences against adversarial attacks that offer *guarantees* of robustness backed by rigorous theory.

Some of the formative work on these certified defenses modified the training procedure to minimize robustness violations. [24] used semidefinite programming to upper bound adversarial loss which they incorporate into the training procedure. [25] constrained the weight matrices of the linear and convolutional layers to be parseval tight frames during training to minimize sensitivity of the inputs. [26] formulate a dual problem to upper bound the adversarial polytope to use in training. However, these approaches do not readily scale to larger and more complex networks due to several restrictions and assumptions made about the networks [24] and high computational overhead.

To mitigate the computational overhead, another approach [27] combined robust optimization and adversarial training to to give formal guarantees of optimization. However, these guarantees required the network to be smooth (infinitely differentiable), and that the robustness guarantees provided were for expected loss rather than the accuracy.

The first general purpose robustness framework usable for most neural networks was PixelDP [13]. PixelDP exploits the similarity between robustness and differential privacy (DP). In DP, computations on databases are randomized to guarantee that output changes are bounded despite small changes in input. This is analogous to minor change (of a few pixels) of the input image, resulting in only a bounded change in its output. PixelDP uses this approach to add a calibrated noise layer to the DNN during training, so that inferences are guaranteed robustness up to a threshold. In practice [13] and my own independent experiments, this method does indeed produce robust DNNs. While DNNs produced by PixelDP are robust, their certified accuracy is considerably lower than their actual accuracy (by more than 40% in some cases) [13]. Moreover, though they have only a marginal increase in training, they have a significant prediction overhead.

The most recent, general purpose and state-of-the-art robustness framework is [28] which can turn any classifier that classifies well under gaussian noise into a smooth one that is robust under the $l2$ norm. Moreover, they prove a tight guarantee of robustness for their smoothed neural network. Additionally, their evaluations show that their smoothing framework outperforms [13] and [26] in accuracy without sizable training and prediction overhead.

## 1.3 Goals of the Thesis

The goals of this thesis are as follows.

1. Develop a framework to speed up prediction on PixelDP without sacrificing accuracy or certified accuracy.

2. Implement framework using Tensorflow on GPU.

3. Empirically compare performance of framework with PixelDP. In particular, analyze the runtimes of predictions and compare them to the baseline.

## 1.4 Outline

Chapter 2 introduces the details of the baseline model PixelDP [13]. Section 2.1 introduces the mathematical preliminaries and definitions used repeatedly in this work. Section 2.2 is a brief explanation of the PixelDP framework. Section 2.3 introduces the two key theorems that emphasize the connection between robustness and differential privacy. Section 2.4, discusses the nuances of the implementation details.

After the baseline is introduced and described in detail, we introduce our algorithm FAST_-DP_PREDICT in Chapter 3. Section 3.1, motivates the need for faster turnaround time during inference. Section 3.2 provides a brief high level explanation of FAST_DP_PREDICT without going into the implementation details. Section 3.3 introduces the theoretical rationale behind FAST_DP_PREDICT and provides the algorithm, comprehensive explanation and implementation details behind it.

With FAST_DP_PREDICT thoroughly explained in Chapter 3, we detail the evaluation setup, metrics and results in Chapter 4. Section 4.1 clearly details the hyperparameters, configuration and datasets used in all the experiments. Section 4.2 neatly tabulates the results obtained in the comparison to the baseline. Section 4.3 neatly summarizes the results in Section 4.2.

In Chapter 5, we conclude this thesis by discussing the FAST_DP_PREDICT's performance, implications, limitations and future steps. Section 5.1 elaborates on the key results between FAST_-DP_PREDICT and the baseline PixelDP model. Section 5.2 looks at the broader implications of FAST_DP_PREDICT to the adversarial robustness community and to industry. Section 5.3 meanwhile, looks at the (few) shortcomings of FAST_DP_PREDICT and methods to mitigate it. Section 5.4 explores meaningful future directions for this work.

# Chapter 2

# Description of Baseline: PixelDP [13]

In this chapter, we will introduce the baseline model, PixelDP in detail. This detail is necessary as FAST_DP_PREDICT heavily borrows from PixelDP. Moreover, the mathematical formulations and features of PixelDP are critical to implement FAST_DP_PREDICT.

We start by introducing some mathematical preliminaries and definitions.

## 2.1 Preliminaries

### 2.1.1 Basic Machine Learning (ML) Background [13]

We first introduce the basic ML classification background and formulations. A standard ML model in multiclass classification is defined as a function $f : \mathbb{R}^n \longrightarrow K$ where the $n$-dimensional space is mapped to labels in set $K = \{1, \text{...}\ \text{K}\}$. An input $x$ is mapped to a vector of scores $y(x) = y_1(x), ...y_K(x)$ such that $\forall k$, $y_k(x) \in [0, 1]$ and $\Sigma_{k=1}^{K} y_k(x) = 1$. The model then returns the label with highest probability, i.e score, $f(x) = \text{argmax}_{k \in K} y_k(x)$. The function mapping the input $x$ to the vector of scores $y$ is denoted as $Q$ and referred to as the scoring function. The function which returns the actual label of the prediction for an input $x$ is denoted as $f$ or the prediction function. In simple words, a classification model takes in various inputs and returns its predicted label.

