Using Nonlinear Data Transformations to Increase Robustness of ML

Paul Lintilhac
paul.s.lintilhac.th@dartmouth.edu
Dartmouth College

Uttam Rao
uttam.rao.gr@dartmouth.edu
Dartmouth College

Patrick Niccolai
patrick.m.niccolai.gr@dartmouth.edu
Dartmouth College

ABSTRACT

We propose the use of nonlinear data transformations as a potential defense against adversarial examples (evasion attacks) on machine learning classifiers. Linear data transformations, such as linear Principal Component Analysis (PCA), have proven to be effective against the best known evasion attacks for a wide range of classifiers across application domains, but provide only marginal security improvements for convolutional neural networks (CNNs). We investigate strategies for incorporating a variety of alternative data transformations aside from linear PCA, such as kernel PCA, to increase the robustness of machine learning, with specific regard to CNNs. We then empirically evaluate the effectiveness and feasibility of our chosen nonlinear transformations as a protection against evasion attacks using well known datasets (e.g. MNIST) on various models (e.g. papernot-CNN etc.). Lastly, we note some important limitations that apply specifically to non linear transformations and suggest further avenues of research based on our findings.

KEYWORDS

datasets, neural networks, principal components analysis, kernel PCA, evasion attacks, dimensionality reduction

ACM Reference Format:
Paul Lintilhac, Uttam Rao, and Patrick Niccolai. 2021. Using Nonlinear Data Transformations to Increase Robustness of ML. In . ACM, New York, NY, USA, 12 pages.

1 INTRODUCTION

In today's world, machine learning and artificial intelligence are omnipresent in daily life. From consumers to businesses to governments, machine learning is used for a variety of applications: image recognition [18], self-driving vehicles [4], malware detection [29], natural language processing [9], fraud detection, and many others. Through continued innovation and improvement, machine learning classifiers have achieved high accuracy for these applications, allowing for their widespread deployment. Along with this ubiquity, however, comes risk. Machine learning classifiers are often trained on sensitive private data and/or may be used for critical applications, which make them a target for adversaries. In many cases, machine learning itself is being deployed in these adversarial scenarios, in which the adversary benefits from compromising the security or privacy guarantees of an ML system. Devising defenses and protections for machine learning systems in these adversarial environments is important for ensuring that we can continue to trust the security and privacy of machine learning systems.

Adversarial machine learning is certainly not a new concept. In the last twenty years, researchers have exposed the many vulnerabilities of machine learning systems when faced with competent and strategic adversaries [20][14]. Some attacks aim to compromise the privacy of ML systems. For example, an adversary may use a model inversion attack [10] reconstruct training data given a model's parameters or a membership inference attack [24] to determine if a given entry was part of a model's training data. Other attacks seek to damage an ML system's security, which includes the system's confidentiality or integrity. For example, poisoning attacks [7] add corrupted data to the training set of a model to compromise its integrity and cause the model to produce erroneous output at test time. There is also a subarea of research aimed at devising methods to steal ML models [27].

This paper focuses solely on evasion attacks on ML classifiers, in which an adversary strategically adds perturbations to the test data to cause the model to misclassify the input. These perturbed inputs are known as adversarial examples. Szegedy et al. showed in 2014 [15][26] that various machine learning classifiers are susceptible to evasion attacks. Researchers have devised evasion attacks against a plethora of machine learning classifiers including support vector machines (SVM) [6], random forest and other tree based classifiers [3], and neural networks [15][26][8]. These include several efficient methods to generate adversarial examples including various targeted optimization attacks [26] as well as non targeted gradient-based methods such as the fast gradient sign method [15]. These attacks have been demonstrated in academic, laboratory environments or application areas such as voice recognition, image recognition, malware detection, and more. Researchers have also considered various levels of access for the adversary, ranging from full white box, which allows access to any information about the model, to full black box, which only provides a label for a given input. Researchers have also shown that adversarial examples are quite robust across classifiers [26]. In the past few years, these evasion attacks have started to be demonstrated on systems deployed in the real world [13] which underscores the need for effective defenses.

1.1 Problem motivation

Since the discovery of these vulnerabilities in ML classifiers, a few defenses have emerged including adversarial training, ways to identify adversarial inputs, and secure kernel machines [19][22]. While

, ,

effective in specific use cases, most of these proposed defenses are limited to specific types of models or application domains and may increase vulnerability to other attacks, such as privacy attacks like membership inference attacks [3][23][25]. A few of these defenses are detailed in the related work section. PCA-based data transformations, however, have been shown to increase the robustness of machine learning systems against adversarial examples, and represent a highly general solution that can be easily implemented for any algorithm [3]. Bhagogi et. al. showed that data transformations based on PCA act as a form of regularization. Regularization techniques, while coming at a slight cost in terms of accuracy and utility of ML classifiers, do not cause a major trade-off with privacy (unlike adversarial training and gradient masking, they do not improve accuracy of membership inference attacks) [5]. Finally, recent work by Wang et. al. [28] shows that mutual information regularization can help to improve robustness against model inversion attacks. The defense proposed by Bhagoji et al. in 2017 served as the inspiration for this paper. Their defense is applicable across many application domains and ML classifiers (SVM, MLP), as well as being effective against optimal white box adversaries. However, investigations into regularization using PCA defense have been somewhat limited, and have been shown to be less effective against CNNs, which are an important state-of-the-art ML that needs to be addressed in order for this defense to be widely used. Bhagoji et al. also only consider linear data transformations and do not investigate nonlinear transformations such as kernel PCA, autoencoders, or clustering. Our motivation for this paper was to address these limitations by investigating non linear transformations as a defense against evasion attacks, with specific regard to CNNs.

1.2 Challenges

We faced a number of challenges over the course of this project, most of which revolved around debugging code and the hardware limitations of our personal machines. We decided to fork the source code from Bhagoji et al.'s paper, then modify and supplement it with our implementations of our chosen non linear transformations. However, the initial fork of the Bhagoji code caused errors for almost every command we tried to run for every model. The issues with the original code base included mismatched function signatures between where they were defined and called, completely missing functions, missing datasets and much more. Additionally, the original code base was written in python2 which has since become deprecated and not supported on many remote virtual environments. We upgraded the code base to python3 and fixed the bugs relating to both the defenses and attacks.

Some of the most challenging issues to fix were configuring the code to properly utilize GPUs, fixing an issue with code that outputs the accuracy value of the models, and creating and fixing functions to build a CNN model with reduced dimensions using the Lasagne python library. Fixing these issues took hours of debugging and creating new functions to fix what ended up being issues with only a few lines in the original code. In some cases, we were able to go through the commit history of the original Bhagoji code and find what changes they made which caused the code to break. We also had trouble finding the GTSRB dataset as the link to download the original dataset from Institut für Neuroinformatik was not working. We finally found the dataset on Kaggle.

Although fixing bugs in the original codebase was challenging and frustrating at times, the real challenge was dealing with the long runtimes and lack of computational power we had on our personal machines. For this paper, we used a CNN which fits the specifications defined by Papernot et al. [21] (full specifications are in the preliminaries section). The Papernot-CNN is a 9 layer CNN and takes a long time to train, in some cases taking multiple hours per epoch on a MacBook Pro with 32 GB or RAM. The CNN models need to be trained for at least 50 epochs to reach an accuracy near 99 percent. This runtime issue was exacerbated when using kernel PCA and finding the best hyperparameters for it as we found it took almost one to two orders of magnitude longer to converge than linear PCA. We were able to get around these issues while testing our code in a few ways. First, we used Dartmouth's high performance computing clusters, Andes and Polaris, which have much better performance and compute power than our local machines. We also used a smaller dataset (10 percent of whichever dataset we were testing) and a small number of dimensions to run our code and iron out any bugs. We were also able to find some trained CNN models in older commits of the original Bhagoji codebase. In the end, however, we did run all of our defenses without using any of these compromises to get our final results.

