Using generative methods to create adversarial images to improve classification task.

Bharat Sesham
State University of New York
at Buffalo
bharatse@buffalo.edu

Gaurav Vijay Vergiya
State University of New York
at Buffalo
gvijayve@buffalo.edu

Abstract

In this paper, we implemented a pipeline of deep learning networks to generate adversarial examples using a conditional generative adversarial network and then we further proceeded to use these generated examples to train the classifier and validated if there is any improvement in performance. In our implementation we use a weak classifier trained on an original dataset for a few epochs and use this classifier to fabricate a new dataset of adversarial examples (by attacking the original dataset with an adversarial attack). Then we use this adversarial dataset to train a conditional generative adversarial network that produces similar adversarially generated adversarial examples that were able to trick even a strong classifier trained on the same original dataset. Then we also observed that these generated adversarial examples can be used to further train the classifier to increase its accuracy by inferring the generated examples as a real-world data with noise.

1. Introduction

In this paper, we are going to discuss our architecture along with the various models used to achieve the desired results and discuss prospective ideas on how we can use these results to further improve the classification accuracy of any classifier. The structure of the paper is as below, In Section 1 the paper starts by introducing some basic concepts and architectures that we used, some of which are generative adversarial network (GAN), white-box adversarial attacks (such as FGSM and C&W) and basic classifiers (CNNs). Then in Section 2, we discuss our architecture in-depth and explain how various components are bridged together. Following that in Section 3 of the paper, we discuss the methodologies and other details that we followed during the implementation, and also discussed some possible improvements or ideas that can be implemented in future publications. Then we proceed with the Results in Section 4 followed by conclusion in Section 5 and project learnings in Section 6. Now we proceed to discuss the primary architectures in brief, before we move on to our implementation.

1.1. Classifier

Image classifiers have become more common in the modern days and are used almost everywhere. The traditional classifier we are using in our implementation is the convolution neural network. We will be having a classifier which will be trained for a small number of epochs (referred as weak classifier) and the same classifier trained for a greater number of epochs (referred as strong classifier).

1.2. Adversarial Attacks

With the increase in popularity and usage of various machine learning and deep learning algorithms to tackle a real-world problem raises a huge concern of security. We already know that these algorithms accept numeric vectors as input and there are well established methods/ways to design an input in a specific way to get the wrong result from a model. These methods are referred to as adversarial attacks. These attacks can be broadly classified into two categories. The first one is the black-box attack in which the attacker doesn't have detail of the model or dataset it was trained on but uses a custom dataset and the original model outputs on this dataset, as the ground truth to estimate a new model that will be further used to generate adversarial examples. The effectiveness of these examples in fooling the original model are shown in various papers recently. The second kind of attack is the white-box attack in which the attacker has the complete access to the dataset and the model being attacked. In this paper, we focus on implementing two white-box attacks and use one of the attacks for further use. The two attacks are fast gradient sign method attack (FGSM) and Carlini and Wagner's adversarial attack (C&W attack).

1.2.1 Fast Gradient Sign Method attack (FGSM)

In the fast gradient sign method, we use the gradients of the known neural network to create adversarial examples. For a given input image, FGSM method uses the gradient of the loss with respect to the input image to fabricate a new image that maximizes the loss. This created image is called an adversarial image. The following expression can be used

to summarize the attack:

$$adv\_x = x + \epsilon * \text{sign}(\nabla_x J(\theta, x, y))$$

In this expression, *x* is the original image, *y* is the original label/ground truth, ϵ is the epsilon which is used as a multiplier to ensure the perturbations are small, θ is the model parameters and finally *J* is the Loss. Then the final created adversarial image will be *adv_x*. Below is a sample perturbation and the corresponding adversarial image and we created using a weak classifier:

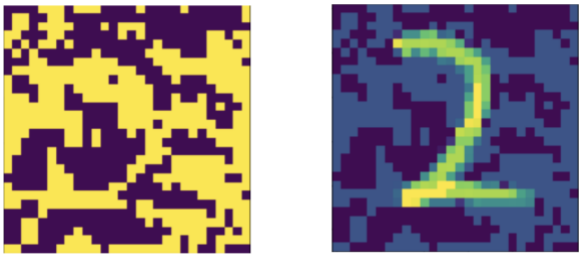

Figure 1: Sample perturbation and the corresponding adversarial image created using a FGSM attack.

