On the Connection between Differential Privacy and Adversarial Robustness in Machine Learning

Article · February 2018

CITATIONS
7

READS
865

5 authors, including:

Mathias Lécuyer
Columbia University
16 PUBLICATIONS **263** CITATIONS

SEE PROFILE

On the Connection between Differential Privacy and Adversarial Robustness in Machine Learning

Mathias Lecuyer, Vaggelis Atlidakis, Roxana Geambasu, Daniel Hsu, Suman Jana
Columbia University

Abstract

Adversarial examples in machine learning has been a topic of intense research interest, with attacks and defenses being developed in a tight back-and-forth. Most past defenses are best-effort, heuristic approaches that have all been shown to be vulnerable to sophisticated attacks. More recently, rigorous defenses that provide formal guarantees have emerged, but are hard to scale or generalize. A rigorous and general foundation for designing defenses is required to get us off this arms race trajectory. We propose leveraging differential privacy (DP) as a formal building block for robustness against adversarial examples. We observe that the semantic of DP is closely aligned with the formal definition of robustness to adversarial examples. We propose *PixelDP*, a strategy for learning robust deep neural networks based on formal DP guarantees. PixelDP networks give theoretical guarantees for a subset of their predictions regarding the robustness against adversarial perturbations of bounded size. Our evaluation with MNIST, CIFAR-10, and CIFAR-100 shows that PixelDP networks achieve accuracy under attack on par with the best-performing defense to date, but additionally certify robustness against meaningful-size 1-norm and 2-norm attacks for 40-60% of their predictions. Our experience points to DP as a rigorous, broadly applicable, and mechanism-rich foundation for robust machine learning.

1 Introduction

Deep neural networks (DNNs) perform exceptionally well on many artificial intelligence tasks, including safety- and security-sensitive applications such as self-driving cars [4], malware classification [37], face recognition [36], and critical infrastructure systems [47]. Robustness against malicious attacks is important in many of these applications, yet in recent years it has become increasingly clear that DNNs are extremely vulnerable to a broad range of attacks. Among the possible attacks – broadly surveyed in [35] – are *adversarial examples*: the adversary finds small perturbations to correctly classified inputs that cause the model to produce an erroneous prediction, often of the adversary's choosing [41]. Adversarial examples pose serious threats to security-critical applications. A classic example is an adversary that attaches a small, human-imperceptible sticker onto a stop sign that causes a self-driving car to recognize it as a yield sign. Adversarial examples have also been demonstrated in other domains such as reinforcement learning [24] and generative models [23].

Since the initial demonstration of adversarial examples [41], numerous of attacks and defenses have been proposed, each building on one another. For example, model distillation, proposed as a robust defense in [34], was subsequently broken in [6]. Similarly, [27] claimed that it is unlikely that adversarial examples will fool machine learning (ML) models in the real-world due to the rotation and scaling introduced by even slightest camera movements. However, [2] demonstrated a new attack strategy that is robust to rotation and scaling. While this back-and-forth has clearly advanced the state of the art in adversarial ML, we believe that the field needs rigorous, theory-backed foundations for defense to set us off this arms' race trajectory. Very recently, research into provable defenses has emerged [22, 40], but no solution has been shown to scale to real-world-sized DNNs.

This paper proposes using *differential privacy* (DP) theory as a rigorous, mathematical foundation for defense against adversarial examples. Somewhat surprisingly (though intuitive in retrospect), we observe that the semantic of DP is closely aligned with a formal definition of robustness to adversarial examples. We establish this relationship formally in §2. Informally, DP is a framework for randomizing algorithms running on databases of rows by adding noise such that their outputs enforce a particular privacy semantic for individual rows or small sets of rows in the database. The privacy semantic enforced by DP is that small changes in the database (e.g., removing or altering one row or a small set of rows) will result in small and bounded changes in the output distribution. Separately, robustness against adversarial examples can be defined as ensuring that small changes in an input (such as changing a few pixels in an image) will not result in drastic changes to a DNN's predictions (such as changing its label from a stop to a yield sign). Thus, if we think of a DNN's inputs (e.g., images) as databases in DP parlance, and individual features (e.g., pixels) as rows in DP, we observe that randomizing the outputs of a DNN's prediction function to enforce DP on a small number of

pixels in an image *guarantees* robustness of predictions against adversarial examples that can change up to that number of pixels. The connection can be expanded to standard attack norms, including 1- and 2-norms.

Based on the DP-robustness connection, we develop *PixelDP*, the first strategy for learning robust ML models based on DP theory (§3). PixelDP lets models give theoretical guarantees for a subset of their predictions, regarding the robustness against adversarial perturbations of bounded size. Specifically, for inputs on which a PixelDP DNN gives a *robust prediction*, it guarantees that there are no other inputs in the vicinity of those inputs (up to a particular distance, L, measured in standard 1- or 2-norms) for which the model would have given a different prediction. A PixelDP DNN incorporates in its architecture an additional layer, called the *noise layer*, which randomizes the network's computation to bound its sensitivity to changes in the input up to L. We then adapt the DNN's training and prediction procedures to account for this randomization and leverage the DP bounds to reason about the robustness of individual predictions.

Our experience implementing PixelDP for three image classification DNNs (MNIST, CIFAR-10, and CIFAR-100) and evaluating these networks on three datasets (§4) suggests that: (1) PixelDP's accuracy on benign testing examples is on par with state-of-the-art defenses that lack formal robustness guarantees (88.1% vs. 87.3%). (2) Accuracy on attacked testing examples also matches those defenses, but predictions that PixelDP deems *robust* (40-60% of our testing sets) are substantially more precise than predictions of those defenses. (3) The design space afforded by DP – a very general, flexible, and mature theory – is large and presents numerous untapped opportunities for further improving robustness in the future.

We commit to making public all code and experimental settings. We hope that the prototypes will spur exploration of the large and rigorous design space opened by PixelDP to further improve robustness in ML.

2 DP-Robustness Connection

This section provides the formal framework for aligning the two concepts, differential privacy and robustness against adversarial attacks. We prove the connection for a restricted case here and generalize it in §3.

2.1 DP Background

The theory of DP is concerned with whether the output of a computation over a database can reveal information about individual records in the database. To prevent such information leakage, randomness is introduced into the computation so that details of individual records are "hidden" by the randomness in a certain sense.

A (randomized) algorithm A that takes as input a database d (i.e., a set of records, or rows) and outputs a value in a space B is said to satisfy (ε,δ)-*DP with respect to a metric* ρ *over databases* if, for any databases d and d' with $\rho(d,d') \leq 1$, and for any subset of possible outputs $S \subseteq B$, we have

$$P(A(d) \in S) \leq e^{\varepsilon} P(A(d') \in S) + \delta. \quad (1)$$

Here, $\varepsilon > 0$ and $\delta \in [0,1]$ are parameters that quantify the strength of the privacy guarantee. In the standard definition of DP, the metric ρ is the Hamming metric, which simply counts the number of entries that differ in the two databases. For small values of ε and δ, the standard (ε,δ)-DP guarantee implies that changing a single entry in the database cannot change the output distribution very much. We shall discuss different metrics ρ that correspond to norms in §3.2.

A key property of DP is the *post-processing guarantee* [10]: any computation applied on the output of an (ε,δ)-DP randomized algorithm remains (ε,δ)-DP.

2.2 ML Background

An ML model can be viewed as a function mapping inputs – typically a vector of numerical feature values – to an output. For multiclass classification tasks, the output is a label from a set of possible labels; for regression tasks, the output is a real number. In this work we focus on multiclass classification models, which are a good fit for image classification – the type of task on which most research in adversarial examples focuses. We discuss extensions to regression models in §3.5.

Formally, a multiclass classification model is a function $f\colon \mathbb{R}^n \to \mathscr{Y}$ that maps n-dimensional inputs to a label in the set $\mathscr{Y} = \{1,\ldots,Y\}$ of all possible labels. Such models often predict a probability distribution $(p_1(x),\ldots,p_Y(x))$ over the Y possible labels for a given input x, and then return a label with highest probability, i.e., $f(x) = \arg\max_{y\in\mathscr{Y}} p_y(x)$.

2.3 Adversarial Examples Background

Adversarial examples are a class of attack against ML models, particularly studied in the context of deep neural networks for multiclass image classification. The attacker constructs a small change to a given, fixed input, that wildly changes the predicted output. Notationally, if the input is x, we denote an adversarial version of that input by $x+\alpha$, where α is the change or perturbation introduced by the attacker. When x is a vector of pixels (for images), then x_i is the i'th pixel in the image, and α_i is the change to the i'th pixel.

It is natural to constrain the amount of change an attacker is allowed to make to the input, and for simplicity we measure this by the p-norm of the change, denoted by $\|\alpha\|_p$. For $1 \leq p < \infty$, the p-norm of α is defined by $\|\alpha\|_p = (\sum_{i=1}^n |\alpha_i|^p)^{1/p}$; for $p = \infty$, it is $\|\alpha\|_\infty = \max_i |\alpha_i|$. Also commonly used is the 0-norm

(which is technically not a norm): $\|\alpha\|_0 = |\{i : \alpha_i \neq 0\}|$. A small 0-norm attack is permitted to arbitrarily change a few entries of the input; for example, an attack on the image recognition system for self-driving cars based on putting a sticker in the field of vision is such an attack [11]. Small p-norm attacks for larger values of p (including $p = \infty$) require the changes to the pixels to be small in an aggregate sense, but the changes may be spread out over many or all features. A change in the lighting condition of an image may correspond to such an attack [25]. The latter attacks are generally considered more powerful, as they can easily remain invisible to human observers.

Let $B_p(r) := \{\alpha \in \mathbb{R}^n : \|\alpha\|_p \leq r\}$ be the p-norm ball of radius r. For a given classification model, f, and a fixed input, $x \in \mathbb{R}^n$, an attacker is able to craft a successful adversarial example of size L for a given p-norm if they find $\alpha \in B_p(L)$ such that $f(x+\alpha) \neq f(x)$. The attacker thus tries to find a small change to x that will change the predicted label.