1.3 Main contributions

We investigate the use of non linear data transformations as a defense against evasion attacks. We first reproduce the results reported by Bhagoji et al. which showed that linear transformations can significantly reduce the success of an adversary executing an evasion attack. We then investigate the use of non linear transformations, specifically kernel PCA, show that it also can be used as a successful defense, and that is significantly better than linear transformations when the perturbation added to an adversarial example is large. Although the improvement is smaller, this result still holds for CNNs. Our defense is applicable across many ML classifiers, including multi layer perceptrons and CNNs. Our defense should also applicable across different datasets in different application areas. We also show that our defense holds for a powerful white box adversary that is aware of our defense and can perform an optimal attack. Lastly, we show that by changing the number of reduced dimensions, the ML model designer can control the trade off between model utility and security. We leave readers with a clean code base to reproduce both our results and those of Bhagoji et al.

1.3.1 Our defense. We implement alternative data transformations aside from linear PCA. Specifically we consider non linear dimensionality reduction methods such as kernel PCA. Kernel PCA represents a very general class of transformations. Kernel PCA leverages the fact that non-linear, positive semi-definite kernel functions applied to the features can be represented as an inner product in some reproducing kernel Hilbert space (RKHS). Therefore, kernel PCA can be applied to these transformed features, allowing for the orthonormal eigenbasis to be some set of nonlinear functions. Kernel PCA can be considered a general regularization method. We

consider various families of non linear kernels. These include radial basis functions, which we chose because the non-linearity RBF models were shown in [17] to be significantly resistant to adversarial examples, and have been used for kernel density estimation in successful adversarial detection defenses. We also optimize the kernel hyperparameters within the RBF family. Our code base implements two defenses using kernel PCA. The first, called the reconstruction defense transforms input data back into the original space of the undefended model before testing. This allows the defense to not require any additional training of a model before the defense is applied. The second defense, called the retrain defense, retrains a new model with the reduced number of dimensions from the kernel PCA. This model targets both the classification and training phases and is significantly stronger than the reconstruction defense.

1.3.2 Evaluation. We demonstrate the usefulness of our defenses using:

- various supervised learning ML classifiers, including multi layer perceptrons and Papernot-CNNs
- the fast gradient and fast gradient sign methods to generate adversarial examples
- multiple benchmark and real world datasets including the MNIST handwritten digits dataset [16] and the GTSRB traffic sign dataset

The main performance metric we consider is the reduction in accuracy of the classifier in the presence of a white box adversary. We also look at any reduction in the precision or overall utility of the classifier in the presence of the defense.

Our key findings are that using kernel PCA is better than linear PCA for a high level of perturbation and is highly general across ML classifiers. Although the defense is effective, there is still a utility-security tradeoff. Our defenses is generally model agnostic and should be dataset agnostic. They can be used along with many other defenses against adversarial examples. Our results can be reproduced using the source code at https://github.com/paullintilhac/cosc189-project.

2 PRELIMINARIES

Most of the notation and terminology in this paper is either commonly used and widely known or immediately defined. The neural network models in this paper may be referred to as MLP (multi layer perceptron), FC (fully connected), or CNN (convolutional neural networks). Both MLP and FC refer to the same type of model which is a fully connected two layer network. Any CNN discussed in this paper meets the specifications described by Papernot et al. [21], which has the following architecture: 2 convolutional layers with 32 filters, a max pooling layer, 2 more convolutional layers with 64 filters, another max pooling layer, 2 fully connected layers with 200 neurons each, and softmax layer with the same number of neurons as the number of classes in the dataset (for MNIST this is 10, for GTSRB this is 43). All neurons in the hidden layers use the sigmoid function.

3 RELATED WORK

In their paper "Improving the Robustness of Machine Learning via Data Transformations", Bhagoji et. al. find that applying a PCA dimensionality reduction on the features before passing it to the machine learning model can greatly reduce the impact of adversarial examples, while only marginally reducing the utility of the model. In some cases, they found that applying PCA dimensionality reduction can actually increase the validation accuracy. By analyzing the effects of PCA on the objective function of an SVM, they show that the technique amounts to a new regularization term, which effectively pushes the input data further away from the decision boundary, and acts as a method of denoising the data. Since regularization and denoising are known to reduce overfitting of machine learning, it makes sense that the effects of adversarial examples are reduced, and could actually increase the validation accuracy, and even improve robustness to privacy attacks such as membership inference attacks. Thus, we thought dimensionality reduction techniques were a promising path towards increasing robustness of machine learning.

One our main inspirations for using dimensionality reduction with RBF-based Kernel PCA were from the recent work on detecting adversarial examples using Gaussian Mixture Models, such as in "Towards Robust Decetion of Adversarial Examples" by Pang et. al. (2017) [19]. This paper implemented a kernel density estimator using a GMM, in order to detect adversarial examples that lay far from the center of mass of the input data. The further points are away from the main centers of mass, the lower their estimated likelihood, and if they are too far away from these centers of mass, they will be classified as anomalies, and likely adversarial examples. This work is based largely on the well known hypothesis that adversarial examples are a consequence of the fact that most input data often lies on a low dimensional manifold within the input space, and therefore perturbations away from this manifold can easily bring them outside of the input distribution, despite being only small perturbations in the full input space.

Shortly after this work on detection of adversarial examples, Carlini et. al. argued that in "Adversarial Examples are not easily detected" that many previous defenses, while effective for a black box adversary on the MNIST data, were not as effective for adversaries who were aware of the defense, or the distribution of the training data. In order to bypass the KDE defense described by Yang, they perform a strategic attack that penalizes the the likelihood of being detected by such a method by creating their own detector based on the training data, and then adding the distance from the training set as a penalty during the generation of adversarial examples. This effectively bypassed their attack, though this attack relies on complete knowledge of both the defense and the training set.

Carlini also criticized the Bhagoji paper, claiming that their defense was easily outperformed by a standard undefended convolutional neural network, and therefore does not add anything as a defense in practice. It should be noted that this finding does not agree with our findings about convolutional neural networks: as a benchmark, we tested the 9-layer convolutional neural net proposed by Papernot

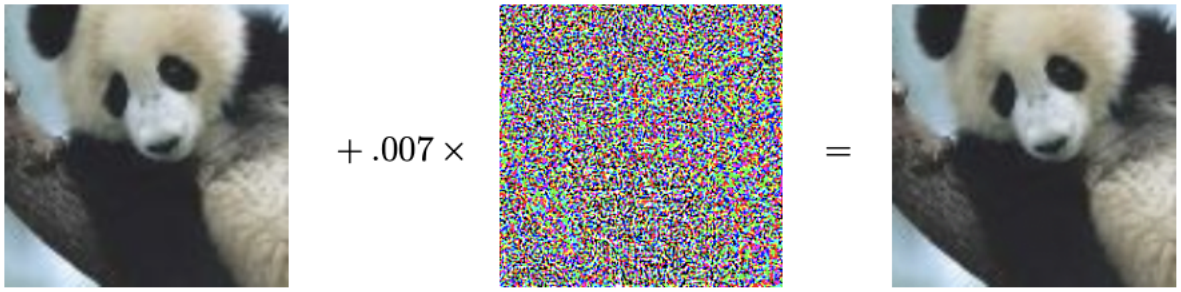

Figure 1: The image on the left is x, the original example. The middle image is the perturbation, $sgn(\nabla_x J(x, y, \theta))$. ϵ is 0.007. The right image the adversarial example, $x_i + \epsilon sgn(\nabla_x J(x, y, \theta))$. This is an example FGSM attack taken from [15], in which the panda is classified as a gibbon after the perturbation.

et. al. (2016) that achieved 98.6

Our work using Kernel PCA is related to recent work on manifold learning, which is closely related to kernel methods. It is well known that dimensionality reduction can aid in machine learning models, but often the data lies on a non-linear or "curvy" manifold. This can create problems, as for example the linear interpolation between two points in the manifold may lie outside the manifold [1]. Linear techniques such as PCA do not work well in these settings. In the context of security in machine learning and in particular adversarial examples, this creates a major weakness for machine learning models whose training data lies on a non-linear, low-dimensional manifold. Algorithms that use iterative methods to find adversarial examples are prone to generate examples that lie outside of this manifold, near a decision boundary, and near from a training point a sense based on the L2 distance, but far away from the manifold in terms of its similarity to other data points.