An intriguing property of the FGSM method is that the gradients are taken with respect to each input image, as the objective is to create an image that maximizes the loss. To accomplish this, we find out how much each pixel in the image is contributing to the loss value and then add the perturbation accordingly. The computation is fast as we can use the chain rule to find the pixel contribution to the loss and finding the required gradients. So, these created images, although look like the original images, can be used to fool the model. For example, the above image is classified as 1 with 69.2% probability when we evaluated it with the same classifier.

1.2.2 Carlini and Wagner's attack (C&W attack)

In C&W attack, the main objective is to find an adversarial instance of an image by finding the smallest noise and adding it to the image that will change the classification to a different class which is not the ground truth. We can find the noise by:

$$\text{minimize } ||\delta||_p \quad \text{subject to } C(x+\delta) = t, \quad x + \delta \in [0,1]^n$$

Here δ is the noise, *x* is the image and *t* is the misclassified class. So, *C(x)* is the label returned with the image *x*. The noise level is measured in the Lp distance and finding this distance while ensuring that the fabricated image will have the class t is a difficult non-linear optimization problem. So, C&W introduce function to deal with this issue by restricting the $f(x+\delta) \leqslant 0$. So, the above equation is reduced to:

$$\text{minimize } ||\delta||_p \quad \text{subject to } f(x+\delta) \leq 0, \quad x + \delta \in [0,1]^n$$

Also, the recommended choice for function f is given below. The Z(x)tis the output of the neural network before the SoftMax layer.

$$f(x) = \left([\max_{i \neq t} Z(x)_i] - Z(x)_t\right)^+$$

The implementation from Yumi's article has helped us in understanding the C&W attack in detail and below is sample output after the C&W attack on one of the sample images:

Figure 2: Original image and the corresponding adversarial image created using a C&W attack.

As we can see from the above figure that the distortion in the noise in the perturbed image is very less (in fact 770 pixels remained undisturbed), however when we evaluate the image on the right, we got prediction of 2 with confidence of 0.303 and the remaining probability is distributed between the classes 1 and 3. The original image is classified as the 1 with 100% confidence by the same classifier. So, we can see how even a minimal change in the input is causing the classification to be wrong.

We can see that these notorious inputs are indistinguishable to the human eye but cause the neural network to fail to identify the contents of the image. And we have also seen how these adversarial examples are successful in confusing a neural network, resulting in the misclassification of a given input. Some of the innocuous (and presently available) networks like a network trained to drive a car could be at risk to such attacks and could lead to having serious consequences. So, it is very important to design neural networks that are more reliable and are resistant to such adversarial attacks.

1.3. Generative Adversarial Network (GAN)

Generative Adversarial Networks (GANs) is a class of machine learning framework that was introduced in 2014 and are mainly used to generate synthetic data/output starting with noise or some randomized inputs that is sampled from a predefined latent space such as a multivariate normal distribution. A standard GAN will have two networks that are competing against each other. These two networks are the generator network and discriminator networks. The output from the generator model will be

passed to the discriminator as inputs which will flag them either as real or synthetic data. The generator trains on whether the generated output succeeds in fooling the discriminator, while the discriminative network trains to distinguish candidates generated by the generator from the true data. Backpropagation procedures are applied independently to both networks such that the generator produces images that are more realistic, while the discriminator becomes skilled at flagging synthetic images. A better summarization from Wikipedia is "The generator is typically a deconvolutional neural network, and the discriminator is a convolutional neural network". Below are the examples generated by our implementation basic GAN:

Figure 3: Images generated using GAN.

1.3.1 Conditional Generative Adversarial Network

We have also used a variation of the generative networks called the conditional generative adversarial networks (cGAN). Although GANs are capable of generating new images from a given dataset, there is no way to control the type of images that will be generated and are mostly random at times. The only way to distinguish them is by human estimation; i.e., by observing at the generated images. To resolve this dependency, we can make use of cGAN that can involve conditional generation of images by the generator model. The condition on the class label can be used to generate corresponding class images, which gives us the flexibility to generate targeted images of a given type. The samples image generated by our implementation of cGAN for each class in the MNIST dataset can be viewed in the Figure 4.