Attacks may be further dichotomized as follows: (i) untargeted attacks, in which the adversary tries to change the predicted label to any other label (reflected by the right-side condition of the preceding equation); and (ii) targeted attacks, in which the adversary tries to change the predicted label to a particular label, $y \neq f(x)$ (the right-side condition in the preceding equation changes to $f(x+\alpha) = y \neq f(x)$). Targeted attacks are generally more challenging to achieve than untargeted attacks (in terms of requiring bigger changes to an input), although completely undefended networks have been shown to be highly vulnerable to both types of attacks [26, 6].

2.4 Robustness Definition

Intuitively, a predictive model may be regarded as robust if its output is insensitive to small changes to any plausible input that may be encountered in deployment. To formalize this notion, we first must establish what qualifies as a plausible input. This is difficult: the literature on adversarial examples has not settled on such a definition. Instead, model insensitivity to input changes is typically assessed on inputs from a test set of data that are not used in model training, and in this work, we shall also adopt this approach.

Next, we must establish a definition for sensitivity to small changes to an input. We say a model f is insensitive, or *robust*, to attacks of p-norm L on a given input x if $f(x) = f(x+\alpha)$ for all $\alpha \in B_p(L)$. If f is a multiclass classification model based on label probabilities (as in §2.2), this is equivalent to

$$\forall \alpha \in B_p(L) \,.\, p_i(x+\alpha) > \max_{j:j\neq i} p_j(x+\alpha). \quad (2)$$

In other words, a small change in the input does not change which label receives the highest probability.

2.5 DP-Robustness Connection

Equations 1 and 2 provide a link between DP and robustness to adversarial examples. Regard the feature values (e.g., pixels) of an input x as the records in a database, and regard $(p_1(x), \ldots, p_Y(x))$ as the probability distribution for the output of a randomized algorithm A on input x with range $\mathscr{Y}$.[1] Now, we say that A is an *(ε, δ)-pixel-level differentially private* (or *(ε, δ)-PixelDP*) model if it satisfies (ε, δ)-DP (both for a given metric). This is formally equivalent to the standard definition of DP, but we use this terminology to emphasize the context in which we apply the definition.

Proposition 1. *Suppose A satisfies (ε, δ)-PixelDP with respect to the 0-norm metric, and let $p_j(x) = P(A(x) = j)$ for each $j \in \mathscr{Y}$. For any input x, if for some $i \in \mathscr{Y}$,*

$$p_i(x) > e^{2\varepsilon} \max_{j:j\neq i} p_j(x) + (1+e^{\varepsilon})\delta,$$

then the multiclass classification model based on label probabilities $(p_1(x), \ldots, p_Y(x))$ is robust to attacks of 0-norm 1 on input x.

Proof. Consider any $\alpha \in B_0(1)$, and let $x' := x + \alpha$. From Equation (1), we have

$$P(A(x) = i) \leq e^{\varepsilon} P(A(x') = i) + \delta,$$
$$P(A(x') = j) \leq e^{\varepsilon} P(A(x) = j) + \delta, \quad j \neq i.$$

From the first inequality, we obtain a lower-bound on $p_i(x')$, and from the second inequality, we obtain an upper-bound on $\max_{j\neq i} p_j(x')$. The hypothesis in the proposition statement implies that the lower-bound is higher than the upper-bound, which in turn implies the condition from Equation (2) for robustness at x. □

3 PixelDP Learning Strategy

Based on the preceding connection, we formulate the *PixelDP learning strategy*, a generic and broadly applicable approach to increase robustness of ML models. At a high level, the strategy is to use DP techniques to ensure that a learned model is not (very) sensitive to small changes in the input. Focusing on DNN models for image classification, Fig. 1 shows an example three-layer DNN architecture (original network shown in blue) and the changes introduced to the network under the PixelDP strategy (changes shown in red). The changes can be summarized as: (1) introduction of a *noise layer*, which randomizes the DNN to enforce (ε, δ)-PixelDP, and (2) changes to the *training* and *prediction procedures* to account for the added noise. After an overview of the PixelDP DNN's functioning (§3.1), we detail our current design of the new layer and procedures (§3.2-3.4). §3.5

[1] The probabilities $p_i(x)$ themselves may also be determined using additional randomness.

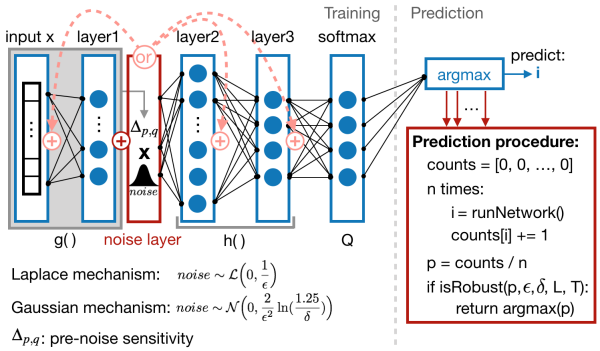


Fig. 1: **PixelDP architecture.** In blue, the original DNN. In red, the noise layer that provides the (ε, δ)-DP guarantees. The noise can be added to the inputs *or* any of the following layers, but the distribution is rescaled by the sensitivity $\Delta_{p,q}$ of the computation performed by each layer before the noise layer. Predictions use multiple noise draws to measure the noise-induced probability distribution over labels.

discusses broader applications of our strategy to general classification and regression problems.

3.1 Overview

The DNN defines a map Q from images x to probability distributions over labels $Q(x) = (Q_1(x), \ldots, Q_Y(x))$. PixelDP turns Q into a randomized mechanism A_Q that, upon input x, returns a random label $i \sim Q(x)$. If arbitrary randomization is applied, then mechanism A_Q may remain as sensitive to small input changes as Q. To reduce this sensitivity, PixelDP alters the computation of some initial layers of the DNN to satisfy (ε, δ)-PixelDP (with respect to p-norm). We regard subsequent computation after this randomized DP computation as *post processing* of its output.

In detail, suppose $Q(x) = (h_1(g(x)), \ldots, h_Y(g(x)))$, where g is the computation of some number of initial layers of the DNN (just one in Fig. 1), and for each label i, h_i is the subsequent computation that produces $Q_i(x)$ (the second layer and beyond in Fig. 1). We replace g with a randomized function $\tilde{g}$ that satisfies (ε, δ)-PixelDP. Concretely, we achieve this by introducing a *noise layer* (shown in red in Fig. 1) that adds zero-mean noise to the output of g with standard deviation proportional to the sensitivity Δ of g (i.e., the preceding layers, the grey box in Fig. 1). Now, we obtain a new randomized function $\tilde{Q}$ via $\tilde{Q}(x) = (h_1(\tilde{g}(x)), \ldots, h_Y(\tilde{g}(x)))$, which, through the post-processing property of DP [10], also satisfies (ε, δ)-PixelDP. The overall randomized mechanism $A_{\tilde{Q}}$ that returns a random label $i \sim \tilde{Q}(x)$ upon input x has the property that small changes to the input image do not change the distribution of the outputted label very much: we have for each $i = 1, \ldots, Y$,

$$P(A_{\tilde{Q}}(x+\alpha) = i) \leq e^{\varepsilon} P(A_{\tilde{Q}}(x) = i) + \delta, \quad \alpha \in B_p(1).$$

The resulting PixelDP DNN architecture is non-standard because it is randomized. Nevertheless, we can still train this randomized architecture just as standard DNNs are trained, with a few notable modifications. We describe the new *training procedure* in §3.3.

The robustness properties afforded to our PixelDP DNN architecture concern the probability distribution of its output, $P(A_{\tilde{Q}}(x) = \cdot)$, given any input x. Due to the potentially complex nature of the layers of the DNN following the noise layer, it may be difficult to compute these output probabilities. We therefore resort to Monte Carlo methods to estimate these probabilities at *prediction time*; these probabilities are used to reason about the robustness of the predictions, as shown in §3.4.

Each of the components of the PixelDP strategy – noise layer, training procedure, and prediction procedure – present a rich design space. In subsequent sections we describe a few design options for each – those for which we have experience to date – and give our recommendations for how to navigate the choices ("Our Approach" headings in each section). Section 3.5 gives a sense of the broader design space that DP affords for robustness against adversarial examples.

3.2 Noise Layer

The noise layer enforces PixelDP by inserting noise inside the DNN using one of two well-known DP mechanisms: the Laplacian and Gaussian mechanisms. Both rely upon the *sensitivity* of the pre-noise layers (function g). The sensitivity of a function g is defined as the maximum change in output that can be produced by a small change in the input, given some distance metrics for the input and output (p-norm and q-norm, respectively):

$$\Delta_{p,q} = \Delta^g_{p,q} = \max_{\|x - x'\|_p \leq L} \|g(x) - g(x')\|_q.$$

Here, L is the *construction attack bound*, an important parameter of the sensitivity analysis that denotes the size (with p-norm) of the attack that we allow.

Assuming we can compute the sensitivity of the pre-noise layers (addressed shortly), the noise layer leverages the Laplace and Gaussian mechanisms as follows. On every invocation of the network on an input x (whether for training or prediction) the noise layer computes $g(x) + Z$, where the coordinates $Z = (Z_1, \ldots, Z_m)$ are independent random variables from a noise distribution:

- The Laplacian mechanism uses the Laplace distribution with mean zero and standard deviation $\sigma = \sqrt{2}\Delta_{p,1}/\varepsilon$; it provides $(\varepsilon, 0)$-DP.
- The Gaussian mechanism uses the Gaussian distribution with mean zero and standard deviation $\sigma = \sqrt{2\ln(\frac{1.25}{\delta})}\Delta_{p,2}/\varepsilon$; if $\varepsilon \leq 1$ it provides (ε, δ)-DP.

As with most applications of DP, the most difficult task in designing the PixelDP noise layer is *sensitiv-*

ity analysis, i.e., computing $\Delta_{p,q}$ of the pre-noise function, g. In our case, the process depends on where we choose to place the noise layer in the DNN. Recall that the post-processing property of DP [10] ensures that the DNN's final output remains (ε, δ)-PixelDP independent of where the noise is inserted. This gives us great flexibility in placing the noise layer, which opens up a big space of possibilities. While a full investigation of this design space is beyond the scope of this paper, our experience with several options reveals that placing the noise layer earlier in the network simplifies sensitivity analysis of the pre-noise function.