### 2.1.2 Adversarial Example Definition [13]

We define an adversarial example of an input $x$ as $x + \alpha$ where $\alpha$ is a modification made by the attacker to input $x$. $\alpha$ has the same dimensions as $x$. To evaluate amount of change an adversary is allowed to make to the input, we constrain $\alpha$ to some arbitrary $p$-norm denoted by $||\alpha||_p$. For $1 \leq p < \infty$, $||\alpha||_p = (\Sigma_{i=1}^{n} |\alpha_i|^p)^{1/p}$.

The space that an adversary can choose alpha from is defined as the $p$-norm ball of radius $r$ or $B_p(r) := \alpha \in \mathbb{R}^n : ||\alpha|| \leq r$.

In simple words, an adversarial example is an input that has perturbed only as much as we allow it.

### 2.1.3 Robustness Definition [13]

Now that we have the definition of an adversarial example, we can proceed to define robustness to such examples. Given an input, a model is regarded robust to that input if it is insensitive to small perturbations to it. If a model is insensitive to small changes on input $x$, then $f(x) = f(x + \alpha)$ or the arg max label does not change for a bounded adversarial attack. This can be mathematically formulated as the following. $\forall \alpha \in B_p(L) \cdot y_k(x + \alpha) > \max_{i:i \neq k} y_i(x + \alpha)$ $L$ here is the magnitude of the attack according to the desired $p$-norm metric.

In simple words, a model is robust for an input if small changes to that input do not change the output.

## 2.2 High Level Description

The PixelDP framework is a certified defense against $p$-norm bounded adversarial attacks using a connection to differential privacy. It exploits a connection to differential privacy by adding a carefully calibrated noise layer such that the mean of the predictions for an input are $(\epsilon, \delta)$ DP with respect to the inputs. This result is proven by the theorems in Section 2.3. These theoretical implications combined with Monte-Carlo estimation of the mean output for an input at prediction time result in certified adversarial robust predictions.

## 2.3 Theorems

Two theorems are critical in certifying the robustness of the PixelDP framework. The following theorem, is a modified version of the expected output stability bound presented in the classical [29] which introduced the idea of differential privacy. A proof is irrelevant to comprehend FAST_DP_-PREDICT but can be seen in [29].

**Theorem 1 (Expected Output Stability Bound [29])** *Suppose a randomized function $A$ with bounded output $A(x) \in [0,1]$ satisfies $(\epsilon, \delta)$ Differential Privacy with respect to some $p$-norm metric where*

$$A(x) = (y_1(x), ...y_k(x)), y_k(x) \in [0,1]$$

*. Then the expected value of its output meets the following property.*

$$\forall k, \forall \alpha \in B_p(1) \cdot \mathbb{E}(y_k(x)) \leq e^{\epsilon}\mathbb{E}(y_k(x+\alpha)) + \delta$$

In simple words, Theorem 1 assures us that the expected value of any index of the output vector of the randomized function $A$ for an input $x$ is bounded for modifications to $x$. This fact is used to prove Theorem 2 and is critical to certify robustness.

The following theorem is proven in [13].

**Theorem 2 ([13])** *Suppose $A$ satisfies $(\epsilon, \delta)$- DP with respect to a $p$-norm metric. Then, for any input $x$ if for some $k \in K$*

$$\mathbb{E}(A_k(x)) > e^{2\epsilon} max_{i:i\neq k}\mathbb{E}(A_1(x), ..., E(A_k(x))) + (1+e^{\epsilon})\delta$$

*Then the multiclass classification models based on label probability vector $y(x) = (E(A_1(x), ..., E(A_k(x)))$ is robust to attacks $\alpha$ of size $||\alpha||_p \leq 1$ on input $x$*

Theorem 2 defines the robustness condition an $(\epsilon, \delta)$- DP function must satisfy in order to be robust to input $x$. This is proved in a few steps by using Theorem 1 and the robustness definition used previously.

In simple words Theorem 2 shows that for a model and input if the mean of the predicted probability for the predicted label is always greater than the mean probability for any of the other labels added to a few constant terms, then the model is robust for that input.

## 2.4 Implementation Details

Now that we have explained and elaborated on the theory, we can proceed to discuss and detail the implementation.

### 2.4.1 Adding Noise

The carefully calibrated noise layer is a crucial part of the PixelDP defence. This noise layer, from DP theory [29] can parametrized only when we know the sensitivity of the randomized function in question. The sensitivity of a function $g$ is defined as the maximum change in the output that can be produce by a change in the input given some distance metrics. ($p$ and $q$ norm respectively.

$$\Delta_{p,q}^{g} = \text{max}_{x,x':x \neq x'} \frac{||g(x) - g(x')||_q}{||x - x'||_p}$$

In this work, we will exclusively be using a $p$ and $q$-norm metric of 1 and 2 respectively. Using this sensitivity, we can calculate the noise needed using mechanisms from [29]. Two mechanims arise that can be used to generate the noise. (1) The Laplacian distribution allows us to get $(\epsilon, 0)$ DP (2) The Gaussian distribution allows us to get $(\epsilon, \delta)$ DP . In this work, we will be only using the Gaussian mechanism to add noise due to the additional flexibility provided. This Gaussian DP noise can be sampled from $N(0, \sigma^2)$ where $\sigma = \sqrt{2ln(\frac{1.25}{\delta})\Delta_{p,2}L/\epsilon}$. Here, $L$ is the constructive attack bound or the maximum attack size that we specify the network has to provably defend against.