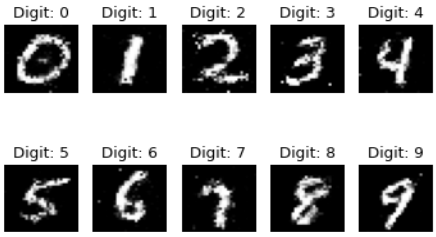

Figure 4: Images generated using cGAN for each class from MNIST dataset.

As we have discussed most of the components that we will be using, we can proceed to the next section to discuss our architecture and design.

2. Architecture

In this section, we will be discussing our architecture in detail. Also, as we have discussed most of the components in the previous section, we will directly dive deep into our architecture. In our architecture there are three main bridges/sections that can be individually swapped with different functionality. Firstly, there is an adversarial dataset fabricator section which includes a weak classifier and an adversarial attack (we used the FGSM method discussed above). In the second section, there is a conditional generative adversarial network that takes the input through bridge 1/ section 1 and in the last section we have used a strong classifier (trained on a greater number of epochs) which takes the input from section 2 (output of cGAN), this is mainly bridged to evaluate our generated adversarial examples.

2.1. Weak Classifier and FGSM attack

In this section, we construct a simple classifier that we train on a limited number of epochs (weak classifier), and then use it to generate FGSM attack examples for the entire original dataset. Also, the idea to use a weak classifier instead of any strongly trained classifier was inspired from the discussed black-box attack where the attacker tries to estimate the model parameters with the knowledge of the model. By this, we were able to show that the adversarial examples fabricated by the gradients and loss of the weak classifiers can be used to train the cGAN which is described in detail in the next section. However, we were able to observe that these generated images from conditional GAN were able to fool even a strong classifier despite learning from an adversarial attack on a weak classifier. This could be a potential advantage while generating new adversarial examples to train on, as we only need a weak classifier to fabricate the adversarial examples.

2.2. Conditional Generative Adversarial Network (cGAN)

As specified in section 2.1, the input to the conditional GAN will be the fabricated adversarial images from the weak classifier. The main objective of this section is to generate images which are close to the provided adversarial examples. Initially, we constructed a traditional GAN to check if this idea is feasible and the GAN is able to replicate the perturbation from the input adversarial examples. And after some iteration, we were able to get a decent output that looks very similar to the adversarial example presented above. Below is one of the examples generated by the GAN when we trained it on the perturbed data:

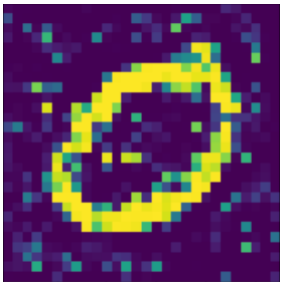

Figure 5: GAN generated adversarial example.

Not only this, when we tried to classify it with the weak classifier, it has classified this image as "8" with a 91.46% confidence. Now, as we got proof of concept working that GAN will be able to generate adversarial examples, we wanted to implement a conditional GAN which allows us to have more control over the class of the adversarial image generated. So, we used the same perturbed dataset to train the cGAN and it was also able to generate images close to the actual adversarial examples. Below is the sample of perturbed images generated using conditional GAN:

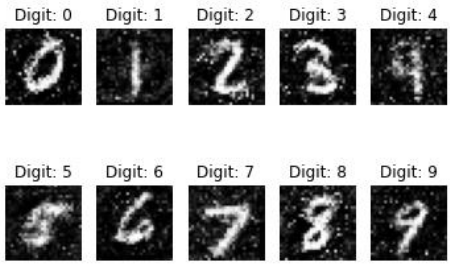

Figure 6: Adversarial examples generated using cGAN.

2.3. Strong Classifier

This is the final step of our architecture in which we evaluate the cGAN generative adversarial images on a strong classifier. The output from the generator network which is passed to the discriminator that flags it as either real or synthetic data, is also passed to the classifier to predict the class of the generated images. As we are generating adversarial examples, it is the expectation that the classifier will incorrectly predict the class with a high accuracy as from the FGSM experiment. One more thing which we plan to test in the future, is that we can combine two different types of attacks while training the GAN with different weights, so the classifier will have even more difficulty in classification.

