CSCE496 Deep Learning: Final Project Report

Cale Harms, Colton Harper, and Krishnamohan Sunkara

May 3, 2018

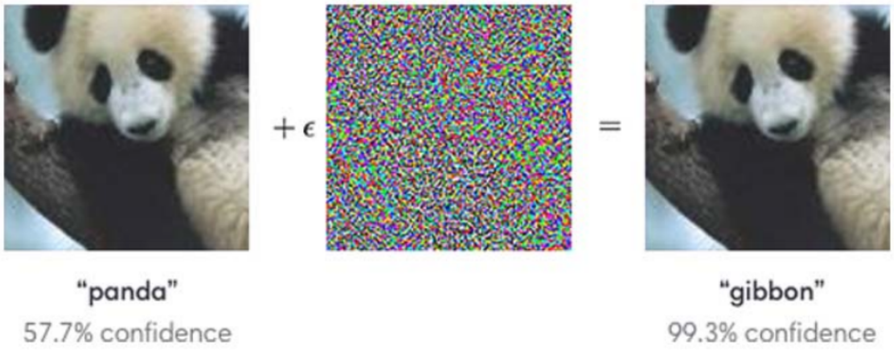
1 ABSTRACT

Rapid advancements in neural networks have resulted in many more applications of neural networks. Some of these applications are safety-critical. For instance, neural networks are being trained to classify traffic conditions for autonomous vehicles. It has been recently demonstrated that neural networks have blind spots where small perturbations to the original input image can make the neural network fail. For current and future applications of neural networks, especially safety-critical applications, it is imperative that we develop effective defense methods to protect against adversarial examples. For this work we implemented ensemble methods using bagging with noise and adversarial retraining defenses against adversarial attacks, such as the Fast Gradient Sign, Basic Iterative, and Limited Memory BFGS methods. We tested the aforementioned defense strategies against attacks using our own models for the MNIST and CIFAR-10 datasets and the Inception V3 model for the Tiny ImageNet dataset. We found that the ensemble method using bagging + noise and adversarial retraining both improved accuracy against adversarial attacks, with the adversarial retraining strategy outperforming our ensemble strategy for the MNIST and CIFAR-10 models and datasets. We also found that the adversarial retraining strategy improved performance quite a bit on the non-adversarial test data.

2 INTRODUCTION

Modern methods used in deep learning have enabled it to be a truly transformative technology. Neural networks have been effectively used for handwritten character recognition, image classification, image compression, stock market prediction, along with many more applications. In many cases, neural networks have achieved super-human levels accuracy. Recently, neural networks have been studied as a method to help increase the security of systems (e.g. detecting if internet connections are secure), however, in Goodfellow *et al.*, 2015 deep security flaws in neural networks are explored. Neural networks are vulnerable to being tricked, much like the human brain is vulnerable to optical illusions. These optical illusions to modern neural networks are called adversarial examples. As neural networks are being integrated into more products, it is particularly important that we develop methods to protect against adversarial examples. In this work, we will briefly describe adversarial examples, summarize methods of generating adversarial examples, and discuss our efforts to defend against these adversarial attacks using adversarial training and ensemble methods.

3 PROBLEM DESCRIPTION



3.1 Overview of Adversarial Examples

Neural networks can be fooled with small, imperceptible, perturbations to the pixels of an image. It is important to note that although the perturbations to the original image may look like noise, these perturbations are not usually a product of applying random noise to an image. Rather, the perturbations used to generate adversarial examples are carefully and intentionally crafted. We can make small imperceptible perturbations to each of the pixels in an image and they can generate adversarial examples because they travel far in vector space as measured by the L2 norm. In other words, we can make almost imperceptible alterations to input images but they can result in large changes in the way the neural network perceives them because of the large dot product results from the coefficients that the linear models represent (Goodfellow *et al.*, 2015).

4 RELATED WORK

Within just the past few years our understanding of adversarial examples have increased tremendously. Adversarial examples were thought to be just an intriguing property and now we understand them to be a very challenging phenomenon, we must consider for future applications. In *Intriguing Properties of Neural Networks*, Szegedy *et al.* (2014) creates the foundation for studying adversarial examples in neural networks, referring to the misclassification of perturbed input images as blind spots in the neural networks. This work is significant to this current study because it outlines the problem of adversarial examples. Szegedy *et al.* (2014) also suggests that these adversarial examples can generalize to different models even if they were trained on different subsets of the original dataset. While adversarial examples could be a result of the nonlinear nature of the neural networks, in their closing remarks they explain that their nonlinear hypothesis is only speculation which will need to be further investigated.