Though we have clearly detailed how we calibrate and sample the noise, we still need to decide where to insert it into the DNN. The three most feasible options are to either insert it into the inputs directly, immediately after the first layer or in a specially designed autoencoder trained separately.

1. **Noise in Image**- This method is very trivial to add the noise to and can be thought of as adding noise directly to the input image. The sensitivity calculation here is trivial and is equal to 1 [13].

2. **Noise after first layer** - If the first layer is linear or convolutional, sensitivity can be calculated as follows [13]. If the weights are of the form $W \in \mathbb{R}^{m,n}$ the sensitivity is calculated as the matrix norm or as follows.

   $$||W||_{p,q} = \text{sup}_{x:||x||_p \leq 1}||Wx||_q$$

   This implies that $||W||_{1,2}$ is the maximum 2-norm of $W$'s columns and $||W||_{2,2}$ is the maximum singular value of $W$. Convolutional layers are unpacked into a matrix of the form $\mathbb{R}^{nd_{out} \times nd_{in}}$ where $n$ is the input size, $d_{in}$ is number of input channels and $d_{out}$ is number of output channels.

3. **Noise in a specially designed autoencoder**- Both of the above two methods have their benefits. However, adding noise in the image may not work well for some DNN architectures or the first layer of the network may not be linear. Consequently, training a separate autoencoder separately that uses one of the two above methods specifically for each dataset and then

stacking it on top of the prediction network allows both to be developed independently. Moreover, autoencoders are generally much smaller and easier to be trained as well. In this work, we exclusively use the autoencoder approach.

### 2.4.2 Training

Now that we have clearly shown how to sample the carefully calibrated noise and where to place this noise layer we can detail the training and inference procedures. Training a PixelDP network is not substantially different from training the unmodified version. The same loss and optimizer of the original optimizer can be used. The main difference is that the pre-noise computation is altered to constrain the sensitivity of the inputs to the $p$-norm adversarial attack. If it were not constrained, the sensitivity would blow up and the DP guarantees would be null and void. Let $Q(x) = h(g(x))$ where $g$ is the pre-noise computation and $h$ is the post-noise computation. To make $Q$ Pixel-DP, $g$ is transformed to $\tilde{g}$ such that $\tilde{g}$ has fixed sensitivity to $p$-norm attacks. Here, to constrain the sensitivity we normalize the weight matrices by column. Further details for other sensitivity constraining mechanisms for other $p$ and $q$ norms can be seen in [25].Then, carefully calibrated noise with standard deviation proportional to the sensitivity $\Delta$ and the attack bound $L$ is added to the pre-noise computation. Subsequently, the revised scoring function can be denoted by $A_Q(x) = h(\tilde{g}(x) + noise(\Delta, L, \epsilon, \delta))$. Note that in the training procedure, we optimize only for one sample of the noise for each input.

### 2.4.3 Inference

Though training was fairly straightforward, inference however, is a difference proposition due to certification of robustness.Theorem 2 requires $\mathbb{E}(A_Q(x))$ to certify and predict for an input. This cannot be calculated analytically as the surface of the neural network can be intractable. Therefore, Monte Carlo methods are used to calculate the expected value to estimate it at prediction time by sampling from the noise distribution repeatedly for any arbitrary confidence. Then using this estimation of $\mathbb{E}(A_Q(x))$, the confidence interval is calculated for each label using either Hoeffding's bounds [30] or Empirical Bernstein Bounds [31]. If the lower bound of the arg max label is strictly greater than the upper bound of every other label with confidence $\eta$, then $A_Q(x)$ is robust to to $p$-norm attacks of size $L$ for input $x$. If not, we cannot certify it to be robust on input $x$.

Though we have forced the PixelDP network to be robust to attacks of size $L$, by generalizing Theorem 2 we can obtain the maximum attack bound $L_{max}$, rather than the construction upper bound $L$ upto which the model can be deemed to be robust on a particular input. This is implemented by binary search combined with Theorem 2 for arbitrary precision metrics.

As $\mathbb{E}(A_Q(x))$ is calculated using Monte-Carlo methods, it is possible to get arbitrary levels of precision needed. In the paper, the authors of PixelDP indicate that 300 samples of noise are sufficient for accurate predictions and certifications.