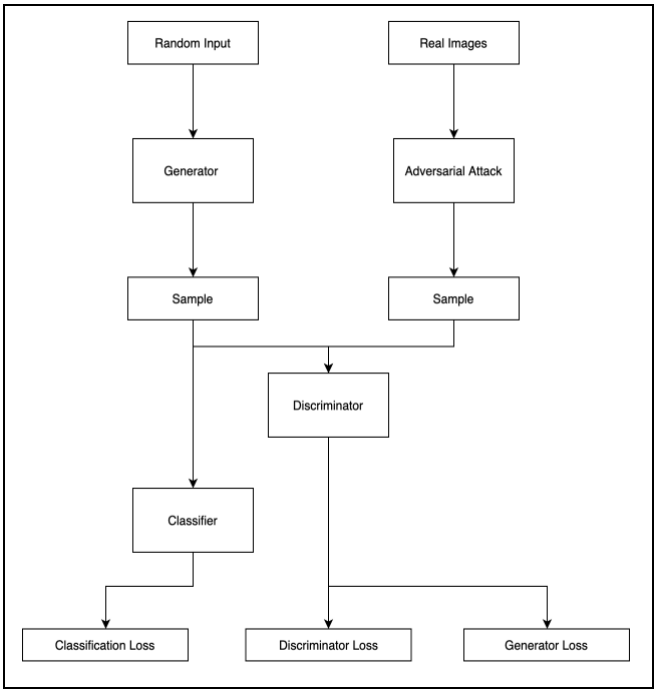

Figure 7: Architecture to generate adversarial examples using GAN and using them to train the classifier.

3. Methodology

3.1. Implementation

As discussed in the earlier sections the goal of the experiment is to train a condition GAN to automatically generate adversarial examples. We in our implementation use MNIST dataset so study this behavior. Firstly, we perform an FGSM adversarial attack on the entire MNIST dataset using a weak classifier and we then use the images generated from this attack to train a condition GAN. We also train another condition GAN on the original MNIST images to study how well the classifier performs on the new fake images generated by the cGAN. Later we study how well the classifier performs on the new fake images generated by the cGAN which was trained using the FGSM attack examples. We then use the images generated by the later cGAN (which were successful in fooling even a strong classifier by generating images similar to FGSM) to train the classifier which would result in a more reliable classification and will be less prone to such attacks.

3.2. Architecture and Implementation details.

3.2.1 Dataset

MNIST dataset was used to perform the experiments as it was a computationally inexpensive dataset.

3.2.2 MNIST CNN Classifier

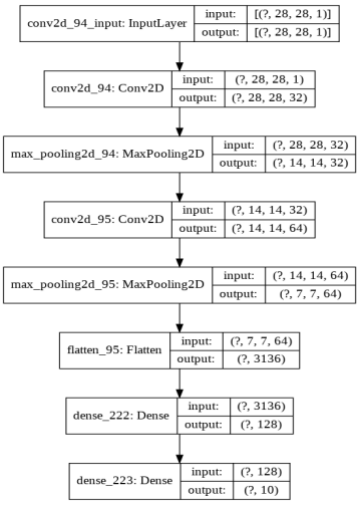

Figure 8: Architecture of the CNN classifier used in the implementation.

MNIST Classifier Model Parameters:

- Loss function: categorical_crossentropy
- Optimizer : Adam
- Learning rate : 0.001
- Beta1 : 0.9
- Beta2 : 0.999.
- Epsilon : 1e-07.

3.2.3 cGAN on MNIST

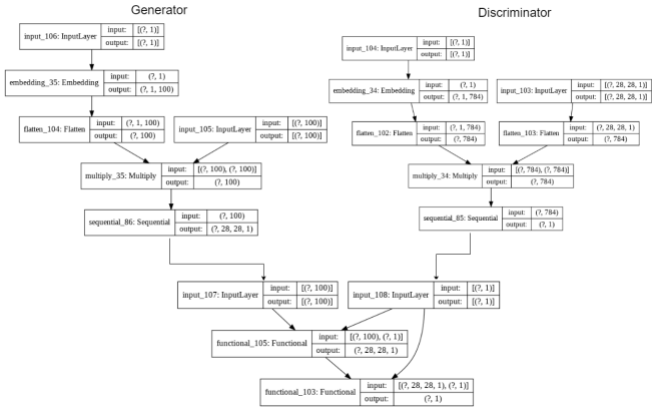

Figure 9: Architecture of the cGAN used in the implementation.