In *Explaining and Harnessing Adversarial Examples*, Goodfellow *et al.* (2015) presents evidence that suggests that the consistent misclassification of adversarial examples by modern neural networks was not a resultant phenomenon from the nonlinearity and overfitting occurring in the model. Instead, they suggest that the main cause of this vulnerability in the neural networks is due to their linear nature in high-dimensional spaces. Based on this new linear perspective of neural network architectures, they developed a very quick and effective method to generate adversarial examples. In this work, they demonstrate the large size of the adversarial example space for modern networks by applying FGSM and another known attack, L-BFGS, to models trained on the MNIST dataset. This work is helpful in this current study because it provides information on adversarial attack methods that are relatively easy to generate.

Goodfellow *et al.* (2015) also confirms results from Szegedy *et al.* (2014) that training on both clean and adversarial examples acted as an effective regularizer for the training process for the models. They even obtained results that outperformed the use of dropout. Goodfellow *et al.* (2015) also presents a hypothesis regarding why adversarial examples generalize well across models. They suggest that modern neural network algorithms were made to generalize and thus are able to achieve relatively stable weight classifications. As a result, the current training methods create about the same linear classifier given the same training set.

In *Ensemble Methods as a Defense to Adversarial Perturbations Against Deep Neural Networks*, Strauss *et al.* (2018), proposed an ensemble method to defend against adversarial examples because ensembles showed promise to defend against certain targeted attacks and enhance the strength of existing adversarial defenses. They found that the ensemble methods they developed for the MNIST and CIFAR-10 datasets improved the accuracy of neural networks on test data and raised their accuracy on adversarial perturbations. They use several ensemble methods including ensembles with identical architectures with random weight initialization, different architectures, Bagging (Breiman, 1996), and addition of Gaussian noise. In the end, they suggest ensemble methods are promising and they hypothesize that they would be especially promising if they are combined with other defense mechanisms like adversarial training. This paper is relevant to this work because it largely served as the inspiration for this study and the methods we use.

In *Ensemble Adversarial Training: Attacks and Defenses*, Tramer *et al.* (2018) implemented an ensemble where each classifier trains on adversarial examples, as suggested by Strauss *et al.* (2018). This approach proved to be successful when implemented on ImageNet, in fact, their implementation won the first round of the *NIPS 2017 competition on Defenses against Adversarial Attacks*. In the end, Strauss *et al.* (2018) acknowledge that although their ensemble adversarial training method works well on attacks like FGSM and some iterative attacks, this method could be exploited by using stronger generative techniques proposed by Baluja and Fischer (2018). This is relevant to our work because their ensemble method is a combination of the methods we use in our paper.

5 METHODS

5.1 Generating Adversarial Examples

When generating adversarial examples the goal is to make nearly imperceptible changes to the input image to actually fool the system. The goal is not to simply change the input image so much that is looks like it should be classified as completely new class. Therefore, in this work, we use adversarial attacks that have been previously demonstrated in other works to generate nearly imperceivable adversarial examples. These attacks include the Fast Gradient Sign Method (FGSM), the Basic Iterative Method (BIM), and the Limited Memory BFGS (L-BFGS) method.

5.1.1 Fast Gradient Sign Method

The FGSM generates an adversarial example, x_{FGSM}, by adding small perturbations to the original input image, x, where the size of each perturbation is $\epsilon > 0$. We can take the sign of the gradient of the cost function, $sign[\nabla_x J\left(\theta, x, y\right)]$, to tell us when to add or subtract epsilon at each point. The gradient can just be computed during prorogation and is computationally cheap to find (Goodfellow *et al.*, 2015).

$$x_{FGSM} = x + \epsilon sign[\nabla_x J\left(\theta, x, y\right)].$$

5.1.2 Basic Iterative Method

The BIM generates an adversarial example, x_{BIM}, similarly to the FGSM but in an iterative manner, where each of the pixel values are updated n times by a size α where $\epsilon \geq \alpha > 0$. The $Clip_{x,\epsilon}$ function used to calculate x_i clips the values of the adversarial example to ensure that each of the pixel values stay within an ϵ bound of the original input image, x (Kurakin *et al.*, 2016).

$$\begin{gathered} x_0 = x, \\ x_i = Clip_{x,\epsilon}\left(x_{i-1} + \alpha sign[\nabla_{x_{i-1}} J\left(\theta, x_{i-1}, y\right)]\right), \\ x_{BIM} = x_n. \end{gathered}$$

5.1.3 Limited Memory BFGS Method

The L-BFGS method generates adversarial examples by finding some optimal value for the constant c, such that the following equation is minimized. To identify the optimal value for c, a binary search can be implemented (Szegedy *et al.*, 2014)

$$min_{x'} c \|\eta\| + J_\theta(x', l'),$$
$$s.t.\ x' \in [0, 1].$$

5.2 Defenses to Protect Against Adversarial Attacks

5.2.1 Adversarial Retraining

The adversarial training defense method consists of adversarial example generation for each of the training examples during training and training on each of the adversarial images. More specifically, a model trains on an image, an adversarial example is generated from the input image. The adversarial example is then injected into the training process of the model. This process repeats for each training image throughout the duration of training.