4 THREAT MODEL

In this section we describe the threat model under consideration. For a successful evasion attack, the adversary wants to create an adversarial example that will cause a model f to make a misclassification when it is tested on that example. The adversary does this by taking a benign input x and the true class y and uses an algorithm A to create a perturbed example $x' = A(x)$. There are two types of attacks: non targeted and targeted. In the former, the adversary does not care what class the model decides the inputed example is as long as it is not the true class. In a targeted attack, the adversary tries to cause a misclassification to a specific class. Of course, for binary classifiers, these two attacks are the same as there are only two classes. An additional constraint is that the adversarial example produced by algorithm A should differ as little as possible from the benign example. The spaces for all examples are vector spaces and p-norms are used as a metric. The Bhagoji paper considers both types of attacks and our source code consequently allows us to run both attacks, but we only considered the non targeted attacks for this project.

Next, we consider how much the adversary knows about our model. In the Bhagoji paper, the authors consider adversaries with three different levels of knowledge. The first is the weakest adversary where they do not have information on the classifier's architecture or hyperparameters, but do have some access to the training dataset. The second adversary knows the model architecture, hyperparameters, and training data, but is not aware if there are any defenses protecting the model. The third adversary is a full white box adversary where they know everything about the model including weights, have access to all the data, and are aware of any defenses protecting the model. Although the first two adversaries may be more realistic in a practical setting, in this project we only consider the third, strongest adversary as an effective defense against this adversary would also be effective against the other two.

We use attacks existing in literature to implement the capabilities of the white box adversary. Specifically, we consider the fast gradient sign method (FGSM) proposed by Goodfellow et al. in [15]. This method is a gradient based attack against neural networks that is non targeted. The perturbation in this attack is calculated in a single step so it is extremely efficient. This attack uses the max norm metric ℓ_∞. The adversarial examples are created by adding a perturbation proportional to the sign of the gradient of the cost function used for the model, $J(x, y, \theta)$. As already defined above, x is the original benign example, y is the real class of x, and θ is the weights of the network.

$$\mathbf{x'}(\epsilon)_\mathbf{i} = \mathbf{x_i} + \epsilon \mathbf{sgn}(\nabla_\mathbf{x} \mathbf{J}(\mathbf{x}, \mathbf{y}, \theta))_\mathbf{i} \tag{1}$$

The gradient in the above equation is easy and efficient to calculate using backpropagation. A demonstration of this attack from Goodfellow et al.'s paper is show in Figure 1. Writing the above equation differently, we can see the perturbation is controls by the parameter epsilon:

$$\|\mathbf{x} - \mathbf{x'}(\epsilon)_\mathbf{i}\|_\infty = \max_\mathbf{i} |\epsilon \mathbf{sgn}(\nabla_\mathbf{x} \mathbf{J}(\mathbf{x}, \mathbf{y}, \theta))_\mathbf{i}| = \epsilon \tag{2}$$

If we had done this project in isolation the FGSM attack would be sufficient, but we want to be able to compare our results to those in the Bhagoji et al. paper as well as other attacks on different

Attack	Classifier	Constraint	Intuition
Fast Gradient Sign Method [15]	DNNs	ℓ_∞	gradient based constant scaling for each feature - smallest epsilon
Fast Gradient [3]	DNNs	ℓ_2 norm	gradient based move in direction of smallest epsilon
Optimal attack on SVMs [3]	linear SVM	ℓ_2 norm	try to push examples close to the classification boundary
Carlini attack [8]	DNNs	ℓ_2 norm	optimization based attack to minimize epsilon

Table 1: Summary of various attacks [3]

classifier types in literature which are constrained by different norms. Bhagoji et al. consider an optimal attack on SVMs as well as an optimization based attack detailed by Carlini et al. in [8], both of which are constrained by the ℓ_2 norm. They propose a modification to the FGSM attack to constrain the method by the ℓ_2 norm instead of the max norm. They call this modified attack the fast gradient method (FG):

$$\mathbf{x}'(\epsilon) = \mathbf{x} + \epsilon \frac{\mathbf{sgn}(\nabla_{\mathbf{x}} \mathbf{J}(\mathbf{x}, \mathbf{y}, \theta))_{\mathbf{i}}}{\|\mathbf{sgn}(\nabla_{\mathbf{x}} \mathbf{J}(\mathbf{x}, \mathbf{y}, \theta))_{\mathbf{i}}\|_2} \tag{3}$$

Here, in equation 3, epsilon indicates the magnitude (the ℓ_2 norm) of the perturbation. Table 1 [3] shows a summary of the various attacks.

5 METHODOLOGY

Our approach uses Kernel Principal Components analysis with an RBF kernel in order to apply a non-linear data transformation to the training data before training the model. Similar to Bhagoji et. al., we then apply this same data transformation to any new test inputs before classifying them. As compared with standard linear PCA, the basic intuition behind this approach is that, rather than projecting our data onto a linear subspace of the original data, we project it onto a particular low-dimensional manifold defined by closeness of the training points to each other [2].

We first need to review aspects of PCA, as well as Mercer's Theorem and the kernel trick, in order to understand what our algorithm is doing. Recall that in linear PCA, we are extracting the top eigenvectors (those with largest eigenvalues) of the matrix $C = \frac{1}{N}\sum_{i=1}^{N} x_i x_i^T$ Note that in this representation, rather than defining the correlation matrix element-wise in terms of the correlation of each pair of features, the summation over observations is instead happening in the outer layer, with each observation contributing to the entire kernel matrix with its outer-product. This representation allows us to easily compare with kernel PCA,

$$K = \frac{1}{N}\sum_{i=1}^{N} \phi(x_i)\phi(x_i)^T \tag{4}$$

Where $\phi(x)$ is a "feature map" for the positive semi-definite symmetric kernel such that $\phi(x)^T\phi(y) = K(x, y)$ for all y, which is guaranteed to exist according to Mercer's theorem. Any similarity metric of this type can be evaluated easily for any pair of points in the input space as $K(x_i, x_j)$, and can also be represented as an inner product in some high (possibly infinite) dimensional feature space, called the Reproducing Kernel Hilbert Space (RKHS). This allows us to apply well known algebraic and geometric methods to analyze these features. In the case of the RBF kernel, we have the following similarity metric defined between any two points:

$$K(x_i, x_j) = exp(-\gamma||x_i - x_j||^2) \tag{5}$$

where $||*||$ represents the euclidean 2-norm. Note that this kernel is in fact infinite-dimensional; this can be seen by expanding the exponent in terms of the series representation, so that we have all possible combinations of powers of x_i and x_j. The intuition behind this is that in general, the similarity to any arbitrary point cannot be represented as a linear combination of similarities to other points, so in order to represent arbitrary similarities, we need an arbitrary number of dimensions. Mercer's theorem only tells us that there exists a (not necessarily unique) $\phi(x)$ that represents the kernel K in some RKHS, i.e. it maps each x to a functional $\phi(x)$ such that $K(x, y) = \phi(x)^T\phi(y)$ for all y (Mercer's Condition).

Mercer's Theorem is just an existence theorem, and it does not actually tell us the form of $\phi(x)$ in general. In fact, there could be many different forms of $\phi(x)$ that satisfy this condition. In practical machine learning applications that depend primarily on taking an inner product of features, such as support vector machines or principal components analysis, we can always evaluate inner products of feature vectors by evaluating kernel function between any two points directly as $k(x_i, x_j)$. In the case of Kernel PCA, the form of can explicitly defined as the similarity between a given point x_i and all other points in the input space, as we detail below.