Generator and Discriminator Model Parameters:

- Loss function : Binary_crossentropy
- Optimizer : Adam
- Learning rate : 0.0002
- Beta1 : 0.5
- Beta2 : 0.999
- Epsilon : 1e-07 }

4. Results

4.1.1 cGAN generated images.

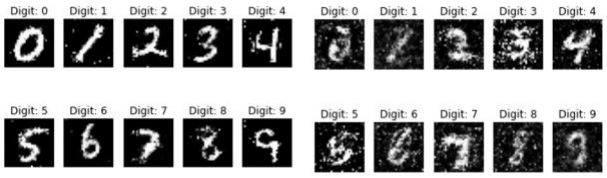

Figure 10. Output images generated by cGAN on the original MNIST dataset and FGSM attacked MNIST dataset respectively.

4.1.2 Evaluation of classifier on images produced by Direct FGSM attack on Original MNIST dataset.

Refer the Figure 11 in the next page.

4.1.3 Evaluation of classifier on images produced by cGAN trained on original MNIST dataset.

Refer the Figure 12 in the next page.

4.1.4 Evaluation of classifier on images produced by cGAN trained on FSGM attacked MNIST dataset

Refer the Figure 13 in the next page.

4.2 Compare performance of various attacks.

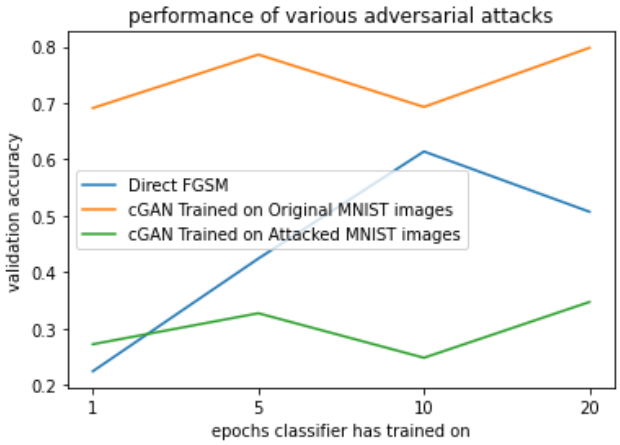

Figure 14. Performance of various adversarial attacks on the strong classifier

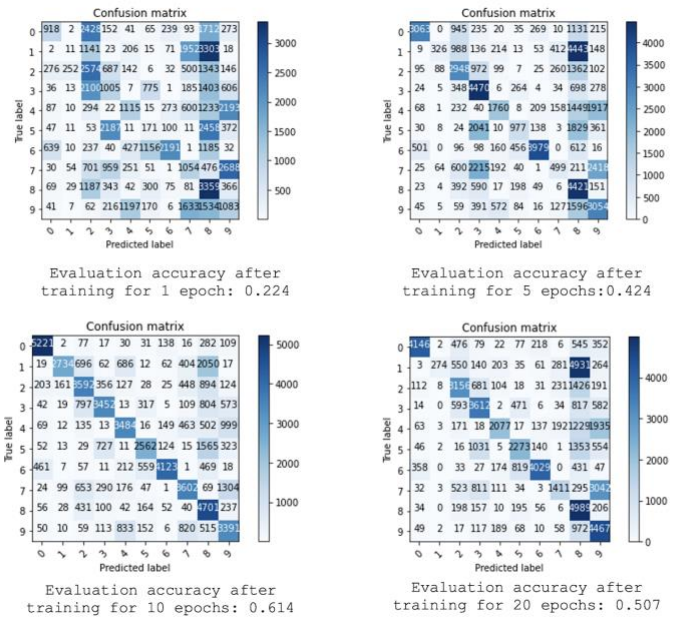

Figure 11. Evaluation of classifier trained for different number of epochs on images produced by direct FGSM attack on original MNIST dataset.