5.2.2 Ensemble

As illustrated in Figure 1, an ensemble defense method, when used to defend against adversarial attacks, is a collection of classifiers, that classify new data points by some average or majority vote of each of their individual outputs. Bagging (Breiman, 1996), is one such ensemble method that is implemented in this paper. The term Bagging is derived from bootstrap aggregation. For bagging, m samples are drawn from the training data set of m input images. After each draw, the image is replaced into the training set. This process results in m bootstrap replicates, which is some subset of the original dataset with some duplicate images probable in each bootstrap aggregate. Finally, in bagging each one of the classifiers of the ensemble are trained with a different bootstrap aggregate. Bagging with noise, what we implement for this work, is nearly identical to bagging, only Gaussian noise is added to each of the images in the training set before generating the bootstrap aggregates. This process is illustrated in Figure 2.

5.3 Datasets

For this work, we used three different datasets of increasing complexity. These three datasets include the standard MNIST, CIFAR-10, and the Tiny ImageNet data sets. The MNIST dataset consists of 70,000 samples of handwritten digits. Each of these samples have an image size of 28x28 pixels, are greyscale, have been size-normalized, and centered. This dataset is partitioned into 10 classes (*i.e.* 0-9) and have a training set of 60,000 images and a test set of 10,000 images.

The CIFAR-10 dataset consists of 60,000 color images of size 32x32 pixels, with

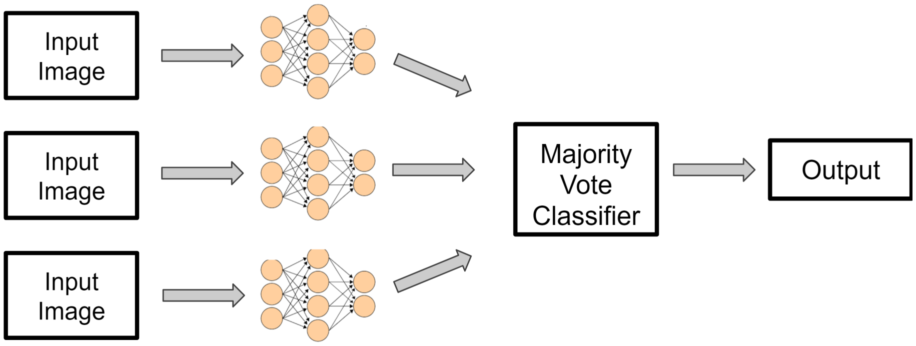


Figure 1: Ensemble Illustration

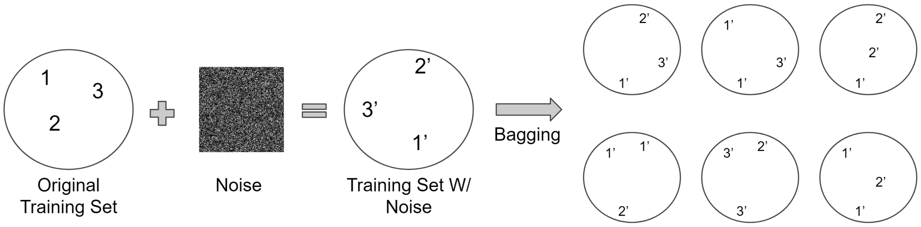


Figure 2: Bagging with Noise Ensemble Illustration

6,000 images for each of the 10 classes. This dataset is partitioned in 10,000 test images and 50,000 training images. The ten classifications for the CIFAR-10 data set are shown in Table 1.

The Tiny ImageNet dataset is a subset of the ImageNet dataset that only contains 200 classifications and 500 training images, 50 validation images, and 50 test images for each class. The Tiny ImageNet dataset has 100,000 colored images of size 64x64 pixels. These images are down-sampled from the original 256x256 pixel images used in the original Tiny ImageNet dataset.