# Chapter 3

# FAST_DP_PREDICT Details

## 3.1 Motivation for Approach

PixelDP was pathbreaking primarily because it was the first general purpose framework that was able to rigorously certify robustness of most DNNs with little training overhead. Given that the best defence upto then [12] required several additional SGD steps to find the best adversarial examples , the progress that PixelDP made in speeding up training is nothing short of remarkable.

Training costs are important but speedy inference is critical as well. A collision detection system in a self driving car needs to be not only robust to adversarial attacks but also needs to quickly decide whether to stop a car or not in case of unforeseen circumstances. If prediction is slow, deployment of these models is almost useless in quick response time applications. Based on the authors' own calculations, certifying a prediction has a $42\times$ overhead compared to a prediction from a non-PixelDP network whereas simple predictions without certification requires only a $3\times$ overhead.

The main reason why the original PixelDP method is slow during prediction and certification time is due to the fact that multiple samples of noise are needed to calculate to both estimate $E(A_Q(x))$ and the confidence interval for it. Calculating the certification bounds either via the Hoeffding's bounds or the Bernoulli bounds is simple arithmetic that does not add appreciable time to the calculation. This can be seen below in Table 3.1

| Job | Average Runtime (secs) | First Runtime (secs) |
|---|---|---|
| One CIFAR-10 image, no additional samples, no certification | 0.022 | 1.256 |
| One CIFAR-10 image, 300 additional samples, no certification | 0.271 | 1.72 |
| One CIFAR-10 image, 300 additional samples, with certification | 0.278 | 1.73 |

Table 3.1: Runtimes for predictions of one CIFAR-10 image for original PixelDP prediction mechanism

From Table 3.1, we can see that there is more than a $10\times$ slowdown in average runtime if we add additional samples that need to be run at prediction time. Additionally, we see that certification adds only a minor addition in runtime. The first runtime for an input is considerably larger than any of the above values as the implementation in Tensorflow performs optimizations and preprocessing [32] when the first input is run. This has no bearing on later predictions as this is simply a onetime overhead. Even then, we see that adding additional samples significantly increases runtime even on the first pass.

Moreover, as most of these predictions are run on a machine with a Graphics Processing Unit (GPU), many of these calculations can be batched together with other inputs needed to be certified. These additional predictions necessary for each input can be thought of as added inputs. Consequently, due to limited GPU memory, it will take much longer to run.

## 3.2 High Level Description

As we can see from Table 3.1, running multiple calls of $A_Q(x)$ for different noise samples for the same prediction is costly, if we were able to reduce the number of calls without losing much in our estimate of $E(A_Q(x))$, we could speed up prediction. We do precisely that, by limiting the number of samples needed, without sacrificing significant accuracy using an application of the Central Limit Theorem. [33]

## 3.3 Details

The following theorem is a variant of the well known central limit theorem [33] for binomial distributions.

**Theorem 3 (De Moivre- Laplace Theorem [33])** *The probability mass function of the sum of successes of independent Bernoulli trials $X_1, X_2, ..., X_n \sim Bern(p)$ converges to the normal distribution $N(np, np(1-p))$ as $n \longrightarrow \infty$*

We use Theorem 3 in conjunction with a simple test commonly used to determine whether enough samples are taken to approximate the binomial distribution with reasonable accuracy. If $np \geq 5$ and $n(1-p) \geq 5$, we can use the the normal approximation to the binomial [34].

If we argmax the $A_Q(x)$ inference as mentioned in Section 2.4.3, we obtain only the selected label for input $x$. Next, we will sum over all predictions for input $x$. If we divide by $n$ we obtain $p$ or the probability that a label is selected. If each of the entries in $np$ and $n(1-p)$ are greater than 5, we obtain only the significant entries and calculate the number of additional samples needed to be run to satisfy the normal approximation condition. If so, we sample them and then and merge with the existing predictions. This procedure is illustrated in the FAST_DP_PREDICTION algorithm listed below

**Algorithm 1:** Minibatched FAST_DP_PREDICT

```
Input: $A_Q(x)$ (PixelDP model), $data$ , $s$ (sample number)
Output: Predictions, Certifications
1  Predictions = []
2  for each minibatch $M$ of size $m$ from $data$ do
       /* If $A_Q(x)$ has dimension $m \times k$, $A_Q^{prediction}(x)$ has dimensions $m \times s \times k$ */
3      sampled_predictions = $A_Q^{prediction}(M, s)$
       /* Argmax predictions */
4      sampled_predictions = $argmax$(sampled_predictions)
       /* Getting probability each label is selected for each entry in $M$,
          dimension is $m \times k$ */
5      mean_predictions = REDUCE_MEAN(sampled_predictions)
6      p = mean_predictions / $m$
       /* Checking if normal approximation can be used */
7      if $mp < 5$ OR $m(1-p) < 5$ then
           /* Getting only those entries larger than a threshhold */
8          $mp, m(1-p)$ = SUBSET($mp$), SUBSET($m(1-p)$)
           /* Getting new $s'$ to get more samples */
9          $s' = |\lceil max(\frac{5}{p}, \frac{5}{1-p}) - s \rceil|$
           /* Merge with original samples and calculate mean predictions */
10         sampled_predictions = MERGE(sampled_predictions, $A_Q^{prediction}(M, s')$)
11         sampled_predictions = $argmax$(sampled_predictions)
12         mean_predictions = REDUCE_MEAN(sampled_predictions)
       /* append minibatch predictions to global predictions */
13     Predictions $\longleftarrow$ mean_predictions
14 return (Predictions, CERTIFY(Predictions, $A_Q(x)$))
```

The REDUCE_MEAN function here simply gets the mean for the total samples for each prediction. SUBSET returns only those entries larger than a threshhold for $np$ and $n(1-p)$. This is to discount entries where hardly any sample is seen. MERGE simple combines the input tensors across their formative axis. CERTIFY is simply the PixelDP certification mechanism as used in [13].