When restricted to the N points in our input space, we can define something called the empirical kernel map, where instead of $\phi(x)$ being an abstract, infinite-dimensional function in our Hilbert space (remember that (multiple feature maps can satisfy Mercer's condition), $\phi_N(x)$ an N-dimensional vector of similarities to each other point in our training dataset, i.e.

$$\phi_N(x) = [K(x_1, x), K(x_2, x), ..., K(x_N, x)] \tag{6}$$

The empirical kernel mapping is intuitively based on the closeness of a new point to each other point in our training set. Recall that in order for $\phi_N(x_i)$ to be consistent with our kernel matrix for all i, we require that the dot product $\phi(x_i)^T\phi(x_j)$ equal $K(x_i, x_j) = \mathbf{K}_{ij}$ exactly. But our initial guess for $\phi_N(x)$ does not satisfy this property, as the similarity between two given points is not exactly the same as comparing their similarities to all other points in the space (though we would expect it to be somewhat representative, as two points that are similar to the same things are in a sense similar). This mismatch can be seen by taking a 2-point dataset in 2D with points

at (0,0) and (1,1), and observing that the value of $K(x_i, x_j)$ is $e^{-2\gamma}$, while the dot product $\phi(x_1)^T\phi(x_2) = 2e^{-2\gamma}$. It can be shown that the empirical feature map can be transformed using the kernel matrix as

$$\phi_m^*(x) = \mathbf{K}^{-1/2}[K(x_1, x), K(x_2, x), ..., K(x_N, x)] \tag{7}$$

in order to ensure that inner products are correct for all points in our dataset. [17]

To verify this, we can see that

$$\begin{aligned}\phi_N^*(x_i)^T\phi_N^*(x_j) = \sum_{k=1}^{N}(\mathbf{K}^{-1/2}K(x_k, x_i))^T\mathbf{K}^{-1/2}K(x_k, x_j) \\ = \mathbf{K}_i^T\mathbf{K}^{-1}\mathbf{K}_j = \mathbf{K}_{ij}\end{aligned} \tag{8}$$

,
where $\mathbf{K}_i$ is the i-th row of our kernel matrix. The transformation just ensures that the inner product of their similarities agrees with their similarity measure in the kernel matrix, i.e. for all points in the training set.

A complication arises when we note that, while the original observations x_i are assumed to have mean zero and therefore their covariance matrix is centered, our transformed features $\phi_N^*(x_i)$ do not necessarily have mean zero, and therefore our kernel matrix K is not guaranteed to be centered. We can deal with this by computing the centered kernel matrix,

$$\mathbf{K'} = \mathbf{K} - \mathbf{1}^N\mathbf{K} - \mathbf{K}\mathbf{1}^N + \mathbf{1}^N\mathbf{K}\mathbf{1}^N \tag{9}$$

,
Where $\mathbf{1}^N$ is an NxN constant matrix with each entry equal to $\frac{1}{N}$.

Kernel PCA is then performed by taking the eigen-decomposition of the centered matrix, such that $\mathbf{K'} = \mathbf{U}\Lambda\mathbf{U}^T$, where $\mathbf{U}$ is our unitary matrix of eigenvectors, and is Λ our diagonal matrix of eigenvalues.

Once we have our centered kernel matrix and have computed its eigendecomposition, our goal is now to be able to take a new point and represent it in terms of the eigenvectors of $\mathbf{K'}$. First, we consider how to represent the new data point in our RKHS. While we know how to do this for any point in our input space using the empirical feature map as shown above, this calculation is only exact for points in our training set. In order to extend this to out-of-sample data points, we can use the Nystrom Extension of the kernel map: $\phi(y) = \Lambda^{1/2}\mathbf{U}^T[K(x_1, y), K(x_2, y), ..., K(x_N, y)]$. Note that if our eigenvalue matrix is not full-rank, then we must take instead the pseudo-inverse of $\Lambda^{1/2}$, $(\Lambda^{1/2})^+$. While the details of why this approximation works are out of the scope of this paper, the basic idea is that the new point y can be mapped to its feature representation (similarities to each of the N training points) using weighted linear combinations of the maps for each training point. To evaluate the projection of any point in the feature space $\phi(y)$ onto the k-th eigenvector $\mathbf{U}^k$, we take

$$\begin{aligned}\mathbf{U}^k\phi(y) &= \mathbf{U}^k\Lambda^{-1/2}\mathbf{U}^T[K(x_1, y), K(x_2, y), ..., K(x_N, y)] \\ &= (K^{-1/2})^k[K(x_1, y), K(x_2, y), ..., K(x_N, y)] \\ &= \sum_{i=1}^{N} u_i^k\phi(y)^T\phi(x_i) = \sum_{i=1}^{N} u_i^k K(y, x_i)\end{aligned} \tag{10}$$

Similarly, if we want to project our new data point onto the top k eigenvectors, we have

$$\begin{aligned}\mathbf{U}\phi(y) = \sum_{i=1}^{k} u_i^k\phi(y)^T\phi(x_i) = \sum_{i=1}^{k} u_i^k K(y, x_i) \\ = (\mathbf{K}^{1/2})^+[K(x_1, y), K(x_2, y), ..., K(x_N, y)]\end{aligned} \tag{11}$$

What are these k terms $u_i^k K(y, x_i)$ in the sum? They are simply the i-th coordinate of the eigenvector u^k, representing its closeness to the i-th point in the dataset, weighted by the input point's closeness to each training point.

Given the projection of our new point onto the kernel's eigen-basis with dimension k, we can then proceed with our machine learning task by simply training our model on the transformed training points, and test the model on new points by transforming the point using the above equation and then feeding it to the model. We refer this as the KPCA-Retrain algorithm.

As an alternative to the above algorithm, we might want to invert the projection of each point onto the kernel eigen-basis back into the input space. In this case, we can view the algorithm as a denoising algorithm, in the sense that it reconstructs the original data point using a smaller number of principal components. We refer to this as the KPCA-Recons algorithm. While this may be trivial for regular PCA, complications arise when we are using nonlinear dimension reduction with kernel PCA. In particular, the inverse for the empirical kernel map is not guaranteed to be unique.

As an example with the specific case of the RBF kernel, suppose there were two points in our dataset, which lie in a 3D feature space. For any third point in the space, there are an infinite number of points in the space that lie at the same distance from both of the other points, namely those on a circle centered around the axis that connects the two points. Therefore, any point y laying on that circle will have the same feature mapping $\phi(y)$. Obviously, this is a particularly problematic example, and it can be shown that the problem becomes less pronounced as the number of data points increases relative to the dimensionality of the feature space [1]. Various methods have been proposed to address this problem both in the general case and for the RBF kernel. The one we implemented for our KPCA-Recons algorithm is the kernel ridge regression as described in [5] and implemented using the scikit-learn package in python.