Option 1: Noise in Image. The most straightforward placement of the noise layer is right after the input layer, which is equivalent to adding noise to individual pixels of the image. This case makes sensitivity analysis trivial: g is the identity function, $\Delta_{1,1} = L$, and $\Delta_{2,2} = L$, where L is the construction attack bound.

Option 2: Noise after First Layer. Another option is to place the noise after the first hidden layer, which is usually simple and standard for many DNNs. For example, in image classification, networks often start with a convolution layer. In other cases, DNNs start with a layer of linear operations followed by a non-linearity, of which the rectified linear unit (ReLU) is the most commonly used. These standard initial layers can be analyzed and their sensitivity computed as follows.

For linear layers, which consist of a linear operator with matrix form W, the sensitivity is the matrix norm, defined as: $\|W\|_{p,q} = \sup_{x:\|x\|_p \leq 1} \|Wx\|_q$. Indeed, the definition directly implies that $\|Wx\|_q \leq \|W\|_{p,q}\|x\|_p$. From the linearity of W and the construction attack bound L we directly get that:

$$\Delta_{p,q} = \|W\|_{p,q} L \tag{3}$$

We use the following matrix norms [43]: $\|W\|_{1,1}$ is the maximum 1-norm of W's columns; $\|W\|_{1,2}$ is the maximum 2-norm of W's columns; and $\|W\|_{2,2}$ is the maximum singular value of W.

For a convolution layer, which is linear but usually not explicitly expressed in matrix form, we abstractly reshape the input (e.g. the image) as an $\mathbb{R}^{nd_{in}}$ vector, where n is the input size (e.g. number of pixels) and d_{in} the number of input channels (e.g. 3 for the RGB channels of an image). We write the convolution as an $\mathbb{R}^{nd_{out} \times nd_{in}}$ matrix where each column has all filter maps corresponding to a given input channel, and zero values. So a "column" of a convolution is formed of all coefficients in the kernel that correspond to a single input channel. Reshaping the input does not change the sensitivity.

Option 3: Noise Deeper in the Network. One can imagine adding noise later in the network using the fact that when applying two functions in a row $f(g(x))$ we have: $\Delta_{p,q}^{(f \circ g)} \leq \Delta_{p,r}^{(g)} \Delta_{r,q}^{(f)}$. For instance, ReLU has a sensitivity of 1 for $p, q \in \{1, 2\}$, for a linear layer followed by a ReLU has the same bound on the sensitivity as the linear layer alone. This combination property is leveraged in [9] to bound $\Delta_{2,2}$, and the paper provides bounds for several other types of layers.

However, we find that this approach for sensitivity analysis is difficult to generalize. Layers such as batch normalization [17], which are popular in image classification networks, do not appear amenable to such bounds. Another example is an input transformation commonly used in image processing called image standardization, which makes each image mean zero and unit variance: $x \rightarrow \frac{x - \bar{x}}{\|x - \bar{x}\|_2}$. This transformation makes the sensitivity depend on the input, and on the attack since the attacker can reduce the variance of the image, thus making the denominator smaller.

Our Approach. Based on our experience, we recommend adding DP noise early in the network – where bounding the sensitivity is easy – and taking advantage of DP's post-processing property to carry the sensitivity bound through the end of the network. §4 evaluates and compares designs with noise in the input and noise after the first convolution layer, and reveals interesting accuracy/robustness tradeoffs. We leave the full examination of these tradeoffs for other placements for future work.

3.3 Training Procedure

While achieving PixelDP guarantees requires adding noise only at prediction time, achieving good accuracy requires incorporating the noise layer upon training as well. We use the same loss function and optimization algorithms, such as stochastic gradient descent (SGD), as the original network. However, unless we add noise directly to the image, training with noise raises a significant challenge. Because the magnitude of the added noise scales with the sensitivity of the layers preceding the noise layer, the optimization procedure, such as SGD, must be made aware of the effect of the DNN parameters on that sensitivity. We propose two approaches for dealing with this challenge.

Option 1: Optimize for Sensitivity. One option is to let the SGD optimization balance two competing goals: making better predictions and keeping the pre-noise layers' sensitivity small. For this, we leverage the *reparametrization trick* from [21]. Denote as $w = (w_g, w_h)$ the parameters of the PixelDP DNN $h(g(x)+z)$, where $z \sim \mathscr{L}(0, \frac{\Delta_{p,q}^{w_g}}{\varepsilon})$ is drawn from the PixelDP noise using, say, the Laplace mechanism. And denote g's sensitivity as $\Delta_{p,q}^{w_g}$ to highlight its dependency on the parameters of the first part of the DNN. To perform SGD updates, we need to estimate the following gradient: $\nabla_w \mathbb{E}_{z \sim \mathscr{L}(0, \frac{\Delta_{p,q}^{w_g}}{\varepsilon})} f(g(x) + z)$. The usual Monte Carlo es-

timate of this gradient – $\frac{1}{L}\sum_{l=1}^{L}\nabla_{w,z} f(g(x)+z_l)$, where L is the number of draws from z – often exhibits too high variance to be practical. The reparametrization trick computes $\nabla_w \Delta_{p,q}^{w_g}$ and rewrites $\mathscr{L}(0, \frac{\Delta_{p,q}^{w_g}}{\varepsilon}) = \Delta_{p,q}^{w_g}\mathscr{L}(0, \frac{1}{\varepsilon})$ to estimate as approximately $\frac{1}{L}\sum_{l=1}^{L}\nabla_w f(g(x)+\Delta_{p,q}^{w_g} z_l)$, which leverages the gradient of the distribution's scale (here the sensitivity $\Delta_{p,q}^{w_g}$) and tends to have lower variance. Like Laplace, Gaussian distributions can be rewritten as $\mathscr{N}(\mu,\sigma^2) = \mu + \sigma\mathscr{N}(0,1)$, hence are similarly amenable to reparametrization.

Option 2: Bound Sensitivity. A second option is to force the pre-noise sensitivity to be smaller than a constant (e.g. $\Delta_{p,q} \leq 1$), and fix the noise distribution's standard deviation for the entire process. Since in this case the noise distribution is fixed, we avoid the problem described in the previous paragraph. For $\Delta_{1,1}$ and $\Delta_{1,2}$, we normalize the columns of linear layers and use the regular optimization process with fixed noise variance. For $\Delta_{2,2}$, we run the projection step described in [9] after each gradient step from SGD. This makes the pre-noise layers Parseval tight frames, enforcing $\Delta_{2,2} = L$. For the pre-noise layers, we thus alternate between an SGD step with fixed noise variance, and a projection step. Subsequent layers from the original DNN are left unchanged.

Our Approach. Both options may not always be available. For example, bounding $\Delta_{2,2}$ requires a complex projection described in [9], and such methods may not always be available. On the other hand, computing $\Delta_{2,2}$ is very slow, so since a bounding method exists, using the first method we describe seems expensive in comparison. When both methods exist, our approach is to choose one via empirical comparison: the first method is more flexible, since it allows the optimization method to scale the sensitivity, and does not impose further constraints. Conversely, bounding the sensitivity often impose stronger constraints than strictly necessary (e.g. [9] imposes that all singular values be 1, yet keeping the highest singular value below 1 is sufficient), but can reduce the complexity of the optimization and may impose a regularization that helps in practice.

3.4 Prediction Procedure

While others have proposed adding some noise (not DP-grade) to the training procedure of a DNN to increase its robustness against adversarial examples [12], we know of no published work that incorporates noise in the prediction procedure. Yet, incorporating noise in the prediction procedure is the key to achieving formal robustness guarantees in PixelDP.

Fig. 1 shows the PixelDP prediction procedure, which differs from the traditional procedure in two ways. In a typical network, the prediction procedure will run the network on a given input *x once* to obtain a probability distribution over the labels, and then apply an argmax function to select the label(s) i with the highest one (or more) probability. In PixelDP, the noise layer induces a new probability distribution over the labels and the prediction procedure changes as follows. First, for a given input, we base our prediction on *multiple* runs of the network, each with a new draw from the Laplacian/Gaussian noise. The multiple runs are part of a Monte Carlo computation to *empirically estimate* the noise-induced distribution over the labels.[2] From these runs, we use the network's argmax predictions to construct a histogram of how many times the noisy DNN outputs each label. We add error bounds to the histogram to account for measurement error. We treat each label as a binomial variable (one label against all), and compute one-sided Clopper-Pearson interval (one sided because we care only about the lower bound to the highest probability and upper bound for the second highest probability). We then apply a Bonferroni correction for the number of labels, so that all bounds are valid at the same time. The resulting histogram, denoted p, accurately estimates the noise-induced label distribution (within some confidence interval, say 95%).

Second, instead of giving a prediction for every input, PixelDP only yields *robust predictions*. For this, it implements a *robustness test* (method *isRobust*$(p,\varepsilon,\delta,L,\_)$ in Fig. 1), which uses DP bounds to compute how much an adversary can hope to change each label probability given a maximum attack size L. If the bounds do not overlap, no attack within this size will be able to change the highest probability, the prediction is deemed robust with respect to an attack of size L, and the argmax label over the histogram is returned. The robustness test is given by the following proposition:

Proposition 2. (Generalization of Proposition 1) *Suppose A satisfies (ε,δ)-PixelDP with respect to changes of size L in p-norm metric, and using the notation from Proposition 1 further let $p_j^{ub}(x)$ and $p_j^{lb}(x)$ respectively be the upper and lower bound of the 95% interval of $p_j(x)$. For any input x, if for some $i \in \mathscr{Y}$,*

$$p_i^{lb}(x) > e^{2\varepsilon}\max_{j:j\neq i} p_j^{ub}(x) + (1+e^{\varepsilon})\delta,$$

then the multiclass classification model based on label probabilities $(p_1(x),\ldots,p_Y(x))$ is robust to attacks of p-norm L on input x with probability higher than 0.95.

The proof is similar to the one sketched for Proposition 1 and is detailed in Appendix A.1.

Proposition 2 implies that if, on the original input, the maximum probability is large enough compared to the second highest one, no adversarial example of size L will

[2] Contrary to typical DP settings, we are not protecting the pixels' privacy but bounding changes to the output distribution. It is thus sound to use multiple noise draws to estimate the full output distribution.