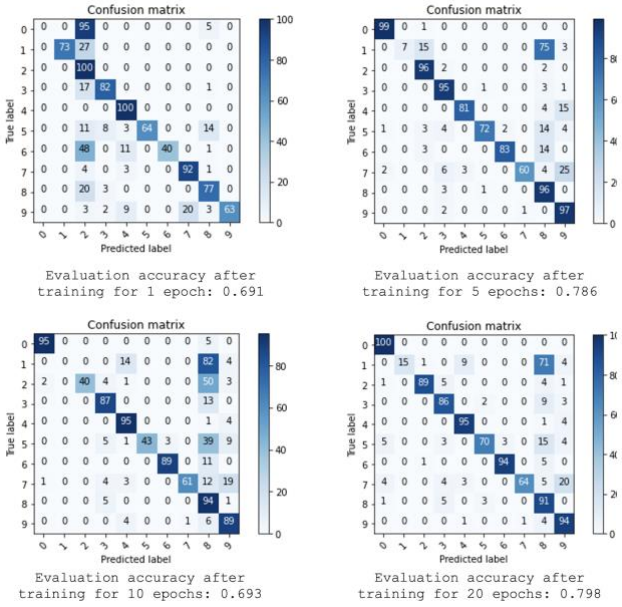

Figure 12. Evaluation of classifier trained for different number of epochs on images produced by cGAN on original MNIST dataset.

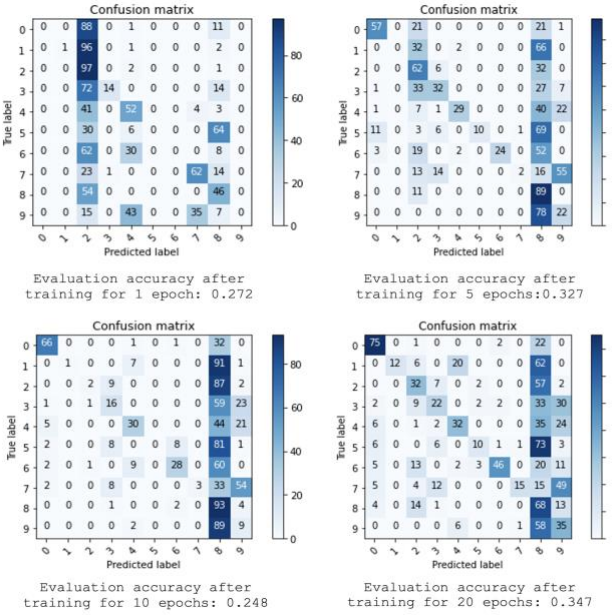

Figure 13. Evaluation of classifier trained for different number of epochs on images produced by cGAN on FGSM attack dataset.

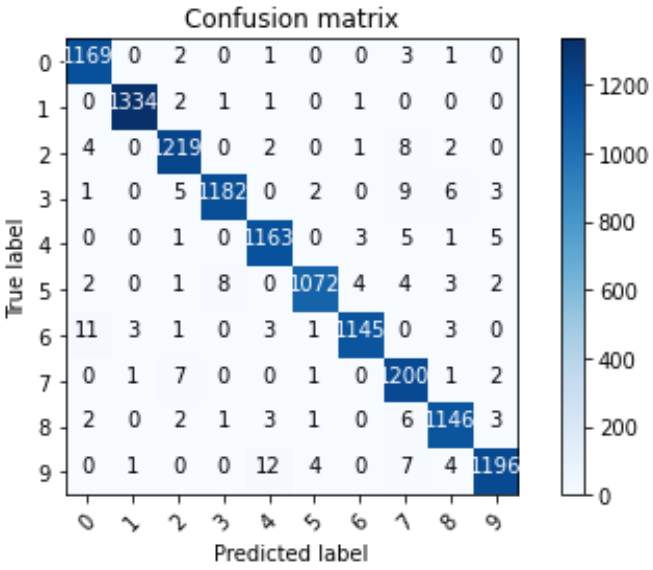

Figure 15. Evaluation of classifier trained for different number of epochs on images produced by cGAN on original MNIST dataset.

4.3 Improved classifier trained on MNIST and CGAN generated adversarial samples

Test loss and test accuracy of the strong classifier after training with the generated images.

```
Test loss      : 0.0659
Test accuracy  : 0.985
```

Refer the Figure 15 for the confusion matrix of the retrained classifier.