# Chapter 4

# Evaluation

## 4.1 Setup and Description

### 4.1.1 PixelDP Setup

The original PixelDP code by the authors of the paper was written in Tensorflow 1 but had multiple bugs and incompatibilities with the CUDA drivers and other system properties on UGPU07. Consequently, the PixelDP implementation was rewritten in Tensorflow 2 independently.

The PixelDP model exclusively used here is a PixelDP autoencoder stacked onto a 28-10 wide resnet [35]. The PixelDP autoencoder is composed of an encoder with 3 convolutional layers. The kernels of these layers have sizes $10 \times 10 \times 32$, $8 \times 8 \times 32$ and $5 \times 5 \times 32$ respectively. The DP noise is added after the first convolution. The decoder is simply the tied convolutional transpose of the encoder, which was implemented by the author. The combined PixelDP model was trained for a $p$-norm of 1, a $q$-norm of 2, an attack bound $L = 0.001$, $\epsilon = 1$ and $\delta = 0.05$.

### 4.1.2 Training Details

The dataset exclusively used here is the well known CIFAR-10 dataset comprising of 60000 RGB channel color images with a total of 10 labels [36]. The training set comprises of 50000 of the images and their corresponding labels while the test set is comprised of the remaining 10000 images. A validation set was created by separating 5000 examples from the training set. The 28-10 wide resnet was trained on an augmented CIFAR-10 dataset until validation loss ceased to improve. The autoencoder was separately trained on a similarly augmented dataset to minimize mean-squared-error. After both the resnet and the autoencoder were trained separately, the autoencoder was stacked onto the resnet and the weights of the resnet were finetuned further. The code for all of the experiments can be found at https://github.com/Vivdaddy/pixeldp2 .

### 4.1.3 Prediction Details

Due to computational constraints, the Hoeffding bounds are used exclusively as the Berenstein bounds requires sample variance calculations [31]. We use 300 $iid$ samples of DP-noise for each input and a batch size of 10 for PixelDP predictions.

### 4.1.4 FAST_DP_PREDICT Implementation Details

The same PixelDP model mentioned above is used for FAST_DP_PREDICT. For prediction and evaluation, Algorithm 1 is used. Once again the CIFAR-10 test dataset is used for evaluation but with different batch sizes and different sampling numbers which are ablated as below. For each sampling number, the maximum batch size for the dataset was used. The maximum batch size here, is defined as the batch size above which the GPU runs out of memory or starts thrashing.

### 4.1.5 System Details

The UGPU07 machine in Westgate 311W was exclusively used to run all of the experiments on a Nvidia GeForce GTX 1080 Ti GPU with CUDA 11.0. Python 3.6.3 and Tensorflow 2.4 were used for all the experiments.

## 4.2 Results

| Job | Average Runtime (secs) | First Runtime (secs) |
|---|---|---|
| PixelDP, 300 samples | 2836.51 | 2844.79 |
| FAST_DP_PREDICT, 10 samples | 149.41 | 153.87 |
| FAST_DP_PREDICT, 20 samples | 235.12 | 244.13 |
| FAST_DP_PREDICT, 30 samples | 325.36 | 327.56 |
| FAST_DP_PREDICT, 40 samples | 410.36 | 416.10 |
| FAST_DP_PREDICT, 50 samples | 493.31 | 496.56 |

Table 4.1: Prediction runtimes for entire CIFAR-10 test set

| Default Sample Size | Average Repeat Percentage (%) | Average Repeat Sample Size | Max Repeat Sample Size | Maximum Batch Size |
|---|---|---|---|---|
| 10 | 2.8 | 12.4 | 41 | 150 |
| 20 | 1.9 | 7.8 | 24 | 120 |
| 30 | 0.7 | 10.3 | 21 | 100 |
| 40 | 0.06 | 3.1 | 4 | 50 |
| 50 | 0 | NA | NA | 30 |

Table 4.2: Repeat Statistics for FAST_DP_PREDICT for threshhold of 0.01

| Job | Average Runtime (secs) | First Runtime (secs) |
|---|---|---|
| FAST_DP_PREDICT 30 samples no repeat, with certification | 0.094 | 1.56 |
| FAST_DP_PREDICT 30 samples, repeat 30 samples, with certification | 0.256 | 2.13 |
| PixelDP 300 additional samples, with certification | 0.278 | 1.73 |

Table 4.3: Runtimes for predictions of one CIFAR-10 image

## 4.3 Analysis

From Table 4.1 we see that as we increase number of samples during the FAST_DP_PREDICT step, the runtime gradually increases. Moreover, we see that we see almost a $6\times$ speedup even for 50 samples when compared to the standard PixelDP prediction.