It is crucial in this algorithm to properly set the hyperparameter γ used in the definition of the Gaussian kernel. If is too large, then the similarity between points will be high for points that are very

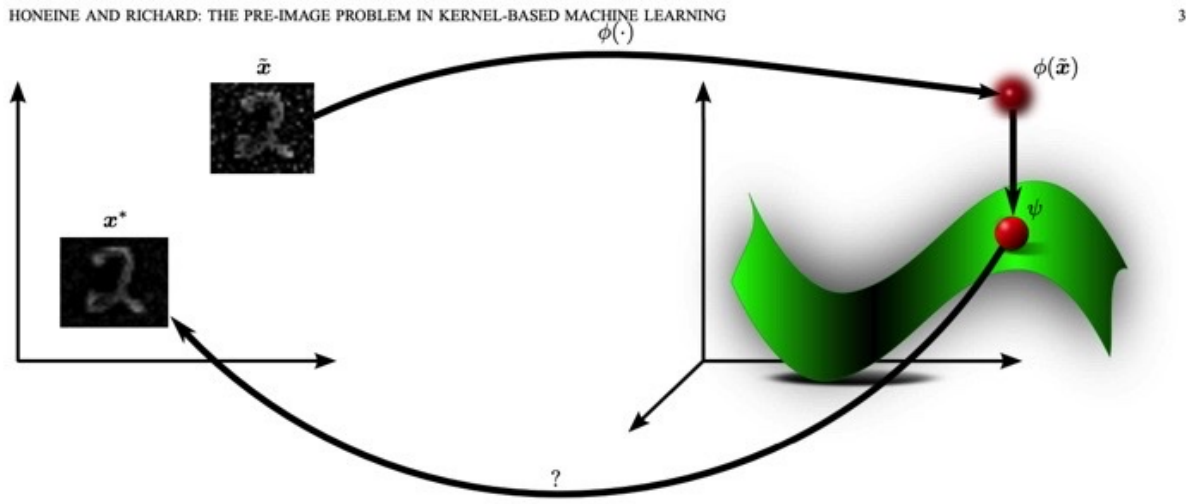

Fig. 1. Schematic illustration of the pre-image problem for pattern denoising with kernel-PCA. While dimensionality reduction through orthogonal projection is performed in the feature space (right panel), a pre-image technique is required to recover the denoised pattern in the input space (left panel).

Figure 2: Visualization of the feature pre-image problem that arises in KPCA-Recons. [12]

far away from each other, and will miss the underlying structure of the manifold defined by the training points. If is too small, then we run the risk of having a low similarity for any points that are not in an immediate vicinity of each other, modeling the training data as a set of isolated points. In order to optimize, we used a grid search. For the MNIST dataset, we found that = 0.0325 yielded the best results in validation.

6 EVALUATION

Github link: https://github.com/paullintilhac/cosc189-project

6.1 Dataset Description

All experimentation of kernel PCA was done using the MNIST dataset. The MNIST dataset consists of 28 x 28 pixel grayscale images, which are handwritten representations of digits 0-9. We chose this dataset because it was the dataset used in the Bhagoji et al. paper [3] that we were attempting to expand on, as well as being a dataset commonly used across security and privacy of machine learning research. So, the models we analyzed were classification models that took an image as input and classified it as one of the ten digits. This dataset consists of 60,000 training examples and 10,000 test examples. However, for certain experiments such a large dataset caused issues with using too much memory and taking too long, as we will explain in further detail later, so in these cases we used 1/10th of the dataset.

6.2 Machine Learning Algorithms

We tested kernel PCA on machine learning models trained with two different algorithms. First we tested on a neural network, we chose to use the FC100-100-10 multilayer perceptron from Szegedy et al. [26]. This is a standard neural network with an input layer, two hidden layers consisting of 100 neurons, and an output layer.

We also experimented on a convolutional neural network from Papernot et al. [21]. This CNN has an input layer, followed by two convolutional layers each consisting of 32 filters, then a max pooling layer, then two convolutional layers each consisting of 64 filters, then a max pooling layer, then two fully connected layers each consisting of 200 neurons, then the output layer.

6.3 Defenses

We analyzed the effects of kernel PCA on two different defenses, the retrain defense and the reconstruction defense. The retrain defense works by training a new model on data that has had a data transformation, in our case either linear PCA or kernel PCA, applied to it. The first step is to transform the training data and train a new model on the transformed data. Then, during classification each example is transformed using the same data transformation and then inputted into the model.

Unlike the retrain defense, the reconstruction defense does not train a new model on transformed training data. Instead it transforms the data inputted to the model during classification, and then transforms it back to the data's original space. This has the benefit of only using the principal components of the data while also not needing to retrain the model, because the data should be essentially the same as before the dimensionally reduced data, but with the same feature set as the un-transformed data.

6.4 Evaluation Metrics

Our experimentation was primarily concerned with the test accuracy of models after they had been attacked by the Fast Gradient attack and then defended by linear PCA or kernel PCA defenses. A

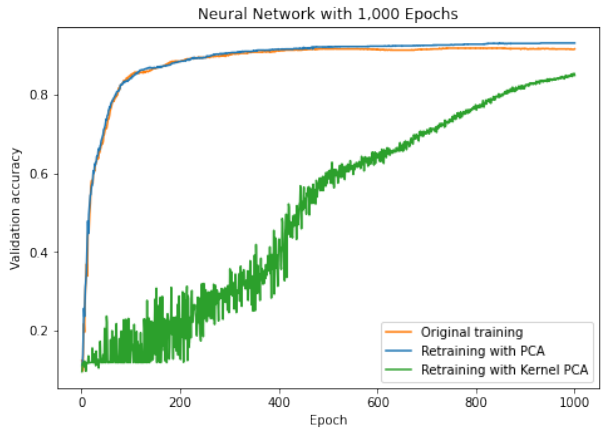

Figure 3: Validation accuracy of an unattacked neural network model trained over 1,000 epochs using no data transformation, linear PCA, and Kernel PCA.

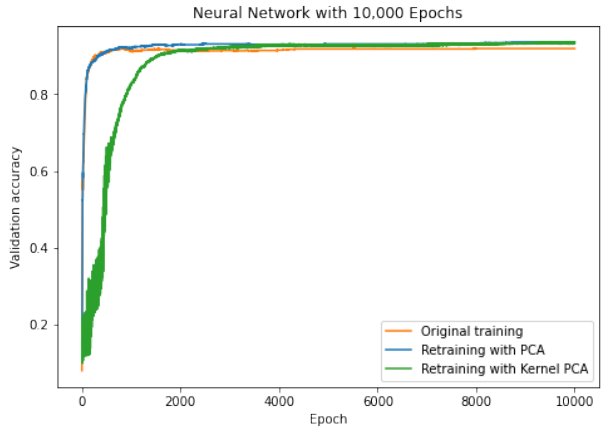

Figure 4: Validation accuracy of an unattacked neural network model trained over 10,000 epochs using no data transformation, linear PCA, and Kernel PCA.

model's test accuracy is the proportion of correctly classified examples, out of examples that were not in the set the model was trained on. Test accuracy is a good metric for determining the robustness of a model because it measures how well the model performs on new data points which the model did not directly learn from. Particularly, we were interested in how a model's test accuracy was affected as the epsilon of the attack increased, where epsilon is the size of the perturbation allowed for the adversarial example. Essentially, a higher epsilon means a stronger attack.

6.5 Experiment Results

6.5.1 Unattacked Models. Before experimenting on attacked models, we decided to analyze the test accuracy of a model that was not attacked, but which was trained using kernel PCA. We did this in order to ensure that the kernel PCA data transformation did not have any negative effects on a model's accuracy while not in the presence of an attack. The results of this experiment can be

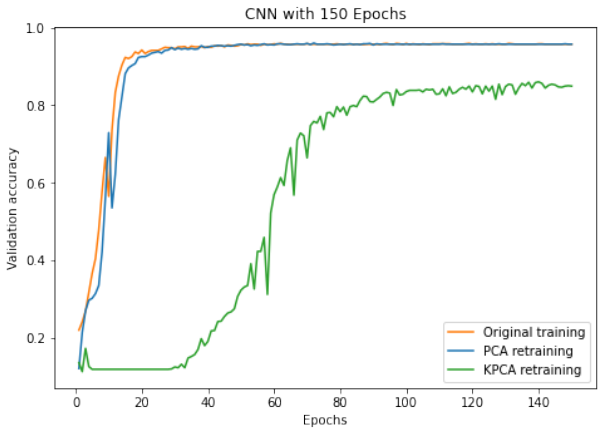

Figure 5: Validation accuracy of an unattacked CNN model trained over 150 epochs using no data transformation, linear PCA, and Kernel PCA.

seen in Figures 3 and 4. These figures graph the test accuracy of an unattacked neural network model using no data transformation, linear PCA, and kernel PCA, as the number of training epochs is increased. An epoch is one pass through the training data while the model is being trained. For both the no data transformation and the linear PCA, the model reaches a high test accuracy with a low number of epochs. Interestingly, this is not the case for the model trained with kernel PCA. Figure A shows that even with 1,000 epochs, the model trained with kernel PCA does not quite reach the accuracy of the other two. However, the kernel PCA model does seem to be slowly converging with the other two, and this is confirmed in Figure B. Figure B shows that even though a model trained with kernel PCA takes a higher number of epochs to achieve a high test accuracy, it will eventually converge with the models trained on no data transformation and linear PCA.