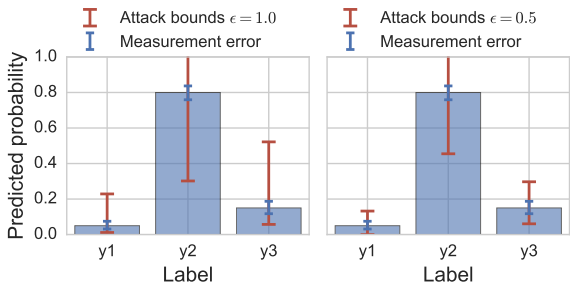


Fig. 2: **Robustness test example.** Histograms for two noise-induced label distributions for a three-label DNN and for two values of ε. Error bars show the measurement error and the DP bounds on adversarial change. *Left:* $(1.0, 0.05)$-PixelDP. The prediction is not robust as the lower-bound for the argmax prediction (y_2) overlaps the upper-bound of y_3. *Right:* $(0.5, 0.05)$-PixelDP. The bounds do not overlap hence prediction is robust.

be able to change the prediction. In that case, we say that the prediction for this image is robust. Fig. 2 illustrates, for two values of ε, the histograms constructed by the prediction procedure to estimate the (ε, δ)-DP label distribution, their corresponding measurement error bars, and the DP bounds of how much an attacker can hope to change the probability of each label by perturbing the input up to a particular attack size, given by ε.

There is another, complementary way to measure the robustness of a prediction. Instead of looking at a binary signal of whether the prediction is robust to an attack of size L, we can compute the maximum attack size that we are robust to for each individual prediction. For some predictions, this will be smaller than L, for others it can be bigger. Recall from 3.2 that the DP mechanisms have a noise standard deviation that grows in $\frac{\Delta_{p,q}}{\varepsilon}$, and from Equation 3 that we express $\Delta_{p,q}$ as a linear function of L (we note $\Delta_{p,q} = \Delta^{mult}_{p,q} L$). For a given standard-deviation σ that is used to train and predict, we can solve for the maximum L for which the prediction is robust: $L_{max} = \max_{L \in \mathbb{R}^+} L$ such that $p^{lb}_i > e^{2\varepsilon} p^{ub}_{j:j \neq i} + (1 + e^{\varepsilon})\delta$ and either $\sigma = \Delta^{mult}_{p,1} L/\varepsilon$ and $\delta = 0$ (for Laplace), or $\sigma = \sqrt{2\ln(1.25/\delta)}\Delta^{mult}_{p,2} L/\varepsilon$ and $\varepsilon \leq 1$ (for Gaussian). The prediction is robust to attacks of maximum size L_{max}. In Fig. 2, the original prediction is not robust to attacks of the original size L (left), but is robust for $\frac{L}{2}$ (right), so the maximum robustness size is in between. We thus implement the robustness test (method *isRobust*$(p, \varepsilon, \delta, L, T)$) by establishing a threshold, T, that we hope to protect against in practice, and returning true iff $L_{max} \geq T$ (**Thresholded Robustness Test**).

We call T the *prediction robustness threshold*, and it is an inference-time parameter that can differ from the *construction attack bound* parameter, Ł, that is used to configure the standard deviation of the DP noise added with each training and prediction. The separation of the two parameters gives engineers the ability to fix one theoretical defense goal when they construct the network (Ł) and then change their mind at prediction time as to what actual attack sizes are permissible to deem predictions robust (T). Of course, training the DNN to defend against a tiny attack size and then imposing a high prediction robustness threshold for its predictions may result in PixelDP making few or zero robust predictions, hence the two parameters must be somewhat aligned.

Our Approach. In practice, the thresholded robustness test provides more flexibility in deciding what attack bounds are acceptable for a given network trained with a fixed ε value. §4 shows that robustness to larger attacks correlates with higher precision, hence some operators may prefer choosing higher values of T. On the other hand, higher values of T imply fewer number of predictions made by the model (because it only makes a prediction when it deems it robust). Moreover, the ability to make many robust predictions for a fixed T very much depends on the number of noise draws performed at prediction time, which in turn impacts prediction performance. Fewer noise draws means greater measurement error bounds, hence looser PixelDP bounds and fewer robust predictions, especially for higher prediction robustness threshold T. More noise draws means slower prediction times. §4.4 shows that in practice a few hundreds draws a sufficient to retain a large fraction of the predictions even for fairly large robustness thresholds.

3.5 Extensions

This section described our initial design of PixelDP for the case of DNN models for image classification. It focused on a few design directions for which we have experience. However, we believe that the idea of using DP – a very general and mature theory – as basis for robust ML has much broader applicability and opens a vaster design space that should be explored in the future.

First, because DP is a very general theory we believe that the PixelDP learning strategy is much more *broadly applicable* than our description implies. The only requirement is that changes to the output of an intermediate DNN layer to adversarial input changes can be bounded in, say, 1- or 2-norm. This is very general and is not specific to DNNs for image classification problems. For example, real-valued outputs (i.e., regression problems) can also be handled, as can discrete-valued inputs (e.g., text data). Furthermore, we can take advantage of plausible structural restrictions on the adversary. For example, if certain parts of the input are secure from adversarial manipulation (e.g., some subset of pixels in an image), then the sensitivity of the DNN layer only need to be determined with respect to changes in the unsecure parts. Few existing defenses against adversarial examples can boast such broad applicability, as most are built upon hacks and tricks specific to image DNNs [18, 7], or leverage classification specific optimization [22, 40].

Dataset	MNIST	CIFAR-10	CIFAR-100
Image size	28x28x1	32x32x3	32x32x3
Labels	10	10	100
Training set	60k	50k	50k
Testing set	10k	10k	10k
DNN	CNN	ResNet	ResNet
Accuracy	99.2%	95.5%	78.6%

Table 1: **Evaluation datasets and baseline models.** Last row shows the accuracy of the baseline models.

p-norm used	DP mechanism	Noise location	Sensitivity approach
1-norm	Laplace	Image	N/A
2-norm	Gaussian	Image	N/A
1-norm	Laplace	1^{st} conv.	$\Delta_{1,1} = 1$
1-norm	Laplace	1^{st} conv.	Reparam. $\Delta_{1,1}$
1-norm	Gaussian	1^{st} conv.	$\Delta_{1,2} = 1$
1-norm	Gaussian	1^{st} conv.	Reparam. $\Delta_{1,2}$
2-norm	Gaussian	1^{st} conv.	$\Delta_{2,2} \leq 1$

Table 2: **Design space explored with our PixelDP DNNs.** For each DNN, we implement defenses for different attack bound norms, requiring different DP mechanisms. We add noise either on the image or after the first convolution. In the latter case, we explore both bounding the sensitivity and using the reparametrization trick for efficient training.

Exceptions would perhaps be [28, 9], though these offer no, or weak, robustness guarantees.

Second, the DP literature, which is very mature at this point, includes many techniques and optimizations that present a *rich design space* for PixelDP. Here we have only used two of the most basic DP mechanisms, both using worst-case sensitivity analysis, hence the amount of noise is very conservative. A promising alternative is to build upon a more fine-tuned technique, such as smooth sensitivity [32] which uses data-dependent sensitivity bounds. That may let us add less noise to "easy to defend" inputs, potentially improving overall accuracy.

Overall, our hope is that this exciting design space will spark interest in the community to develop stronger and more accurate defenses than our initial design affords.

4 Evaluation

As with any defense, the major concerns in PixelDP are (1) how much the defense affects accuracy on *benign, unattacked samples* and (2) whether the defense is effective at increasing accuracy when faced with *maliciously crafted samples*. New in PixelDP is the concern that (3) its repeated invocations of the network at prediction time will affect prediction latency. After describing our methodology for evaluating these aspects (§4.1), we address each concern in turn (§4.2, §4.3, §4.4).

4.1 Methodology

Datasets and Baselines. We evaluate PixelDP on image classification tasks from three standard datasets listed in Table 1. MNIST consists of greyscale handwritten digits, and is the easiest to classify. CIFAR-10 and CIFAR-100 consist of small color images that are each centered on one object of one of 10 or 100 classes, respectively. For each dataset we use existing DNN architectures to train a high performing classifier. The accuracy of the original networks on a held-out testing set (the built in test set has 10k images for each dataset) are also shown in the table and constitute our *baselines* in the evaluation.

On MNIST, we train a Convolutional Neural Network (CNN) with two 5×5 convolutions, with stride 2, and 32 and 64 filters respectively. The convolutions are followed by a fully connected layer with 1024 nodes and a softmax. As a regularizer we use weight decay with a rate of 0.0002. The CNN is trained with a Momentum optimizer with learning rate 0.1 during $15k$ steps, and learning rate 0.01 for $10k$ additional steps.

For both CIFAR datasets, we use a state-of-the-art Wide Residual Network (ResNet) [46]. Specifically, we use the tensorflow implementation of the 28-10 wide ResNet [42] with all the default parameters, including the optimizer and learning rate schedule. As suggested in the documentation, we use $90k$ steps on CIFAR-10 and $70k$ steps on CIFAR-100. The only change we make is to remove the image standardization step. This common step in image processing makes sensitivity input dependent (see §3.2) and had to be removed in our PixelDP models. Interestingly, removing this step also increases the baseline's accuracy, so we kept the change.

PixelDP Models. We make each of the preceding networks PixelDP with regards to 1-norm and 2-norm bounded attacks. Table 2 shows the different noise layers that we used to explore the design space. We use both the Laplace mechanism for sensitivity $\Delta_{1,1}$, and the Gaussian one for $\Delta_{1,2}$ and $\Delta_{2,2}$. We use the projection step from [9] to bound the $\Delta_{2,2}$ sensitivity and implement PixelDP with regards to 2-norm bounded attacks, with noise after the first convolution. We use the same projection parameter $b = 0.0003$ as [9] for the ResNets, and $b = 0.001$ for MNIST. In this case, the sensitivity is not exactly 1, and can actually turn out quite a bit larger. Since the noise standard deviation σ is computed as if the sensitivity was 1, we downscale L accordingly.