5.4 Architectures & Hyperparameters

For the architecture we used on the MNIST dataset (Table 2), we used a fairly simple convolutional neural network. We Use two two-dimensional convolutional layers, one with 32 and the other 64 filters. Both are using a 5x5 kernel. We then apply a max pooling layer to the last convolutional layer. Next, we flatten the pooling layer and connect it to a fully connected layer with 128 neurons. It is finally connected to a soft-max layer of 10 classes for the final classification. We used a dropout and the default learning rate of 0.001. This model was used because it has been shown that a convolutional layer approach performs better on the MNIST dataset than simply using an architecture that only uses dense

Class Name	**Class Number**
Airplane	0
Automobile	1
Bird	2
Cat	3
Deer	4
Dog	5
Frog	6
Horse	7
Ship	8
Truck	9

Table 1: CIFAR-10 Classifications

Layer Type	**Parameters**
2D Convolutional Layer	32 filters, 5x5 kernel
2D Convolutional Layer	64 filters, 5x5 kernel
Max Pooling	(2,2)
Dropout	0.25
Flatten	
Dense Layer	128 nodes
Dropout	0.5
Dense	10 nodes

Table 2: MNIST Model Architecture

layers.

For the CIFAR-10 Dataset we used a more advanced convolutional neural network. We had two two-dimensional convolutional layers with 32 filters each using a 5x5 kernel size. After these two layers, we applied a layer of max pooling and then dropout, with a dropout rate of 0.25. Next, we used another set of two two-dimensional convolutional layers both with 64 filters and a kernel size of 3x3. Then, we used a layer of max pooling and dropout with a rate of 0.25. We then flattened the max pooling layer and connected it to a dense layer of 512 nodes. After applying dropout with a rate of 0.5, we connect it to softmax layer of 10 nodes for classification. For this architecture, we added more convolutional layers than the architecture used on the MNIST dataset to more effectively learn CIFAR-10 which is more complex than the MNIST dataset.c

For the Tiny ImageNet data we used the Inception V3 Model that is included with the Keras library. We used the Inception model because it is a fairly well known and regarded model for classifying ImageNet data. To do this, we removed the last layer and added a new layer with 2000 nodes because the Tiny ImageNet data set only has 2000 classes.

Layer Type	**Parameters**
2D Convolutional Layer	32 filters, 5x5 kernel
2D Convolutional Layer	32 filters, 5x5 kernel
Max Pooling	(2,2)
Dropout	0.25
2D Convolutional Layer	64 filters, 3x3 kernel
2D Convolutional Layer	64 filters, 3x3 kernel
Max Pooling	(2,2)
Dropout	0.25
Flatten	
Dense Layer	512 nodes
Dropout	0.5
Dense	10 nodes

Table 3: CIFAR-10 Model Architecture

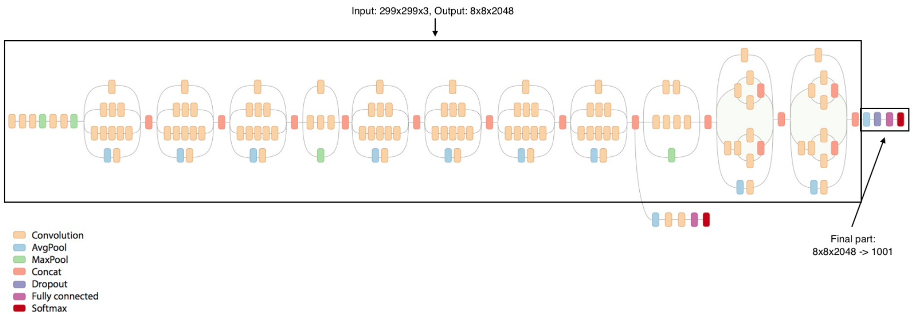


Figure 3: InceptionV3 Model Architecture

For our experiments, we kept each model architecture and hyperparameters constant as we tested the various defense strategies across each adversarial attack method that we tested. We controlled the architectures by creating a model object for each model. We created a new instance for each model in the ensemble and save the models as we train them. Then we loaded the models as we tested them or retrained them.

For each of the models we used the Adam Optimizer and optimized for minimum cross-entropy while training each model. While training each model we applied early stopping by evaluating the accuracy for the classification of each model. In our results, we collected the classification accuracy of the model or ensemble as they are tested on test data, whether it be the case of perturbed or unperturbed data.