From Table 4.2 we see that as the default sample size increases, the average repeat percentage comes down as well. Moreover, we see that the max repeat sample size is at most 41 for the entire CIFAR-10 dataset.

From Table 4.3, we see that the average runtime when there are no repeat samples is far less than if repeat samples are needed than the PixelDP prediction. Even when the FAST_DP_-PREDICT method requires 30 additional samples, the total time taken is still less than that for the PixelDP prediction.

Based on the results we have obtained, we recommend a sampling size of 30 be used during certified prediction as it appropriately trades off accuracy for computational overhead.

# Chapter 5

# Conclusion

## 5.1 Comparison of FAST_DP_PREDICT to Baseline

From Table 4.1 and Table 4.3 we see that FAST_DP_PREDICT is unequivocally faster than the standard PixelDP prediction mechanism. There is more than a $5\times$ speedup compared to the FAST_DP_PREDICT *even* with a sample size of 50. For samples of less size, the speedup obtained is even more.

For a single image, we see that the average runtime of FAST_DP_PREDICT is significantly better when we do not require repeat sampling. Even with repeated sampling, our average runtime is still marginally better than the PixelDP sampling. Even though the first runtime is larger, that can be discounted due to it being a onetime overhead.

From Table 4.2, we can see that we have a very low repeat sampling percentage even for a paltry default sampling size of 10. When the default sampling size reaches 30, the repeat percentage is hardly appreciable (less than 1 out of every 100 samples). When the default sample size is 50, we do not *ever* need to resample.

## 5.2 Implications of FAST_DP_PREDICT

FAST_DP_PREDICT has the potential to revolutionize deployment of adversarial robust neural networks. PixelDP was a pathbreaking work in improving certified adversarial robustness with a general purpose framework. However, it was held down from being useful in practice due to the heavy sampling overhead in the certified prediction mechanism.

FAST_DP_PREDICT, greatly speeds up prediction without sacrificing certification accuracy (significantly) by minimizing the number of random noise samples taken for most inputs. Moreover, if more samples are needed for a certain input, FAST_DP_PREDICT carefully draws only the required additional samples needed for robust prediction and certification.

## 5.3 Limitations of Model

FAST_DP_PREDICT works very well in practice and does not sacrifice any accuracy if the normal approximation thresholds are carefully selected. However, from the theoretical standpoint, there is clearly a loss in certification accuracy by approximating the binomial distribution with a gaussian distribution. This certification loss can be approximated by the Berry-Esseen Theorem [37]. However, bounds obtained from it are much too loose [31]. In practice, with the CIFAR-10 dataset, little no loss is seen from the approximation. Specialized bounds that are tighter for such an approximation would be highly beneficial from a theoretical standpoint.

## 5.4 Future Work

As mentioned above, finding better bounds for the normal approximation to the binomial is a natural direction. One example that could be useful would be incorporating the the realizable test set bound detailed in Section 3 of [38] for the estimation of the certification. This bound could in fact be tighter than the current bound but that needs to be explored.

Additionally, a natural direction for this work to proceed is to investigate other smoothing mechanisms that provide better guarantees of accuracy without adding significant training or prediction overhead. A natural starting point would be the current state-of-the-art work [28].

# Bibliography

[1] Nicolas Papernot et al. “Practical Black-Box Attacks against Machine Learning”. In: *Proceedings of ACM CCS* (2017). DOI: 10.1145/3052973.3053009. URL: http://dx.doi.org/10.1145/3052973.3053009.

[2] Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. *Explaining & Harnessing Adversarial Examples*. 2014. arXiv: 1412.6572 [stat.ML].

[3] Christian Szegedy et al. *Intriguing properties of neural networks*. 2014. arXiv: 1312.6199 [cs.CV].

[4] Alexey Kurakin, Ian Goodfellow, and Samy Bengio. *Adversarial examples in the physical world*. 2017. arXiv: 1607.02533 [cs.CV].

[5] Yinpeng Dong et al. *Boosting Adversarial Attacks with Momentum*. 2018. arXiv: 1710.06081 [cs.LG].

[6] Tianhang Zheng, Changyou Chen, and Kui Ren. *Distributionally Adversarial Attack*. 2018. arXiv: 1808.05537 [cs.LG].

[7] Nicholas Carlini and David Wagner. “Towards Evaluating the Robustness of Neural Networks”. In: *2017 IEEE Symposium on Security and Privacy (SP)* (2017). DOI: 10.1109/sp.2017.49. URL: http://dx.doi.org/10.1109/SP.2017.49.

[8] Nicolas Papernot et al. “Distillation as a Defense to Adversarial Perturbations Against Deep Neural Networks”. In: *2016 IEEE Symposium on Security and Privacy (SP)* (2016). DOI: 10.1109/sp.2016.41. URL: http://dx.doi.org/10.1109/SP.2016.41.