It is worth mentioning that these experiments were performed on the small dataset, which is 1/10th of the MNIST dataset. This is part of the reason that kernel PCA takes longer to converge to a high validation accuracy. With a larger dataset kernel PCA would need a smaller number of epochs to achieve a high accuracy, however it would still be worse than no data transformation and linear PCA for a very low number of epochs.

We then performed a similar experiment on a CNN. The results of this experiment can be seen in Figure 5. Unfortunately, training a CNN requires a lot of time and memory, so training one with thousands of epochs was not feasible, so we only used 150 epochs. The results of this experiment shows that the model trained with no data transformation and the model trained with linear PCA achieve a high test accuracy with a low number of epochs, while the model trained with kernel PCA takes a higher number of epochs to achieve a high test accuracy. While we were not able to train a CNN with enough epochs for the kernel PCA trained model to converge with the other two, the results of the CNNs trained with 150 epochs appear to be similar to the results of the experiment on a neural

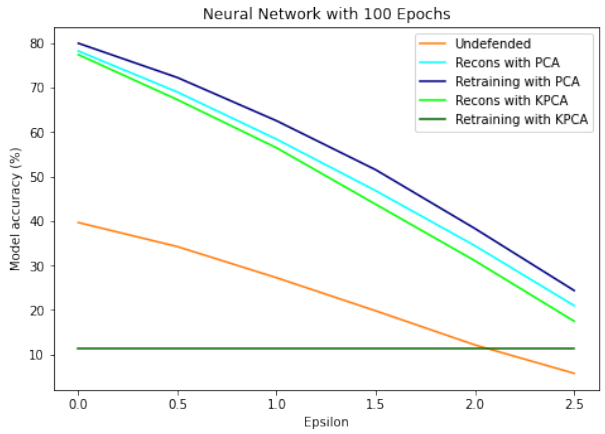

Figure 6: Test accuracy of a neural network model attacked with the FG attack. This figure graphs the test accuracy of the undefended model, and the model with the reconstruction and retrain defenses for both linear PCA and kernel PCA.

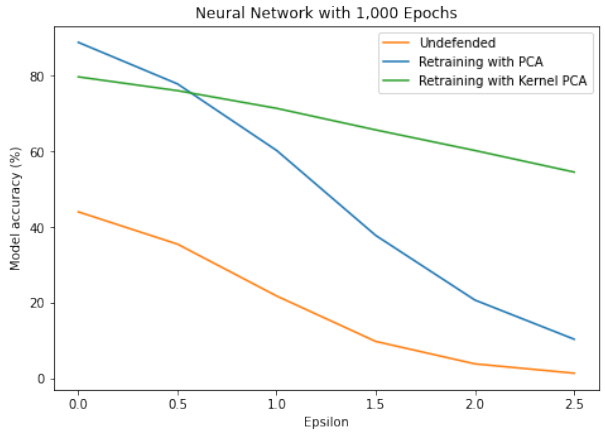

Figure 7: Test accuracy of a neural network model attacked with the FG attack. This figure graphs the test accuracy of the undefended model, and the model with the retrain defenses for both linear PCA and kernel PCA.

network with 1,000 epochs, so we believe that given enough epochs the kernel PCA trained CNN would eventually converge.

6.5.2 Attacked Models. Then we experimented on models which were attacked using the Fast Gradient attack discussed previously. To test the effect of increasing the epsilon of the attack, we performed the FG attack with six different epsilons: 0.001, 0.5, 1, 1.5, 2, 2.5. First, we used a neural network with 100 epochs, and we tested four defenses: reconstruction with linear PCA, retrain with linear PCA, reconstruction with kernel PCA, and retrain with kernel PCA. The results of this experiment can be seen in Figure 6. It is worth mentioning that for this experiment, 1/10th of the MNIST dataset was used, which is why the undefended model has a low accuracy even for low epsilons. As expected, the defenses using linear PCA

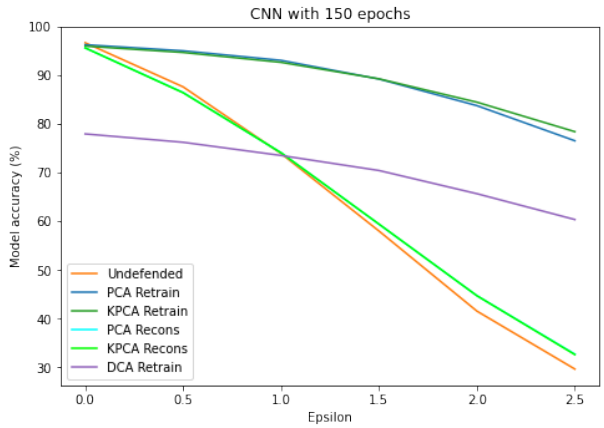

Figure 8: Test accuracy of a CNN attacked with the FG attack. This figure graphs the test accuracy of the undefended model, and the model with the retrain defenses for both linear PCA and kernel PCA.

performed well, having a higher test accuracy than the undefended model. What was unexpected was that the retrain defense using kernel PCA performed poorly, being even worse than the undefended model for all epsilons other than 2.5. However, based on our previous experiments, the kernel PCA retraining requires a large number of epochs to be effective, so we then performed the same experiment but with 1,000 epochs rather than 100.

The results of this experiment can be seen in Figure 7. As expected, the undefended model and the model defended with linear PCA perform roughly the same as when the model only had 100 epochs. On the other hand, the model defended with kernel PCA performs much better, even beating linear PCA for all epochs above 0.5. As the epsilon increases, kernel PCA performs increasingly better than linear PCA. These results demonstrate that under the right conditions, i.e. training a model with a high number of epochs, kernel PCA does perform better than linear PCA for neural network models.

Then, we performed a similar experiment but with a CNN trained with 150 epochs. The results of this experiment can be seen in Figure 8 Interestingly, unlike the previous experiments the defenses using linear PCA and kernel PCA performed very similarly. In fact, only with epsilons 2 and 2.5 did the two retrain defenses perform differently. The kernel PCA did perform better in these cases, but only by a small margin. The reconstruction defenses using linear PCA and kernel PCA performed almost exactly the same for all epsilons.

This experiment also included the DCA, or Discriminant Correlation Analysis, data transformation [11]. DCA is a data transformation designed for pattern recognition which maximizes pairwise correlations across pairs of classifications. The results of a retrain defense using the DCA data transformation can be seen in Figure 8. This defense was less effective than both linear PCA and kernel

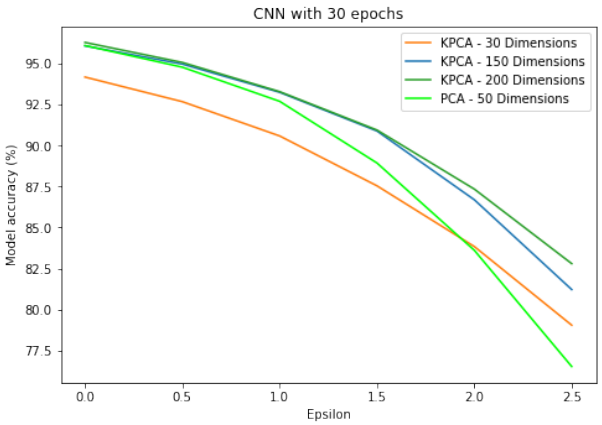

Figure 9: Test accuracy of a CNN attacked with the FG attack. This figure graphs the test accuracy of a model defended with the kernel PCA retrain defense, as the number of dimensions used in the defense is increased.