6 EXPERIMENTAL RESULTS

Because we are focused on the adversarial robustness of a classifier, our primary metric is the accuracy of various networks used. Below, we present the results obtained from different architectures on various datasets as mentioned in Section 5. Figure 4 represents an image from the MNIST dataset before and after adversarial perturbation using FGSM attack. As the image is just $28x28$ size, the perturbations to the image are visible to the naked eye. The original image in figure 4 is labeled as a 0 whereas, the adversarial image is labeled as a 3 by the model. Figure 5 represents the adversarial images generated by BIM and L-BFGS attacks respectively. We can observe that the left image in figure 5 is similar to the right image in figure 4, this is because the BIM attack is an iterative extension of the FGSM attack where the perturbations become much smaller (less visible to the naked eye). The right image in figure 5 is obtained by applying a targeted attack (L-BFGS) on the original image with the target label as 6. We can observe that the right image in figure 5 somewhat appears as 6 to the human eye because a new faint stroke has been generated in the middle of 0 and some pixels on the top-right of the 0 are blurred. The left image in figure 5 is labeled as a 2 by the network and the right image is labeled as 6 since we specified the target class to be a 6.

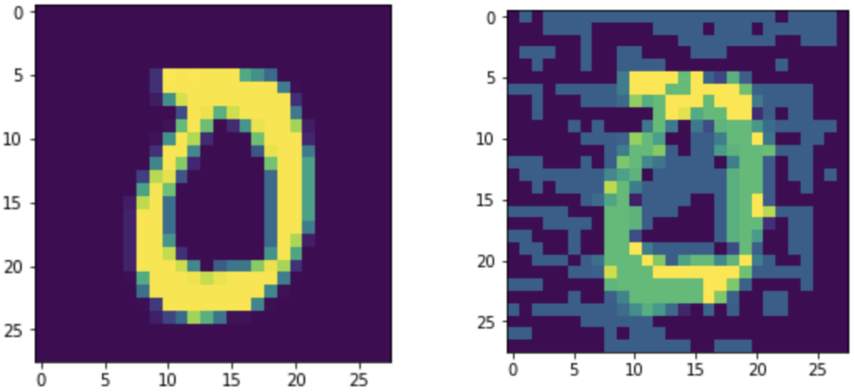


Figure 4: The left image is the original MNIST image which is classified as a 0 by the model. The image on the right is the adversarial image generated using the FGSM attack and is classified as a 3 by the model.

Next, we present an adversarial example obtained using ImageNet on the Inception model. Figure 6 represents an image from the ImageNet dataset. The image on the left is the original image present in the dataset and the right image is an adversarial image obtained by applying FGSM attack on the retrained inception model. We can observe that as the image size increases, the possibility of detecting an adversarial image by a human drastically decreases. The input images provided to the inception model are of size $299x299x3$ pixels.

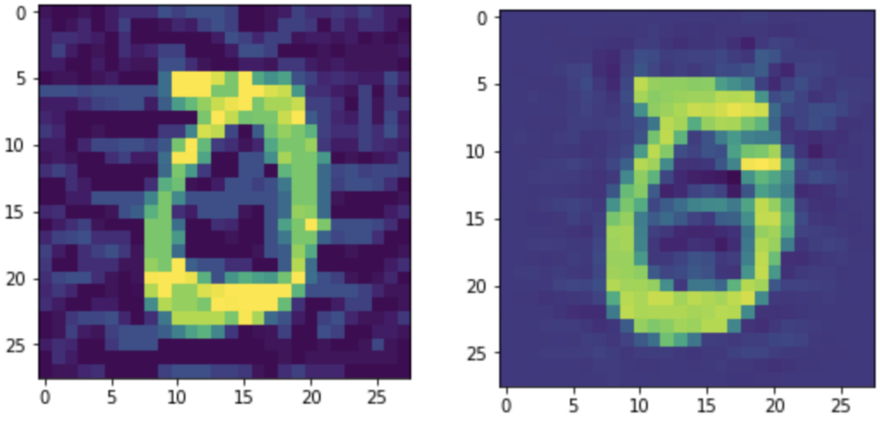


Figure 5: The image on the left is the adversarial MNIST image generated using BIM attack which is classified as a 2 by the model. The image on the right is the adversarial image generated using the L-BFGS attack, which is classified as a 6 by the model.