[9] Pin-Yu Chen et al. *EAD: Elastic-Net Attacks to Deep Neural Networks via Adversarial Examples*. 2018. arXiv: 1709.04114 [stat.ML].

[10] Chaowei Xiao et al. *Generating Adversarial Examples with Adversarial Networks*. 2019. arXiv: 1801.02610 [cs.CR].

[11] Augustus Odena, Christopher Olah, and Jonathon Shlens. *Conditional Image Synthesis With Auxiliary Classifier GANs*. 2017. arXiv: 1610.09585 [stat.ML].

[12] Aleksander Madry et al. “Towards deep learning models resistant to adversarial attacks”. In: *arXiv preprint arXiv:1706.06083* (2017).

[13] Mathias Lecuyer et al. “Certified Robustness to Adversarial Examples with Differential Privacy”. In: *2019 IEEE Symposium on Security and Privacy (SP)* (2019). DOI: 10.1109/sp.2019.00044. URL: http://dx.doi.org/10.1109/SP.2019.00044.

[14] Florian Tramèr et al. *Ensemble Adversarial Training: Attacks and Defenses*. 2020. arXiv: `1705.07204 [stat.ML]`.

[15] Cihang Xie et al. *Mitigating Adversarial Effects Through Randomization*. 2018. arXiv: `1711.01991 [cs.CV]`.

[16] Alexey Kurakin et al. "Adversarial attacks and defences competition". In: *The NIPS'17 Competition: Building Intelligent Systems*. Springer, 2018, pp. 195–231.

[17] Xuanqing Liu et al. *Towards Robust Neural Networks via Random Self-ensemble*. 2018. arXiv: `1712.00673 [cs.LG]`.

[18] Anish Athalye et al. *Synthesizing Robust Adversarial Examples*. 2018. arXiv: `1707.07397 [cs.CV]`.

[19] Weilin Xu, David Evans, and Yanjun Qi. "Feature Squeezing: Detecting Adversarial Examples in Deep Neural Networks". In: *Proceedings 2018 Network and Distributed System Security Symposium* (2018). DOI: `10.14722/ndss.2018.23198`. URL: `http://dx.doi.org/10.14722/ndss.2018.23198`.

[20] Pouya Samangouei, Maya Kabkab, and Rama Chellappa. *Defense-GAN: Protecting Classifiers Against Adversarial Attacks Using Generative Models*. 2018. arXiv: `1805.06605 [cs.CV]`.

[21] Dongyu Meng and Hao Chen. *MagNet: a Two-Pronged Defense against Adversarial Examples*. 2017. arXiv: `1705.09064 [cs.CR]`.

[22] Yash Sharma and Pin-Yu Chen. *Bypassing Feature Squeezing by Increasing Adversary Strength*. 2018. arXiv: `1803.09868 [stat.ML]`.

[23] Nicholas Carlini and David Wagner. *MagNet and "Efficient Defenses Against Adversarial Attacks" are Not Robust to Adversarial Examples*. 2017. arXiv: `1711.08478 [cs.LG]`.

[24] Aditi Raghunathan, Jacob Steinhardt, and Percy Liang. *Certified Defenses against Adversarial Examples*. 2020. arXiv: `1801.09344 [cs.LG]`.

[25] Moustapha Cisse et al. *Parseval Networks: Improving Robustness to Adversarial Examples*. 2017. arXiv: `1704.08847 [stat.ML]`.

[26] Eric Wong and J. Zico Kolter. *Provable defenses against adversarial examples via the convex outer adversarial polytope*. 2018. arXiv: `1711.00851 [cs.LG]`.

[27] Aman Sinha et al. *Certifying Some Distributional Robustness with Principled Adversarial Training*. 2020. arXiv: `1710.10571 [stat.ML]`.

[28] Jeremy M Cohen, Elan Rosenfeld, and J. Zico Kolter. *Certified Adversarial Robustness via Randomized Smoothing*. 2019. arXiv: `1902.02918 [cs.LG]`.

[29] Cynthia Dwork et al. "Calibrating noise to sensitivity in private data analysis". In: *Theory of cryptography conference*. Springer. 2006, pp. 265–284.

[30] Wassily Hoeffding. "Probability inequalities for sums of bounded random variables". In: *The Collected Works of Wassily Hoeffding*. Springer, 1994, pp. 409–426.

[31] Andreas Maurer and Massimiliano Pontil. "Empirical Bernstein bounds and sample variance penalization". In: *arXiv preprint arXiv:0907.3740* (2009).

[32] Martín Abadi et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. 2015. URL: `https://www.tensorflow.org/`.

[33] "Central Limit Theorem". In: *The Concise Encyclopedia of Statistics*. New York, NY: Springer New York, 2008, pp. 66–68. ISBN: 978-0-387-32833-1. DOI: `10.1007/978-0-387-32833-1_50`. URL: `https://doi.org/10.1007/978-0-387-32833-1_50`.