Method. We measure various accuracy-related metrics (described shortly) on samples from the same held-out testing set we use to evaluate the baseline undefended models. To evaluate impact of the defense on *benign samples* (§4.2), we use all the samples from the testing set. To evaluate accuracy of the defense on *malicious samples* (§4.3), we run an (until recently) state-of-the-art attack (Projected Gradient Descent [25]) against 1,000

randomly picked samples from the testing set and evaluate how PixelDP improves accuracy on the perturbed versions of those samples compared to the undefended networks and to the networks defended with the best-performing prior defense, the recent Madry defense [29]. The method falls into the "best effort" category, but was shown to withstand even the most recent attacks that broke all the other known defenses [1].

For each evaluation goal – accuracy on bening samples, accuracy on malicious samples, and performance – we explore multiple combinations of the design choices described in §3. Table 2 specifies the various configurations, which were chosen to best reflect tradeoffs in this space. For each configuration, we also vary the prediction robustness threshold T, measuring the minimum attack size against which the prediction must be robust, and the construction attack bound Ł, measuring the attack size that the DNN has been constructed to defend against. Higher values of L correspond to increased levels of noise, since the noise's standard deviation depends on L. In all experiments, we use Laplacian/Gaussian noise drawn from a fixed distribution (corresponding $\varepsilon = 1$ and $\delta = 0.05$), scaled by parameter L.

Accuracy Metrics. The typical metric to evaluate a traditional image classification DNN is *accuracy*, defined as the fraction of images the DNN correctly classifies from a given testing set. In contrast to traditional DNNs, PixelDP DNNs provide a way to reason about the robustness of individual predictions with respect to attacks of a desired size, given by the prediction robustness threshold T. This feature can be used by applications to only take action when predictions reach the desired level of robustness. In such cases, accuracy entails not only the correctness but also the robustness of predictions. For PixelDP DNNs, we thus evaluate three metrics:

1. *Conventional accuracy* $\frac{\sum_{i=1}^{n} isCorrect(x_i)}{n}$, where n is the testing set size and $isCorrect(x_i)$ denotes a function returning 1 if the prediction on test sample x_i returns the correct label, and 0 otherwise.
2. *Robust accuracy* $\frac{\sum_{i=1}^{n}(isCorrect(x_i)\&isRobust(p_i,\varepsilon,\delta,L,T))}{n}$, where $isRobust(p_i,\varepsilon,\delta,L,T)$ is the thresholded robustness test and p_i is the label histogram corresponding to input x_i. Robust accuracy denotes the fraction of the testing set on which PixelDP's predictions are both correct and robust, for a given prediction robustness threshold T.
3. *Robust precision* $\frac{\sum_{i=1}^{n}(isCorrect(x_i)\&isRobust(p_i,\varepsilon,\delta,L,T))}{\sum_{i=1}^{n} isRobust(p_i,\varepsilon,\delta,L,T)}$. It denotes the fraction of all predictions that PixelDP deems robust for threshold T that are also correct. For systems that choose to actuate only on PixelDP's robust predictions, this metric is especially important, because it measures the percentage of correct predictions out of all actions that were taken.

For $T = 0$ all predictions are robust, so both robust accuracy and robust precision are equivalent to conventional accuracy.

Evaluation Focus. This evaluation shows a small snapshot of the investigation we did. For example, while we experimented with multiple DNNs for each dataset, and with both 1-norm and 2-norm attacks, we focus here on one DNN for each dataset (as indicated in Table 1) and 2-norm attacks (which are stronger than 1-norm attacks). Where appropriate we specify substantial changes in results for 1-norm and other networks. In a few specific cases we use ∞-norm attacks, specifically to compare with the Madry defended DNN, which was trained on such attacks. Finally, of the three datasets, we focus most of the evaluation on CIFAR-10, which is what Madry uses. At the end of each evaluation section we generalize our main conclusions to MNIST and CIFAR-100.

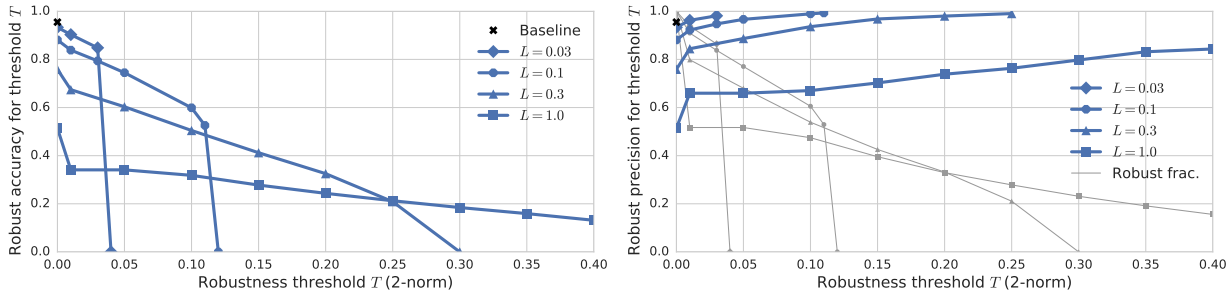
4.2 Accuracy on Benign Samples

We evaluate how PixelDP behaves on benign inputs using our testing set. The key questions are: (1) How do various DP noise levels impact the network's accuracy and robust predictions? (2) How do various design choices from §3 impact accuracy and robust predictions?

Impact of DP Noise. We evaluate the impact of the noise added to meet a particular construction attack bound, L, on conventional accuracy, robust accuracy, and robust precision. We construct four CIFAR-10 ResNets, each satisfying $(1.0, 0.05)$-PixelDP with regard to 2-norm bounded attacks, for four values of Ł: $0.03, 0.1, 0.3, 1$. Higher values of L correspond to larger noise standard deviation σ. Fig. 3 shows the robust accuracy and robust precision for the networks, when robustness is assessed with a growing threshold T. The points on the y axis (x=T=0) show the conventional accuracy.

Focusing on the *conventional accuracy* points on Fig. 3(a), we see that constructing the network for larger attacks (higher L) progressively degrades accuracy. The ResNet with the least noise ($L = 0.03$) reaches 93.3% accuracy, compared to a baseline of 95.5%; increasing noise levels ($L = (0.1, 0.3, 1.0)$) yields 88.1%, 75.9%, and 51.4%, respectively. Defenses against 1-norm bounded attacks show the same trend, but accuracy is higher as the attacks are weaker and thus easier to defend against. For the same sequence of L values, accuracy is 94.1%, 90.5%, 81.8%, and 61.9%, respectively.

On the other hand, looking at the *robust accuracy* lines on Fig. 3(a) ($T > 0$), we observe that as prediction robustness threshold increases, an inversion in the dependency of accuracy on noise appears: PixelDP networks constructed for larger attacks (higher L, hence higher noise), and configured to give more robust predictions (higher T), tend to yield more accurate predictions. For example, the ResNet constructed with $L = 0.03$ has the highest robust accuracy up to $T = 0.03$, but the ResNet



(a) Conventional and Robust Accuracy

(b) Robust Precision

Fig. 3: **Robust metrics with construction attack bound (L) and prediction robustness threshold (T).** *Config:* CIFAR-10 ResNet, 2-norm bounds. *Left:* conventional accuracy (points at $x = T = 0$) and robust accuracy ($x = T > 0$). Conventional accuracy is close to the baseline for low-noise networks (low L). Robust accuracy increases with high-noise networks (high L) that are configured to yield more robust predictions (high T). *Right:* robust precision increases with the prediction robustness threshold, meaning that robust predictions tend to be better. However, the network makes fewer of these predictions as T grows.

constructed with $L = 0.1$ becomes better past that threshold. Similarly, the $L = 0.3$ ResNet has higher robust accuracy than the $L = 0.1$ ResNet above the 0.11 2-norm prediction robustness threshold. As before, observations for 1-norm defenses are consistent with this observation.

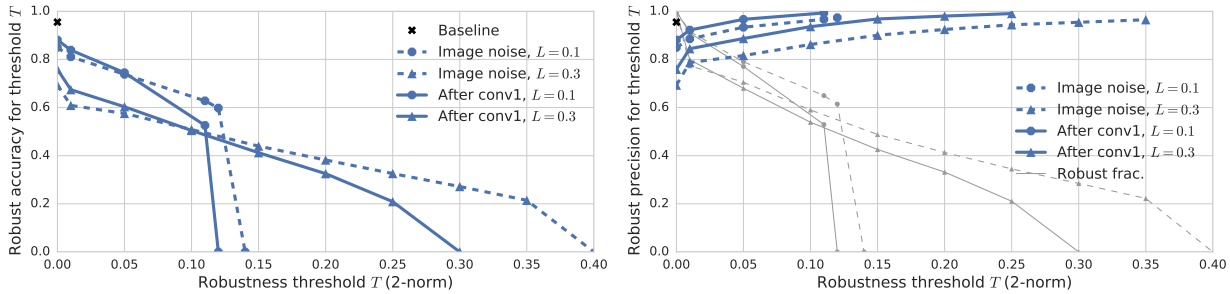
Fig. 3(b) shows that *robust precision* systematically increases with the prediction robustness threshold. At the same time, the fraction of robust predictions (grey lines) decreases with the threshold. This implies that yielding only robust predictions tends to increase precision at the expense of recall. For example, for a prediction robustness threshold of $T = 0.01$, the $L = 0.03$ ResNet makes robust predictions on 90% of the images, reaching 96.3% of precision. The $L = 0.1$ ResNet needs a threshold of 0.05 to beat the baseline, achieving 96.6% accuracy with predictions on 74% of the examples. At the same threshold, the lowest-noise ResNet has 99% precision! Since robustness is linked to the network predicting a higher probability to the correct label, this result suggests that PixelDP probabilities are a good indicator of correctness.

Thus, DP noise introduces a multi-parameter tradeoff between accuracy and robustness. Building more noise into the network hurts accuracy. However, building more noise into the network *and* requiring it to yield more robust predictions enhances the quality of the predictions, even compared to the baseline, undefended networks. On the other hand, requiring the network to yield more robust predictions lowers the number of predictions it can make. We next examine how some of the design choices from §3 impact accuracy and robust predictions.