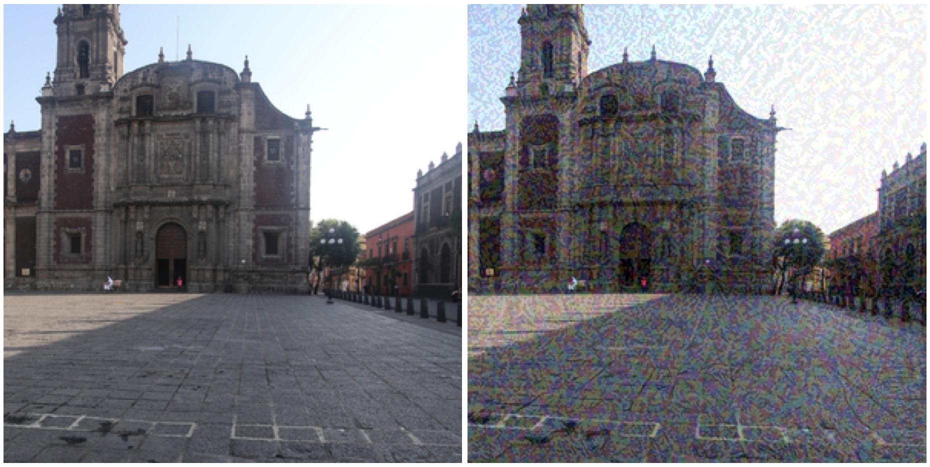


Figure 6: The image on the left is the original image from ImageNet dataset which is classified as a building by the Inception model. The image on the right is the adversarial image generated using FGSM attack on the inception model which is classified as a bookshelf by the model.

Now we present the results obtained by training a set of models as an ensemble using bagging and noisy images. Each model in the ensemble is trained using a noisy subset of the entire training dataset. Figure 7 represents the accuracies of the model obtained on original and adversarial images generated on the original model (trained on the entire dataset) using different attacks. The x-axis represents the number of models in an ensemble and the y-axis represents the accuracy on the test-data or adversarial test data. We can observe that as the number of models in an ensemble increases the accuracies do not vary much. This could be attributed to fewer number of models being considered in our ensemble. We think that the accuracies might increase if the number of models in an ensemble are around one order of magnitude more. Normally, in machine learning techniques an ensemble usually consists of $100 - 500$ models and then an increase in accuracy is observed. However, it is hard to perform those experiments for an ensemble of neural networks because of the computational resource constraints. We also observed that non-targeted attacks have much lower accuracies compared to targeted attacks, this is expected because targeted attacks are much harder to implement and hence cannot work for all images.

We performed the same experiment on adversarial images generated by i^{th} model where $i = 6$ and observe the same trend in results. However, one interesting observation that we found is that training a single model on a subset of noisy data (ensemble with one model) provides better accuracies than the model trained on the entire dataset for adversarial examples. Another observation is that the accuracies of the ensemble on adversarial images generated using the L-BFGS attack on i^{th} model is less compared than those obtained from the original model.

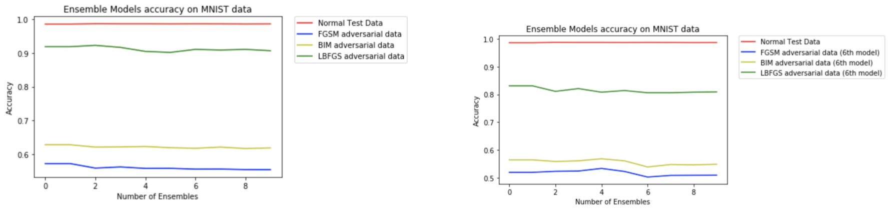


Figure 7: The graph on the left display the accuracies of an ensemble obtained by varying the number of models in an ensemble where the adversarial images are generated on original model. The graph on the right display the accuracies of an ensemble obtained by varying the number of models in an ensemble where adversarial images are generated on i^{th} model.

Figure 8 represents the results obtained on different model settings and on different adversarial data. We can observe that an ensemble performed better than the original model on all different kinds of attacks and reduced the misclassification rate by $20 - 30$ percent. It is also worth mentioning that the i^{th} model in the ensemble performed better than the original model on adversarial examples. Additionally, we can observe that the ensemble performed better when adversarial images are generated on the original model, whereas, it performed slightly worse when the adversarial images are generated on i^{th} model in the ensemble. From this, we can expect that all models in an ensemble are similar to each other so, when an adversarial example is generated only on a single model in an ensemble then there is a high chance that it fools the entire ensemble.

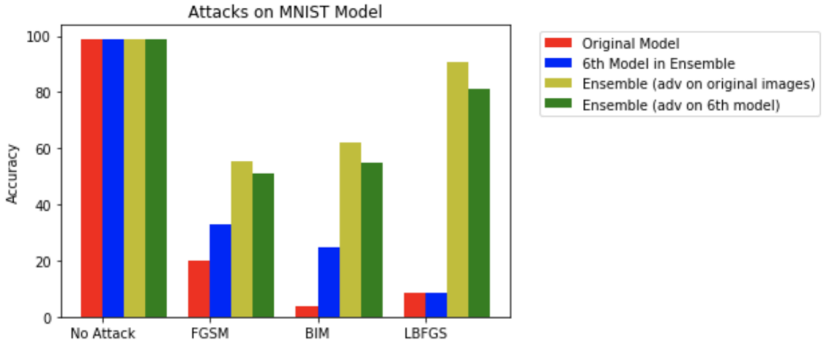


Figure 8: Comparison of accuracies of attacks on different model settings for MNIST.