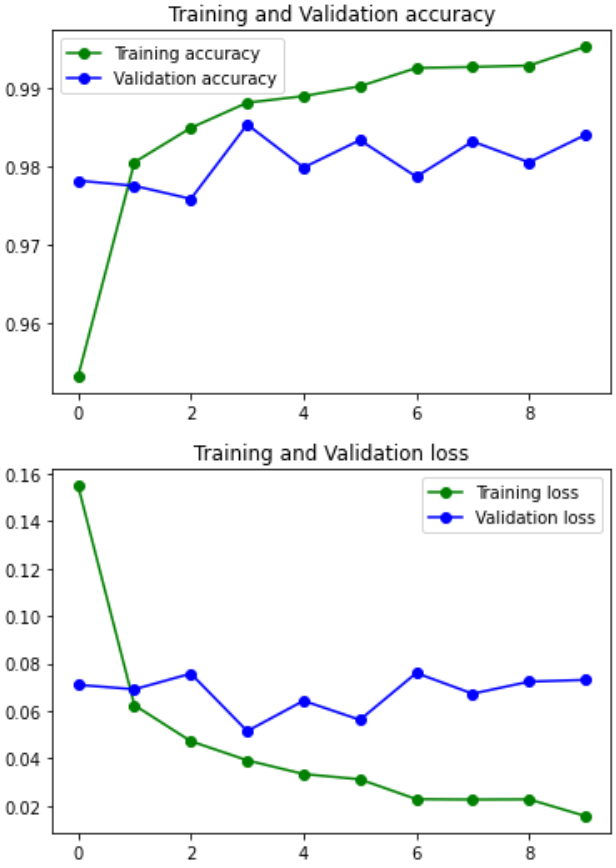

Figure 16. Training and validation accuracy and loss plots of the retrained classifier.

5. Conclusion

It's a known fact that machine learning systems are increasingly being adopted in real life tasks and hence questions about their security are rising, In this experiment we have explored the vulnerabilities of digit recognition systems which are widely adapted by many organizations like postal services to identify handwritten digits, we see how we can fool the classifiers used in these systems using adversarial attacks like FGSM. We found that FGSM attack on a strong classifier didn't produce images which looked like the actual digits. We hence used a weak classifier to do the FGSM attack and we later used the images produced by this attack to train a cGAN which successfully learnt to generate adversarial images which would fool even a strong classifier. This creates the ability to generate adversarial examples using noise and later these adversarial attacked images can be used to improve the classifier and make it more robust and immune to such attacks.

6. Project Learnings

There is a lot to take away from this project and we have learnt a great deal from each aspect of this project, firstly we were able to explore and gain hands-on experience on the adversarial attacks mainly the white-box attacks. Then we were able to inspect generative adversarial networks (GANs) and one of its variations - conditional generative adversarial networks (cGAN) which fit our use case. Also, one of the most important aspects that we learnt from this project is how we can build sizeable and scalable deep learning networks that can have real world impact. We also understood that as future developers of machine learning systems we need to focus on issues like security and reliability of these models and what impact it can have on the end user and society instead of just their performance metrics. Apart from the technical learnings mentioned above, we also got a new perspective on how much of an impact deep learning and machine learning models are having on our daily lives and on society.

7. References

[1] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville and Y. Bengio, "Generative Adversarial Networks," *Proceedings of the International https://arxiv.org/abs/1412.6572v3 [stat.ML]Conference on Neural Information Proces*, 2014.

[2] Wikipedia, "Generative_adversarial_network," [Online]. Available: https://en.wikipedia.org/wiki /Generative_adversarial_network.

[3] tensorflow, "tensorflow tutorials adversarial fgsm," [Online]. Available: https://www.tensorflow.org /tutorials/generative/adversarial_fgsm.

[4] machinelearningmastery, "how-to-develop-a-conditional-generative-adversarial-network-from-scratch," [Online]. Available: https://machine learningmastery.com/how-to-develop-a-conditional-generative-adversarial-network-from-scratch/.

[5] fairyonice.github, "Learn-the-Carlini-and-Wagners-adversarial-attack-MNIST," [Online]. Available: https://fairyonice.github.io/Learn-the-Carlini-and-

Wagners-adversarial-attack-MNIST.html-the-Carlini-and-Wagners-adversarial-attack-MNIST.html .

[6] Medium, "tricking-neural-networks-create-your-own-adversarial-examples," [Online]. Available: https://medium.com/@ml.at.berkeley/tricking-neural-networks-create-your-own-adversarial-examples-a61eb7620fd8

[7] souravsingh, "MNIST-Adversarial-Attack (github)," [Online]. Available: https://colab.research.google.com/github/souravsingh/mnist-adversarial-attack/blob/master/Keras_MNIST_attack.ipynb