Noise in Image vs. After First Layer. We show the impact of the noise layer placement. Fig. 4(a) shows that adding noise after the first convolution, as opposed to directly into the image, gives a slightly higher conventional accuracy, but at the cost of lower robust accuracy for higher prediction robustness thresholds. The more noise (higher values of L), the stronger this tradeoff. For example, for $L = 0.1$, adding noise after the first convolution brings conventional accuracy from 85% to 88.1%, but is less robust to attacks with 2-norm bigger than 0.05. For $L = 0.3$ on the other hand, adding noise to the image reduces the accuracy from 75.9% to 69.3%, but gives higher robust accuracy for attacks of 2-norm above 0.1, with a 6 percentage points increase at 0.2 , and a 10 percentage points one at 0.3. Fig. 4(a) shows that this effect is due to the ResNet making a higher number of robust predictions for a given threshold T, which results in higher robust accuracy but worse robust precision. Depending on use-case, both designs can be advantageous.

Optimizing vs. Bounding Sensitivity at Training. An interesting question is what happens if we exclude PixelDP's noise layer from training. After all, PixelDP's robustness guarantees depend only on randomizing predictions. We evaluated this naïve design: the conventional accuracy of the network is a dismal 10% (basically random guesses) even for tiny amounts of noise added for prediction. Hence, if one adds noise to prediction, one must add it to training too. We presented two design options for how to do so effectively: optimizing for vs. bounding pre-noise sensitivity (§3.3). Adding noise into training without either of these optimizations also results in very poor performance. However, at least in CIFAR-10, the choice between the two proposed techniques does not appear significant: at any robustness threshold the mechanisms' robust accuracies differ by at most 1 percentage point. One should choose based on convenience.

Other Datasets. We ran the same experiments on CIFAR-100 and MNIST models but omit the graphs for space reasons. Our main conclusion – that making a DNN more robust (higher L and higher T) hurts conventional accuracy but enhances the quality of its robust predictions – holds in both cases. **CIFAR-100**: For 2-norm bounded attacks, using $L = 0.03$ yields an accuracy of 73.6%, and $L = 0.1$ an accuracy of 66.4%, compared to a baseline of 78.6%. The respective robust precision at the $T = 0.01$ threshold are 84.2% and 77.8%, with pre-



(a) Conventional and Robust Accuracy

(b) Robust Precision

Fig. 4: **Adding noise to the image vs. after the first convolution.** *Config:* CIFAR-10 ResNet, 2-norm bounds. *Concl:* Adding noise after the first convolution is better for low robustness thresholds; adding noise to the image is better for higher thresholds.

dictions on 66.4% and 59% of the dataset. However at $T = 0.1$, neither model makes robust predictions, showing the robust accuracy drops fast with the attack threshold. Adding noise at the image level degrades accuracy and robust precision, but improves robust accuracy. For $L = 0.1$ accuracy is only 57.3%, but the robust accuracy is still 35% at $T = 0.1$. **MNIST**: The CNN can handle higher noise levels. For $L = 0.1$, accuracy is at 99.3%, slightly beating the baseline. Robust accuracy remains high until $T = 0.14$ where it is at 97.2% before dropping to 0. Robust precision quickly reaches 99.7% at $T = 0.07$, with predictions on 98.5% of the testing set. The $L = 0.3$ CNN has an accuracy of 98.5%, and a longer tail of robust accuracy. At $T = 0.14$ it is still bellow the lower noise CNN with an accuracy of 94.4%, but at $T = 0.35$ it still has 77.7% of robust accuracy. The robust precision also goes to 99.9% at $T = 0.25$, with predictions on 88% of the data.

4.3 Accuracy on Malicious Samples

A standard method to evaluate the strength of a defense in this space is to measure the (conventional) accuracy of a defended model on malicious samples obtained by running a state-of-the-art attack against samples in a held-out testing set [28]. We apply the same method, but measure not only conventional accuracy but also the robust metrics that our defense uniquely affords. Key questions are: (1) How does PixelDP's accuracy under attack compare to state-of-the-art defenses? (2) How does PixelDP's accuracy under attack scale with larger datasets?

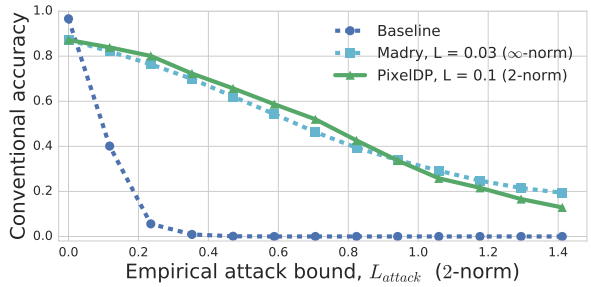
Attack Methodology. We use the Projected Gradient Descent (PGD) attack [25], which until recently was state-of-the-art.[3] We use the PGD attack in a whitebox setting, where the attacker has access to all parameters and hyperparameters of the attacked network, except for the seed for the DP noise generator. We run the attack on the first 1,000 images in each of our datasets' held-out testing set. We quantify the conventional accuracy under attack by counting as correct the adversarial samples that are still classified with their ground truth label. We quantify robust metrics similarly.

As is customary with this methodology [28], we execute PGD in a bounded way, to control the maximum size of the perturbation that the attack can produce on each of the 1,000 samples. We denote this bound as the *empirical attack bound* (L_{attack}) and treat it as a parameter. We measure L_{attack} in either 2-norm or ∞-norm. Although our defense does not currently support ∞-norm attacks, the state-of-the-art defense against which we compare (Madry) was designed for that, hence for an apples-to-apples comparison, we perform both types of attacks on both PixelDP and Madry networks. We report L in $[0,1]$ pixel range: a 2-norm attack of size of 0.39 in $[0,1]$ pixel range corresponds to a 2-norm attack of size 100 in a $[0,255]$ pixel range; an ∞-norm attack of size 0.031 in $[0,1]$ pixel range corresponds to an ∞-norm attack of size 8 in a $[0,255]$ pixel range. Appendix §A.3 contains important but lower-level details about our methodology.

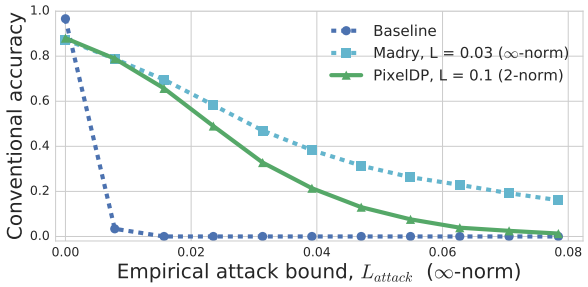
Madry Defense Background. We evaluate accuracy under attack, for increasing empirical attack bound, and compare it with the accuracy of state-of-the-art models provided by the Madry Lab for CIFAR-10 [29]. Those models were developed in the context of ∞-norm attacks, and use an efficient adversarial training strategy to learn from both benign and malicious samples from such attacks [28]. This methodology is best effort and supports no formal notion of robustness for individual predictions, as we do in PixelDP. However, because the Madry models are the best-performing and the only ones still unbroken by the recent attacks [1], they represent a good comparison point for PixelDP. For fair comparison, we evaluate both conventional accuracy (relevant to both PixelDP and Madry) and robust metrics (relevant to PixelDP). We also evaluate both 2-norm and ∞-norm attacks.

Accuracy under Attack Compared to Madry. Fig. 5 compares conventional accuracy of a PixelDP model to that of a Madry model, as the empirical attack bound

[3] A recent arxiv paper [1] seemingly improves upon that attack, but we only learned about it a few days prior to submission. We plan to evaluate it, however our attack appears to satisfy the principles for rigorous attack formulated in that paper.



(a) 2-norm attack



(b) ∞-norm attack

Fig. 5: **Conventional accuracy under attack for PixelDP vs. Madry.** *Config:* CIFAR-10 models. PixelDP model is trained using 2-norm noise $L = 0.1$ after the first convolution. Madry model is optimized with adversarial training using PDG for ∞-norm attacks up to $L = 0.03$. *Left:* 2-norm bounded attacks. *Right:* ∞-norm bounded attacks. *Concl:* PixelDP is on par with Madry for 2-norm attacks; Madry outperforms PixelDP for ∞-norm attacks.

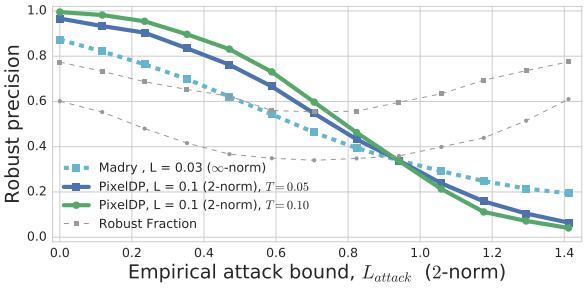


Fig. 6: **PixelDP robust precision vs. Madry accuracy under attack.** *Config:* CIFAR-10 ResNets, 2-norm PDG attack. Same models as in Fig. 5. Madry's robust precision is equivalent to conventional accuracy. *Concl:* PixelDP makes fewer but more correct predictions for up to a certain attack size.

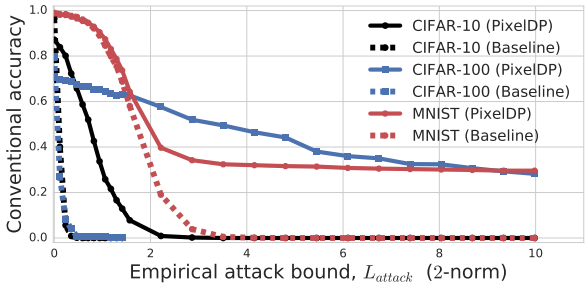


Fig. 7: **Accuracy under attack for all datasets.** *Config:* MNIST CNN, CIFAR-10 and CIFAR-100 ResNet, all with $L = 0.1$ 2-norm attack bounds. *Concl:* PixelDP is much more resilient to attack on CIFAR-100 than PixelDP CIFAR-10.

bound increases and for both 2-norm (left) and ∞-norm (right) attacks. The two models are identical structurally (barring PixelDP-specific modifications). The PixelDP model is configured with noise after the first convolution and a construction attack bound of 0.1. For 2-norm attacks, our model achieves conventional accuracy on par with than Madry. In fact, it is slightly better for 2-norm empirical attack bounds up to 0.82, which corresponds to a 210 2-norm attack in [0-255] pixel range. Both models are dramatically more robust under this attack compared to the baseline (undefended) model. For ∞-norm attacks our model shows much worse conventional accuracy than Madry. This is expected, since ∞-norm attacks are stronger than the 2-norm attacks for which our defense is designed to protect. Still, it is encouraging to see that PixelDP provides some experimental protection (albeit no guarantee) over the baseline even for ∞ attacks.