Adversarial Retraining	**FGSM**	**BIM**
Normal Data	98.36	98.86
Adversarial Data	96.68	93.16

Table 4: Accuracies obtained by adversarial training on MNIST.

Next, we performed adversarial training for the original model, trained on the entire dataset for two different attacks. Our implementation of adversarial training is a process where adversarial images are generated after each epoch and their loss values are also considered in the process of training. So, updates to weights of the network are done based on the normal loss value and the adversarial loss value of training data. This training makes the network more robust against adversarial examples. We observed high accuracies even on adversarial test examples for different attacks, which outperforms the results obtained from the ensemble technique.

Here, we present the results obtained on the CIFAR-10 dataset. For the results of the CIFAR-10 dataset for the ensemble method, it helped improve the accuracy somewhat, but not significantly versus just testing on one of the models in the ensemble. This is shown in Figure 10.

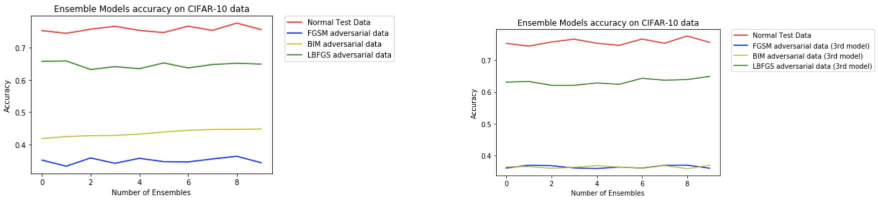


Figure 9: The graph on the left show the accuracies of an ensemble obtained by varying the number of models in an ensemble where the adversarial images are generated on original model. The graph on the right show the accuracies of an ensemble obtained by varying the number of models in an ensemble where adversarial images are generated on the i^{th} model.

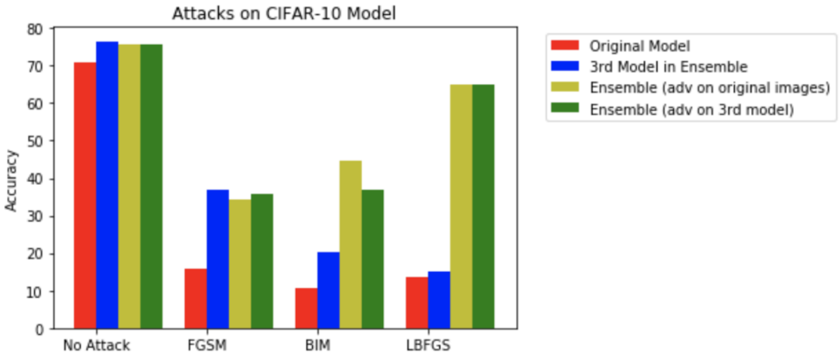


Figure 10: Comparison of accuracies of attacks on different model settings for CIFAR-10.

Adversarial Retraining	**FGSM**	**BIM**
Normal Data	74.56	73.65
Adversarial Data	66.78	58.97

Table 5: Accuracies obtained by adversarial training on CIFAR-10.

We also show in Figure 10 the comparison of accuracies of attacks as well as no attacks on the original model, the third model in the ensemble, an ensemble using 10 models where the adversarial examples are made using the original model, and an ensemble with 10 models using the third model to generate adversarial examples.

In Table 5 we show the results of adversarial retraining for our CIFAR-10 model. We were only able to use this method on the FGSM and BIM attacks. These results are quite promising like the MNIST adversarial retraining from Table 4.

Figure 11 shows a comparision of no attacks, the FGSM, and BIM attacks for

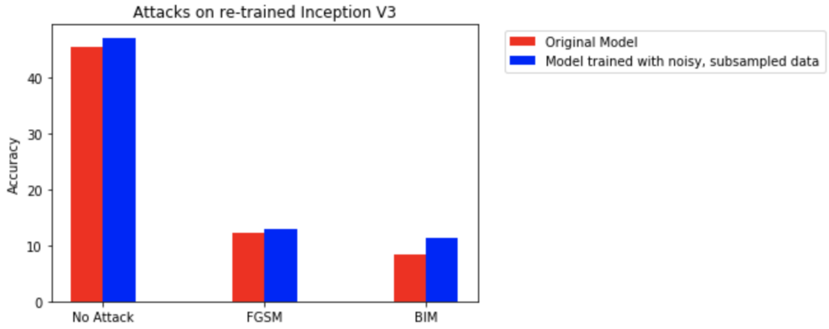


Figure 11: Comparison of Accuracy on Attacks of Tiny ImageNet and the Inception Model.

the original model, as well as, a model trained by data that had noise and subsampling applied to it.