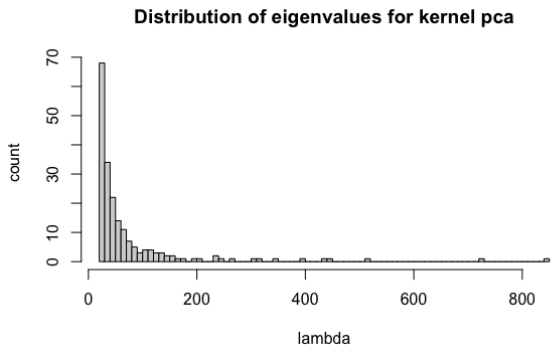

Figure 10: Distribution of Eigenvalues of the Kernel Matrix.

PCA retrain defenses for all epsilons. It was also worse than the undefended and reconstruction defense models for epsilons 1 and lower.

6.5.3 Number of dimensions on kernel PCA. Kernel PCA, like linear PCA, is a dimensionality reducing data transformation. With our kernel PCA we are able to specify what number of dimensions the data should have after undergoing the data transformation. So, we decided to test how a different number of dimensions would affect the robustness of a retrain attack with kernel PCA. The results of this experiment can be seen in Figure 10. We only tested three different numbers of dimensions: 30, 150, and 200. 200 dimensions creates a better defense than the lower numbers of dimensions, especially when the epsilon of the attack is above 1.5. This is a promising result because previous experiments in this paper used 50 dimensions for kernel PCA, so it is possible that increasing the dimensions would result in an even better performance of kernel PCA compared to linear PCA.

Figure 10 also plots linear PCA with 50 dimensions. Although we were unable to compare linear PCA with 50 dimensions to kernel PCA with 50 dimensions, these results are promising because for epsilons 2 and above, kernel PCA with 30 dimensions performs better than linear PCA with 50 dimensions. This is promising because increasing the number of dimensions only improves the performace of kernel PCA, so kernel PCA with 50 dimensions would likely outperform linar PCA with 50 dimensions.

7 LIMITATIONS

Although we do have some promising results, there are a number of limitations to our work and further avenues to be investigated that could strengthen our paper. Firstly, as mentioned earlier in the related works and threat model sections, Carlini et al. [8] proposed an optimization based attack that is stronger than the FGSM and FG attack we consider in this paper. Even though some of Carlini's claims about the effect of adversarial examples on CNNs are not backed up by our findings, to properly evaluate our defense, we need to test it against this attack.

Additionally, all of the results presented in this paper were from the MNIST handwritten digits dataset or a smaller subset of it. While we do have some results for the GTSRB dataset, we did not have time to run enough experiments to present any meaningful results from it in this paper. Using our defense on other more complicated datasets such as CIFAR-10 would allow us to better evaluate our defense. In the Bhagoji paper, they test the linear PCA defense on the HAR dataset which is from a different application domain (activity detection based on sensor data), which allows them to make the claim that their defense is dataset agnostic. Although we do not have any reason to believe this result would not hold with our kernel PCA defense, we still need to run more experiments on other datasets.

The main reason we were not able to test many datasets was due to our computational and time limitations. Optimizing the hyperparameters and running our defense with kernel PCA adds significantly more time than linear PCA to an already long training process for the CNN. Additionally, calculating the adversarial examples in our tests for kernel PCA requires allocating tens of gigabytes of memory, which is not possible on our local machines. We were able to get around this by using Dartmouth's high computing clusters, using a small number of dimensions, as well as using a subset of the MNIST dataset while running our code to weed out any bugs.

We also do not investigate how our proposed defense holds against other attacks beyond evasion attacks, such as model inversion or membership inference attacks, which is an important feature to compare against other proposed defenses.

Lastly, one of the major advantages of linear PCA is that it is easy to implement and does not drastically change the training time of a model. Though kernel PCA does outperform linear PCA based on our results, an ML system designer would need to keep in mind the additional computing resources that may be needed.

8 FUTURE WORK

More testing: In order to properly evaluate our defense, we still need to run more experiments. Those including running the state of the art attacks on the defense (such as the Carlini l-2 attack), and running the tests on a more complex image dataset, such as CIFAR-10 and/or GTSRB. These are recommendations based on the Carlini paper.

Resolving outstanding questions: The Carlini paper claims that the Bhagoji defense does not work as well as any standard, undefended CNN model. This does not agree with our findings, which showed that the papernot CNN, which for all intents and purposes is "standard", is indeed quite vulnerable to adversarial examples, even non-strategic, black box attacks. Carlini did not give any supporting evidence to this claim, nor any references. Thus it would be good to understand better where that claim came from, and whether there is a CNN model that is indeed impervious to attacks. In addition, our findings also disagree with one of the statements at the end of the Bhagoji paper. While they did not present any results on a CNN in their paper, they claimed without evidence or references that their PCA defense did not work well against an attack on that model. To the contrary, our findings show that the PCA defense works quite well. In order to ensure that our results are correct and reproducible, we need to resolve this open question.

This manuscript has focused primarily on the results and implementation for the kernel-pca algorithm, and introduced some of the theory underlying kernel-pca. However, we could go deeper into understanding the mechanics behind how the kernel-pca algorithm is working. For example, it would likely give a better intuitive understanding of the algorithm if we had a better understanding of the distribution of eigenvalues, as well as the behavior of the first few eigenvectors. It would be instructive to see how this compares to the case of regular PCA, as we would expect the eigenvectors to be focused on specific clusters and local regions within the images. In particular, it would be good to see how the number of significant eigenvalues depends on the number of classes represented in the dataset.

The algorithm in this paper could be extended in several ways. First, we could test the impact of changing the distance metric that is used in the RBF kernel. For example, we could use the Laplacian (L-1 distance) kernel, or the γ -norm kernel, where γ is significantly larger than 2. It would be interesting to see how well this defends against different kinds of attacks, in particular fast gradient attacks that use different norms for their constraint on the adversarial perturbation.

In addition, our algorithm could be generalized to non-linear discriminant analysis. Linear Discriminant Analysis is similar to PCA, except that instead of finding orthogonal linear combinations of features that explain the maximum amount of variance, we look for orthogonal linear combinations that separate the different classes as well as possible. LDA has a similar generalization using kernel methods to Kernel PCA, called Kernel Fischer Discriminant Analysis. It would be useful to test the the effects of using the RBF kernel to dimension-reduce the data in a way that incorporates the dependency between the features and the labels, and see how this affects the utility and robustness to adversarial examples, as well as privacy attacks such as Model Inversion attacks, Membership Inference Attacks, or attacks on the fairness of a model.

Finally, we believe one of the strength of this defense to be that it appears to have no major drawbacks in terms of its utility, or its robustness to privacy attacks. While other defenses such as adversarial training are effective, they significantly impact the utility of the model. Similarly, it has been shown that by Shokri et. al. (2019) that several state-of-the-art defenses against adversarial examples increase the vulnerability of a model to membership inference attacks. All of the defenses analyzed in that paper are either based on generating adversarial examples, or they are verification-based defenses that alter the objective function of the classifier, but none of them are based on applying data transformations to the data as a separate pipeline before training. Since dimension reduction and regularization techniques tend to reduce over-fitting, our defense is likely robust to these kinds of attacks. Given the difficulty of securing models against adversarial examples and the trade-offs that exist between security and privacy for many defenses, we believe the true strength of this defense lies in its general applicability as a balanced approach that protects utility, security, and privacy. But a more comprehensive analysis of our defense against these privacy attacks is required to demonstrate this claim.