The robust metrics, computed on the same results and shown in Fig. 6 for two values of the prediction robustness threshold, reflect the benefit to be gained by any application that can leverage our theoretical guarantees to filter out non-robust predictions. Focusing on the robust precision graph, we observe that PixelDP's robust predictions are *substantially more correct* than Madry's predictions up to an empirical attack bound of 0.82. The more conservative the robustness test is (higher T), the more correct PixelDP's predictions are, although it makes fewer of them (Robust Fraction lines). For example, selecting $T = 0.1$ leads to robust precision more than 98.1% for attack sizes up to 0.11. This is even higher than the baseline network on benign samples, whose accuracy is 95.1! However, the prediction rate in that case is extremely low (34-55% for the same attack size interval). For less conservative robustness thresholds, such as $T = 0.05$, PixelDP's robust predictions are still more correct than Madry's predictions (9-21 percentage points more correct up to a 2-bound attack size of 0.82), and can retain a much more meaningful fraction of the predictions (55-73%) for the same attack size interval.

Thus, for applications that can afford to not act on a minority of the predictions, PixelDP's robust predictions under 2-norm attack are substantially more precise than Madry's. For applications that need to act on every prediction, PixelDP offers on-par accuracy under 2-norm attack to Madry's, Interestingly, the first conclusion holds for ∞-norm attacks but the second (as we saw) does not.

Other Datasets. Fig. 7 shows conventional accuracy under attack for a baseline and a PixelDP model for each dataset. Remark that the range of the empirical attack

bound is significantly extended compared to previous graphs, because the PixelDP CIFAR-100 model is significantly more resilient than PixelDP CIFAR-10 models. **MNIST:** Up to attacks of 2-norm 1.5, the baseline and PixelDP models perform similarly under attack, however the PixelDP model provides some level of protection for larger-size attacks. **CIFAR-100:** The baseline model is extremely vulnerable to attack: accuracy drops to zero with an attack of 2-norm 0.47. The PixelDP model starts out with a lower accuracy for no-attack case ($x = 0$), but its accuracy under increasing attack sizes downgrades gracefully, remaining above 50% for attack sizes up to 2-norm 4. Even for an attack of 2-norm 10, PixelDP retains above 35% accuracy. The result surprised us and we investigated enhanced versions of PGD attacks specifically crafted to break PixelDP on CIFAR-100 (e.g., more noise draws, see Appendix A.3). PixelDP showed the same resilience in all cases.

4.4 Performance Overheads

A final concern is PixelDP's computational overhead for training and prediction. PixelDP adds little overhead for *training*, as the only additions are a random noise tensor and sensitivity computations. On our GPU, the CIFAR-10 ResNet baseline takes on average $0.65s$ per training step. PixelDP versions take at most $0.66s$ per training step (1.5% overhead). PixelDP impacts *prediction* more substantially, since it uses multiple noise draws to estimate the probabilities over labels. Making a prediction for a single image with 1 noise draw takes $0.01s$ on average. Making 10 draws brings it only to $0.02s$, but 100 requires $0.13s$, and 1000, $1.23s$. We found that 300 draws were often necessary to properly assess robustness, implying a prediction time of $0.42s$ seconds, a $42\times$ overhead. This is parallelizable, but resource consumption is still a problem. Tighter measurement error bounds may be a future answer.

5 Related Work

Attacks. Since the initial adversarial example attacks on DNNs by Szegedy et al. [41], there has been a flurry of attacks aimed at every new tentative defense mechanism. Attacks have used various norms to quantify the power given to the attacker: the 0-norm [33], 1-norm [19, 6], 2-norm [41, 6, 1], and ∞-norm [12, 44, 6, 1, 25]. State-of-the-art attacks include the PGD method that we use in our evaluation [25], plus a very recent method against defenses obfuscating gradients [1].

Best-effort Defenses. In an arms race with attackers, defenders used multiple heuristics to increase robustness of DNNs, at least empirically. For example, Papernot et al. [34] proposed model distillation, a technique traditionally used for simplifying DNNs, as a defense mechanism. Several other defenses [14, 31, 30] proposed detecting adversarial examples by leveraging their statistical differences to clean examples. Other approaches use input transformations [18, 7] and randomization [13, 8] to obfuscate gradients, or leverage generative models [38, 16, 45]. All these best-effort defenses have been broken, sometimes months after publication [6, 5, 1]. The only empirical defense that still holds is the one from Madry et al. [28], which is based on adversarial training [12] and to which we compare in our evaluation. For the randomization defenses, which may sound similar to PixelDP, the major differences (and flaws in their defenses) are only randomizing training, not prediction, and doing so in an arbitrary way. PixelDP draws its guarantees from randomizing prediction to enforce rigorous DP semantic.

Rigorous Defenses. A recent line of work explores formal verification of DNN safety properties [39, 15, 20, 22, 40]. Unfortunately, none of these techniques scale to complex/large DNNs; this stands in contrast to PixelDP, which scales to real-world-sized DNNs without causing a significant performance drop. Closer to our work, Cisse et al. [9] proposed Parseval networks, a specific type of DNN whose Lipschitz constant is designed to be less than 1, bounding the sensitivity to 2-norm attacks. We could not train an end-to-end Parseval network to perform a direct comparison; the training procedure never converges to non-trivial accuracy. Their published results [9] show faster degradation of accuracy compared to PixelDP DNNs, despite weaker one-step attacks.

Differential Privacy in ML. Previous work studies generalization properties of DP [3]. It is shown that *learning algorithms* that satisfy DP with respect to the training data have statistical benefits in terms of out-of-sample performance. Our learning algorithm is not DP; rather, the predictor we learn satisfies DP with respect to the atomic units (e.g., pixels) of a given test point.

6 Conclusions

We demonstrated a connection between robustness against adversarial examples in machine learning and differential privacy theory. We showed how the connection can be leveraged to develop a rigorous defense against such attacks that is as effective at defending against attacks as today's state-of-the-art best-effort defense. However, our defense additionally provides robustness certification for a meaningful subset of its predictions. Compared to other certified defenses, we believe that our defense presents some unique properties, including broad applicability, scalability to complex and fairly large DNNs and datasets (demonstrated in our evaluation), and a rich design space whose further exploration will likely lead to even more effective defenses.

References

[1] A. Athalye, N. Carlini, and D. Wagner. Obfuscated gradients give a false sense of security: Circumventing defenses to adversarial examples. 2018.

[2] A. Athalye and I. Sutskever. Synthesizing robust adversarial examples. *arXiv preprint arXiv:1707.07397*, 2017.

[3] R. Bassily, K. Nissim, A. Smith, T. Steinke, U. Stemmer, and J. Ullman. Algorithmic stability for adaptive data analysis. In *Proceedings of the forty-eighth annual ACM symposium on Theory of Computing*, 2016.

[4] M. Bojarski, D. D. Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L. D. Jackel, M. Monfort, U. Muller, J. Zhang, X. Zhang, J. Zhao, and K. Zieba. End to end learning for self-driving cars. *CoRR*, 2016.

[5] N. Carlini and D. Wagner. Adversarial examples are not easily detected: Bypassing ten detection methods. In *Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security*, pages 3–14. ACM, 2017.

[6] N. Carlini and D. A. Wagner. Towards evaluating the robustness of neural networks. In *2017 IEEE Symposium on Security and Privacy, SP 2017, San Jose, CA, USA, May 22-26, 2017*, pages 39–57, 2017.

[7] Chuan Guo, Mayank Rana, Moustapha Cisse, Laurens van der Maaten. Countering adversarial images using input transformations. *International Conference on Learning Representations*, 2018.

[8] Cihang Xie, Jianyu Wang, Zhishuai Zhang, Zhou Ren, Alan Yuille. Mitigating adversarial effects through randomization. *International Conference on Learning Representations*, 2018.

[9] M. Cisse, P. Bojanowski, E. Grave, Y. Dauphin, and N. Usunier. Parseval networks: Improving robustness to adversarial examples. In *Proceedings of the 34th International Conference on Machine Learning*, 2017.

[10] C. Dwork, A. Roth, et al. The algorithmic foundations of differential privacy. *Foundations and Trends® in Theoretical Computer Science*, 2014.

[11] I. Evtimov, K. Eykholt, E. Fernandes, T. Kohno, B. Li, A. Prakash, A. Rahmati, and D. Song. Robust physical-world attacks on machine learning models. *arXiv preprint arXiv:1707.08945*, 2017.

[12] I. Goodfellow, J. Shlens, and C. Szegedy. Explaining and harnessing adversarial examples. In *Proceedings of the 3rd ICLR*, 2015.

[13] Guneet S. Dhillon, Kamyar Azizzadenesheli, Jeremy D. Bernstein, Jean Kossaifi, Aran Khanna, Zachary C. Lipton, Animashree Anandkumar. Stochastic activation pruning for robust adversarial defense. *International Conference on Learning Representations*, 2018.

[14] D. Hendrycks and K. Gimpel. Early methods for detecting adversarial images. In *ICLR (Workshop Track)*, 2017.

[15] X. Huang, M. Kwiatkowska, S. Wang, and M. Wu. Safety verification of deep neural networks. In *Proceedings of the 29th International Conference on Computer Aided Verification*, 2017.

[16] A. Ilyas, A. Jalal, E. Asteri, C. Daskalakis, and A. G. Dimakis. The robust manifold defense: Adversarial training using generative models. *CoRR*, abs/1712.09196, 2017.

[17] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International Conference on Machine Learning*, 2015.