Unfortunately, we due to time constraints and issues with running the Inception model on Crane, we lacked the time to obtain results from bagging with noise for the ensemble method and adversarial retraining on the Inception model trained on the Tiny ImageNet dataset. This will surely be included in our future work.

7 EVALUATION

In our work across the MNIST and CIFAR-10 datasets, we noted similar trends. For example, we noticed a slight increase in accuracy using an ensemble method using bagging with noise. We only used ensembles up to size 10. Other machine learning studies use many more models in their ensembles. If we were to use more models in the ensemble we expect we would see a more significant increase in accuracy against adversarial examples.

We noticed that ensemble methods performed much better against the L-BFGS attack than the FGSM and BIM attacks. The ensemble method appeared to lead to the least significant increase in performance against the FGSM attack in both the MNIST and CIFAR-10 experiments. We also noticed that the ensemble method performed better on the non-adversarial CIFAR-10 data than just one model.

For the MNIST experiments the adversarial retraining led to much better performance against adversarial attacks. Retraining outperformed the ensemble methods. Although we were only able to retrain for the FBSM and BIM attacks, so from this study, it is inconclusive whether it would help improve accuracy for the LBFGS attack.

We also observed that adversarial retraining even improved performance on non-adversarial examples. The adversarial retraining likely helped combat overfitting by acting as a regularizer and thus allowing the models to generalize better.

In this study, the FGSM had a greater effect against the Inception Model than the MNIST or CIFAR-10 models we used. We added noise and subsampling to the Tiny ImageNet Data and noticed a minimal increase in performance.

8 CONCLUSION AND FUTURE WORK

Overall, we noticed that using a bagging with noise ensemble improved the accuracy amongst all attacks, especially LBFGS, but we have yet to see whether it significantly improves the accuracy for a more complex datasets and models like ImageNet and the Inception model.

We also noticed that adversarial retraining across the MNIST and CIFAR-10 dataset improved accuracy on both the non-adversarial dating, meaning that adversarial retraining improves the models regular performance. Adversarial retraining was better at defending against attacks than the bagging with noise ensemble method.

From this study, we suggest it would be best to train your models against adversarial examples generated from an original model when trying defend against particular attacks, especially since it would involve less training than an ensemble method and an increase in general performance. Using an ensemble method may be a good choice when defending against unknown attacks, as you cannot perform adversarial retraining against attacks new and/or unknown attacks. Additionally, if added computation power is not a primary concern ensemble methods may be a good supplement to stronger adversarial defenses.

In general, we noticed that defense strategies including ensemble methods using bagging with noise and adversarial retraining alone increase accuracy on test data and increase classifier robustness against attacks with adversarial retraining defending better against particular attacks.

For future work, we would like to finish our experiments for ensemble and adversarial retraining for the Tiny ImageNet dataset and Inception model. After doing so, we would like to try ensemble (bagging with noise) with adversarial retraining to see if there is an even greater defense observed against attacks including new and/or unknown attacks. We would also like to test adversarial retrained models against the ones they were retrained on. Finally, we would also like to test against more types of adversarial attacks.

9 ACKNOWLEDGMENTS

10 REFERENCES

[1] Breiman, Leo. Bagging predictors. Machine learning, 24(2):123140, 1996.

[2] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of

neural networks. International Conference on Learning Representations, 2014.

[3] Florian Tramer, Alexey Kurakin, Nicolas Papernot, Ian Goodfellow, Dan Boneh, and Patrick McDaniel. Ensemble adversarial training: Attacks and defenses. arXiv preprint arXiv:1705.07204, 2017.

[4] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. International Conference on Learning Representations, 2015.

[5] Kurakin, Alexey, Goodfellow, Ian, and Bengio, Samy. Adversarial examples in the physical world. arXiv preprint arXiv:1607.02533, 2016.

[6] Shumeet Baluja and Ian Fischer. Adversarial transformation networks: Learning to generate adversarial examples. In AAAI Conference on Artificial Intelligence, 2018.

[7] Thilo Strauss, Markus Hanselmann, Andrej Junginger, and Holger Ulmer.

Ensemble Methods as a Defense to Adversarial Perturbations Against Deep Neural Networks. arXiv:1709.03423, 2017.