[34] John W. Pratt. "A Normal Approximation for Binomial, F, Beta, and Other Common, Related Tail Probabilities, II". In: *Journal of the American Statistical Association* 63.324 (1968), pp. 1457–1483. ISSN: 01621459. URL: `http://www.jstor.org/stable/2285896`.

[35] Sergey Zagoruyko and Nikos Komodakis. *Wide Residual Networks*. 2017. arXiv: `1605.07146 [cs.CV]`.

[36] Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. "CIFAR-10 (Canadian Institute for Advanced Research)". In: (2008). URL: `http://www.cs.toronto.edu/~kriz/cifar.html`.

[37] A.C Berry. "The Accuracy of the Gaussian Approximation to the Sum of Independent Variates". In: *Transactions Of The American Mathematical Society* (49 1941), pp. 122–136.

[38] John Langford and Robert Schapire. "Tutorial on Practical Prediction Theory for Classification." In: *Journal of machine learning research* 6.3 (2005).

# Vivek Anand

## EDUCATION

**The Pennsylvania State University**, University Park, PA June 2017-May 2021
Bachelor of Science (Honors), Computer Science
Bachelor of Science, Biology
Minor: Statistics

## MAJOR HONORS AND AWARDS

### Millennium Scholar

- Highly selective research-oriented program designed to groom students for a PhD in STEM
- Scholarship granted to ~30 students each year with full tuition and room and board

### Schreyer Honors Scholar

- Honors Program for high achieving students.
- Top 2% of undergraduate students at Penn State.

## RESEARCH EXPERIENCE

### Summer Research Intern

PI: Prof. Adam Wierman **June 2020 - Present**
**The California Institute of Technology (Caltech)**

- Synthesized algorithmic frameworks to schedule directed acyclic graphs of machine learning jobs in the cloud with energy considerations. (Preparing manuscript for publication)
- Theoretically proved bounds for pareto front objective using optimization and graph theory.
- Identified and extensively tested candidate heuristics for possible workload.

### Research Assistant

PI: Prof. Daniel Kifer **July 2019-Present**
**The Pennsylvania State University**

- Developing theoretical methods using differential privacy to weight and tradeoff adversarial robustness and accuracy.
- Harnessed evolutionary algorithms to improve adversarial robustness of Deep Neural Networks.
- Performed ablation studies using Tensorflow 2 for neural network robustness on GPUs.

### Summer Research Intern

PI: Prof. Sitabhra Sinha **June 2019- August 2019**
**The Institute of Mathematical Sciences, Chennai, India**

- Simulated a financial market using a mechanistic agent based financial model in python.
- Verified that resulting time series of returns followed inverse cubic law using power law fitting tools.
- Coded several feedforward-neural networks to model agent trading behaviour.

### Research Assistant

PI: Prof. Ottar Bjornstad **Nov 2018-May 2019**

**Center for Infectious Disease Dynamics, The Pennsylvania State University –**

- To perform multivariate analyses of Human Papilloma Virus (HPV) data from the Norwegian Government (data from mandatory nationwide vaccination for HPV from 2009 for all women at age 12 – tracks the prevalence and strains of the virus involved.)
- Model above information for the country of Norway.

### Summer Research Intern

**Life Sciences Innovation Labs, Tata Consultancy Services, June-July 2018**

- Performed several tests to predict transcription factor binding probabilities using a deep neural network.
- Obtained binding probabilities and AUCs (Area Under Receiver Operator Characteristic Curve) for various cell type transcription factor combinations.
- Assessed efficacy of the model with several Single Nucleotide Polymorphisms (SNPs) using programming and graphical techniques.

### Summer Research Intern

PI: Prof. Madhav Marathe **May-June 2018**

**Virginia Tech Biocomplexity Institute, Network Dynamics and Simulation Systems Laboratory,**

- Modelled spread of diseases and intervention measures (vaccination, social distancing, school closure etc.) using several mathematical tools and a special web enabled social networks simulation tool.
- Examined various vaccination strategies for influenza in rural populations using a computational tool Synthetic Information Based Epidemiological Laboratory (SIBEL) by changing various input parameters such as days of experiment, efficacy and compliance of vaccines, basic reproductive number of disease etc.
- Used statistical and survey analyses to prepare a custom survey to interpret school networks in government and private schools in India to better understand the spread of disease.

## OTHER HONORS AND AWARDS

### The President's Freshman Award

- Perfect GPA after freshman year

### Dean's List

- Cumulative and term GPA > 3.5 every semester.

### Paul Morrow Endowed Scholarship

- Scholarship for high achieving College of Engineering Students at Penn State.

## PROGRAMMING LANGUAGES & ENVIRONMENTS

Python 2 & 3, Tensorflow 2, C, R, GNU/Linux OS, Latex, Matlab, Pytorch

## PUBLICATIONS AND PRESENTATIONS

- **Energy Aware Scheduling Algorithms for Machine Learning Precedence Constrained Tasks** with A. Wierman, J. Yu, Y. Su **(in preparation)**
- Summer Undergraduate Research Fellowship (SURF) Poster Day as REU participant at Caltech