9 CONCLUSION

Our results show that the KPCA-retrain algorithm significantly outperforms the PCA defense for the MLP model. However, this is only the case when the model being defended was trained with a high number of epochs. This is because kernel PCA requires a high number of epochs to achieve a high validation accuracy, even for an unattacked model. However, with a high number of epochs kernel PCA will converge with linear PCA at a high validation accuracy.

For the CNN models we looked at, we saw that with a high number of epsilons, kernel PCA does outperform linear PCA, but only by a small margin. Interestingly linear PCA works fairly well, which seems to contradict findings from Bhagoji et al. [3]. We also find that for CNNs, for both linear PCA and kernel PCA, the retrain defenses perform much better than the reconstruction defenses, which only slightly improve the robustness of the model compared to the undefended model.

We also find that when specifying the number of dimensions to which the kernel PCA reduces the dimensionality of the data, a higher number of dimensions leads to a stronger defense, especially as the epsilon of the attack increases. Also, for high epsilons kernel PCA can outperform linear PCA, even when the kernel PCA uses a lower number of dimensions.

REFERENCES

[1] Pablo Arias. 2007. Constrained Pre-Image for Kernel PCA. Application to Manifold Learning. https://www.colibri.udelar.edu.uy/jspui/bitstream/20.500.12008/2873/1/Ari07.pdf

[2] Pablo Arias, Gregory Randall, and Guillermo Sapiro. 2007. Connecting the Out-of-Sample and Pre-Image Problems in Kernel Methods. In *2007 IEEE Conference*

on Computer Vision and Pattern Recognition. 1–8. https://doi.org/10.1109/CVPR.2007.383038

[3] Chawin Sitawarin Arjun Nitin Bhagoji, Daniel Cullina and Prateek Mittal. 2017. Enhancing Robustness of Machine Learning Systems via Data Transformations. (2017). arXiv:arXiv:1704:02654

[4] Mrinal R. Bachute and Javed M. Subhedar. 2021. Autonomous Driving Architectures: Insights of Machine Learning and Deep Learning Algorithms. *Machine Learning with Applications* 6 (2021), 100164. https://doi.org/10.1016/j.mlwa.2021.100164

[5] Gökhan Bakir, J. Weston, Bernhard Schölkopf, S. Thrun, and L. Saul. 2004. Learning to Find Pre-Images. *Advances in Neural Information Processing Systems, 449-456 (2004)* (03 2004).

[6] Battista Biggio, Igino Corona, Davide Maiorca, Blaine Nelson, Nedim Šrndić, Pavel Laskov, Giorgio Giacinto, and Fabio Roli. 2013. Evasion Attacks against Machine Learning at Test Time. In *Machine Learning and Knowledge Discovery in Databases*, Hendrik Blockeel, Kristian Kersting, Siegfried Nijssen, and Filip Železný (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 387–402.

[7] Battista Biggio, Blaine Nelson, and Pavel Laskov. 2012. Poisoning Attacks against Support Vector Machines *(ICML'12)*. Omnipress, Madison, WI, USA, 1467–1474.

[8] N. Carlini and D. Wagner. 2017. Towards evaluating the robustness of neural networks. (2017). arXiv:arXiv:1608.04644

[9] Ronan Collobert, Jason Weston, Léon Bottou, Michael Karlen, Koray Kavukcuoglu, and Pavel Kuksa. 2011. Natural Language Processing (Almost) from Scratch. *J. Mach. Learn. Res.* 12, null (nov 2011), 2493–2537.

[10] Matt Fredrikson, Somesh Jha, and Thomas Ristenpart. 2015. Model Inversion Attacks That Exploit Confidence Information and Basic Countermeasures. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security* (Denver, Colorado, USA) *(CCS '15)*. Association for Computing Machinery, New York, NY, USA, 1322–1333. https://doi.org/10.1145/2810103.2813677

[11] M. Hagihat, M. Abdel-Mottaleb, and W. Alhalabi. 2016. Discriminant Correlation Analysis: Real-Time Feature Level Fusion for Multimodal Biometric Recognition. (2016). arXiv:doi: 10.1109/TIFS.2016.2569061

[12] P Honeine and C Richard. 2011. Preimage Problem in Kernel-Based Machine Learning. (2011). https://doi.org/doi:10.1109/MSP.2010.939747

[13] Hossein Hosseini, Baicen Xiao, and Radha Poovendran. 2017. Deceiving Google's Cloud Video Intelligence API Built for Summarizing Videos. (03 2017).

[14] Ling Huang, Anthony D. Joseph, Blaine Nelson, Benjamin I.P. Rubinstein, and J. D. Tygar. 2011. Adversarial Machine Learning *(AISec '11)*. Association for Computing Machinery, New York, NY, USA, 43–58. https://doi.org/10.1145/2046684.2046692

[15] Jonathon Shlens Ian Goodfellow and Christian Szegedy. 2015. Explaining and harnessing adversarial examples. (2015). arXiv:arXiv:1412.6572

[16] Yan LeCun, Corinna Cortes, and Christopher Burges. [n.d.]. The mnist database of handwritten digits. ([n. d.]). http://yann.lecun.com/exdb/mnist/

[17] Sanparith Marukatat. 2016. Kernel Matrix Decomposition via Empirical Kernel Map. 77, C (jul 2016), 50–57. https://doi.org/10.1016/j.patrec.2016.03.031

[18] Jan Matuszewski and Adam Rajkowski. 2020. The use of machine learning algorithms for image recognition. In *Radioelectronic Systems Conference 2019*, Piotr Kaniewski and Jan Matuszewski (Eds.), Vol. 11442. International Society for Optics and Photonics, SPIE, 412 – 422. https://doi.org/10.1117/12.2565546

[19] T Pang, C Du, Y Dong, and J. Zhu. 2017. Towards robust detection of adversarial examples. (2017). arXiv:arXiv:1905.10291v3

[20] Nicolas Papernot, Patrick McDaniel, Arunesh Sinha, and Michael P. Wellman. 2018. SoK: Security and Privacy in Machine Learning. In *2018 IEEE European Symposium on Security and Privacy (EuroS P)*. 399–414. https://doi.org/10.1109/EuroSP.2018.00035

[21] N Papernot, P D McDaniel, X Wu, S Jha, and A Swami. 2016. Distillation as a defense to adversarial perturbations against deep neural networks. (2016). arXiv:arXiv:1511.04508

[22] P Russu, A Demontis, B Biggi, G Fumera, and F Roli. 2016. Secure kernel machines against evasion attacks. (2016). https://doi.org/10.1145/2996758.2996771

[23] R Sahay, R Mahfuz, and A El Gamal. 2019. Combatting adversarial attacks through denoising and dimensionality reduction: A cascaded autoencoder approach. (2019). arXiv:arXiv:1812.03087

[24] Reza Shokri, Marco Stronati, and Vitaly Shmatikov. 2016. Membership Inference Attacks against Machine Learning Models. *CoRR* abs/1610.05820 (2016). arXiv:1610.05820 http://arxiv.org/abs/1610.05820

[25] L Song, R Shokri, and P Mitall. 2019. Privacy Risks of Securing Machine Learning Models against Adversarial Examples. (2019). arXiv:arXiv:1905.10291v3

[26] C Szegedy, W Zaremba, I Sutskever, J Bruna, D Erhan, I Goodfellow, and R Fergus. 2014. Intriguing properties of neural networks. (2014). arXiv:arXiv:1312.6199

[27] Florian Tramèr, Fan Zhang, Ari Juels, Michael K. Reiter, and Thomas Ristenpart. 2016. Stealing Machine Learning Models via Prediction APIs *(SEC'16)*. USENIX Association, USA, 601–618.

[28] T Wang, Y Zhang, and R Jia. 2020. Improving Robustness to Model Inversion Attacks using Mutual Information Regularization. (2020). arXiv:arXiv2009.05241

[29] Weilin Xu, Yanjun Qi, and David Evans. 2016. Automatically Evading Classifiers: A Case Study on PDF Malware Classifiers. https://doi.org/10.14722/ndss.2016.