[18] Jacob Buckman, Aurko Roy, Colin Raffel, Ian Goodfellow. Thermometer encoding: One hot way to resist adversarial examples. *International Conference on Learning Representations*, 2018.

[19] A. Kantchelian, J. Tygar, and A. Joseph. Evasion and hardening of tree ensemble classifiers. In *International Conference on Machine Learning*, 2016.

[20] G. Katz, C. W. Barrett, D. L. Dill, K. Julian, and M. J. Kochenderfer. Reluplex: An efficient SMT solver for verifying deep neural networks. *CoRR*, 2017.

[21] D. P. Kingma and M. Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[22] J. Z. Kolter and E. Wong. Provable defenses against adversarial examples via the convex outer adversarial polytope. *arXiv preprint arXiv:1711.00851*, 2017.

[23] J. Kos, I. Fischer, and D. Song. Adversarial examples for generative models. *arXiv preprint arXiv:1702.06832*, 2017.

[24] Kos, Jernej and Song, Dawn. Delving into adversarial attacks on deep policies. *arXiv preprint arXiv:1705.06452*, 2017.

[25] A. Kurakin, I. J. Goodfellow, and S. Bengio. Adversarial examples in the physical world. *arXiv preprint 1607.02533*, 2016.

[26] A. Kurakin, I. J. Goodfellow, and S. Bengio. Adversarial machine learning at scale. *arXiv preprint 1611.01236*, 2016.

[27] J. Lu, H. Sibai, E. Fabry, and D. Forsyth. No need to worry about adversarial examples in object detection in autonomous vehicles. *CVPR*, 2017.

[28] A. Madry, A. Makelov, L. Schmidt, D. Tsipras, and A. Vladu. Towards deep learning models resistant to adversarial attacks. *CoRR*, abs/1706.06083, 2017.

[29] Madry Lab. CIFAR-10 Adversarial Examples Challenge. `https://github.com/MadryLab/cifar10_challenge`. Accessed: 1/22/2017.

[30] D. Meng and H. Chen. Magnet: A two-pronged defense against adversarial examples. In *CCS*, pages 135–147. ACM, 2017.

[31] J. H. Metzen, T. Genewein, V. Fischer, and B. Bischoff. On detecting adversarial perturbations. In *Proceedings of the 6th International Conference on Learning Representations*, 2017.

[32] K. Nissim, S. Raskhodnikova, and A. Smith. Smooth sensitivity and sampling in private data analysis. In *Proceedings of the thirty-ninth annual ACM symposium on Theory of computing*, 2007.

[33] N. Papernot, P. McDaniel, S. Jha, M. Fredrikson, Z. B. Celik, and A. Swami. The limitations of deep learning in adversarial settings. In *Proceedings of the 37th IEEE European Symposium on Security and Privacy*, 2016.

[34] N. Papernot, P. McDaniel, X. Wu, S. Jha, and A. Swami. Distillation as a defense to adversarial perturbations against deep neural networks. In *Security and Privacy (SP), 2016 IEEE Symposium on*. IEEE, 2016.

[35] N. Papernot, P. D. McDaniel, A. Sinha, and M. P. Wellman. Towards the science of security and privacy in machine learning. *CoRR*, abs/1611.03814, 2016.

[36] O. M. Parkhi, A. Vedaldi, A. Zisserman, et al. Deep face recognition.

[37] R. Pascanu, J. W. Stokes, H. Sanossian, M. Marinescu, and A. Thomas. Malware classification with recurrent networks. In *Acoustics, Speech and Signal Processing (ICASSP), 2015 IEEE International Conference on*. IEEE, 2015.

[38] Pouya Samangouei, Maya Kabkab, Rama Chellappa. Defense-GAN: Protecting classifiers against adversarial attacks using generative models. *International Conference on Learning Representations*, 2018.

[39] L. Pulina and A. Tacchella. An abstraction-refinement approach to verification of artificial neural networks. In *Proceedings of the 22nd International Conference on Computer Aided Verification*, 2010.

[40] A. Raghunathan, J. Steinhardt, and P. Liang. Certified defenses against adversarial examples. *arXiv preprint arXiv:1801.09344*, 2018.

[41] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus. Intriguing properties of neural networks. In *Proceedings of the 2nd International Conference on Learning Representations*, 2014.

[42] Tensorflow r1.5. Resnet models. `https://github.com/tensorflow/models/tree/r1.5/research/resnet`, 2017.

[43] Wikipedia. Operator norm. `https://en.wikipedia.org/wiki/Operator_norm`, Accessed in 2017.

[44] W. Xu, Y. Qi, and D. Evans. Automatically evading classifiers. In *Proceedings of the 23rd Network and Distributed Systems Symposium*, 2016.

[45] Yang Song, Taesup Kim, Sebastian Nowozin, Stefano Ermon, Nate Kushman. Pixeldefend: Leveraging generative models to understand and defend against adversarial examples. *International Conference on Learning Representations*, 2018.

[46] S. Zagoruyko and N. Komodakis. Wide residual networks. *CoRR*, 2016.

[47] Z. Zohrevand, U. Glässer, M. A. Tayebi, H. Y. Shahir, M. Shirmaleki, and A. Y. Shahir. Deep learning based forecasting of critical infrastructure data. In *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*, CIKM '17. ACM, 2017.

A Appendix

A.1 Proof of Proposition 2

For convenience we state Proposition 2 again, and write the detailed proof.

Proposition. (Basic Robustness Test) *Suppose A satisfies (ε,δ)-PixelDP with respect to changes of size L in p-norm metric, and using the notation from Proposition 1 further let $p_j^{ub}(x)$ and $p_j^{lb}(x)$ respectively be the upper and lower bound of the 95% interval of $p_j(x)$. For any input x, if for some $i \in \mathscr{Y}$,*

$$p_i^{lb}(x) > e^{2\varepsilon} \max_{j:j\neq i} p_j^{ub}(x) + (1+e^{\varepsilon})\delta,$$

then the multiclass classification model based on label probabilities $(p_1(x),\ldots,p_Y(x))$ is robust to attacks of p-norm L on input x with probability higher than 0.95.

Proof. Consider any $\alpha \in B_p(L)$, and let $x' := x+\alpha$. From Equation (1), we have with $p > 0.95$ that

$$p_i(x') \geq (p_i(x)-\delta)/e^{\varepsilon} \geq (p_i^{lb}(x)-\delta)/e^{\varepsilon},$$
$$p_{j:j\neq i}(x') \leq e^{\varepsilon} \max_{j:j\neq i} p_j^{ub}(x) + \delta, \quad j \neq i.$$

Starting from the first inequality, and using the hypothesis, followed by the second inequality, we get

$$\begin{aligned} p_i^{lb}(x) &> e^{2\varepsilon} \max_{j:j\neq i} p_j^{ub}(x) + (1+e^{\varepsilon})\delta \Rightarrow \\ p_i(x') &\geq (p_i^{lb}(x)-\delta)/e^{\varepsilon} > e^{\varepsilon} \max_{j:j\neq i} p_j^{ub}(x) + \delta \\ &> p_{j:j\neq i}(x') \end{aligned}$$

which is the robustness condition from Equation (2). □

A.2 Design Choice: Laplace vs. Gaussian

We study the impact of the DP mechanism used. When training to PixelDP with regards to the 1-norm of the attack, both the Laplace and Gaussian mechanisms can be used after the first convolution, by respectively controlling the $\Delta_{1,1}$ or $\Delta_{1,2}$ sensitivity. Fig. 8 shows that for our ResNet, the Laplace mechanism is better suited to low levels of noise: for $L = 0.1$, it yields a slightly higher accuracy (90.5% against 88.9%), as well as better robust accuracy with a maximum robustness size of 1-norm 0.22 instead of 0.19, and a robust accuracy of 73% against 65.4% at the 0.19 threshold. On the other hand, when adding more noise (e.g. $L = 0.3$), the Gaussian mechanism performs better than the Laplace mechanism, consistently yielding a robust accuracy 1.5 percentage point higher.

A.3 PGD Attack Details

We specify a few important but lower level aspects about our attack procedure for both 2-norm and ∞-norm attacks.

2-norm Attacks: We perform 100 gradient steps and select a step size of $2.5L$ (where L is the 2-norm size of the attack) divided by the number of gradient steps. This ensure that all feasible points within the 2-norm ball can be reach after 100 steps. After each gradient calculation we normalize the gradient tensor with its 2-norm value to ensure numerical stability. Subsequently, we normalize each perturbation enforcing that it stays within the 2-norm ball of the given attack size (projection phase).

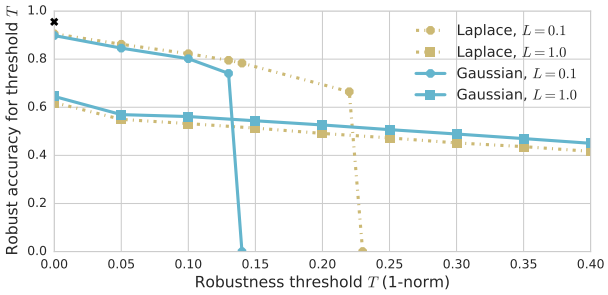


Fig. 8: **Laplace vs. Gaussian.** Robust performance of a ResNet on the CIFAR-10 dataset, for robustness thresholds with regards to 1-norm bounded attacks. The noise standard deviation is computed with $\varepsilon = 1.0$ and construction attack bound L. The Laplace mechanism yields better accuracy for low noise levels, but the Gaussian mechanism is better for high noise ResNets.

∞-norm Attacks: We perform $max(L + 8, 1.5L)$ gradient steps (where L is the ∞-norm size of the attack) and maintain a constant step of size 0.003 (which corresponds to the minimum pixel increment in a discrete $[0, 255]$ pixel range). At the end of each gradient step we clip the size of the perturbation at the granularity of individual pixel values to enforce perturbations within the ∞-norm ball of the given attack size (projection phase).

We investigated various customizations to make PGD attacks more efficient against PixelDP DNNs: We tried multiple noise draws (up to 25) during gradient updates of the adversarial perturbation; we tried stronger targeted attacks selecting the least likely class rather than a randomly selected class (as is customary); we tried higher number of iterations (up to 200), and smaller/larger learning rates. None of the above made any significant difference.