GNR_652 : Course Project

Topic : Adversarial Training (Attack and Defence) on Mnist dataset

Team Members :

1) Jay Tukaram Sawant (18D070050)
2) Lokesh Sudam Pawar (18D170017)
3) Prashil Pramod Patil (18D070059)
4) Yash Vinesh Gadhia (180100130)

Course Instructor : Prof. Biplab Banerjee

INDEX :

Problem Statement

Most existing machine learning classifiers are highly vulnerable to adversarial examples.
An adversarial example is a sample of input data which has been modified very slightly in a way that is intended to cause a machine learning classifier to misclassify it. In many cases, these modifications can be so subtle that a human observer does not even notice the modification at all, yet the classifier still makes a mistake.

Aim :

To build an accurate deep learning model and investigate methods to attack & defend it using adversarial examples. More broadly, the goal is to explore about the current techniques towards an adversarially robust machine learning system.

Dataset used:

MNIST handwritten digit classification dataset

Motivation for the Project

- Though today(in 2020) we have state of the art Machine Learning Systems which even surpass human level performance on certain tasks, these models can be easily fooled (even if the adversary has no access to the underlying model) by carefully tweaking the input image & thus adversarial examples pose a serious security threat to AI systems.
- For example, Tesla has come a long way in building self driving cars. However, recently it was seen that the models used by Tesla can be fooled by simple stickers(adversarial patches) on the road, which the car interprets as lane diverging, causing it to drive into oncoming traffic.
- Apart from adversarial training being an important field, another motivation we have is that this is a currently active research area in the machine learning community & after getting a kick-start through this project we would wish to further explore our interest in Adversarial Machine Learning & AI Security.

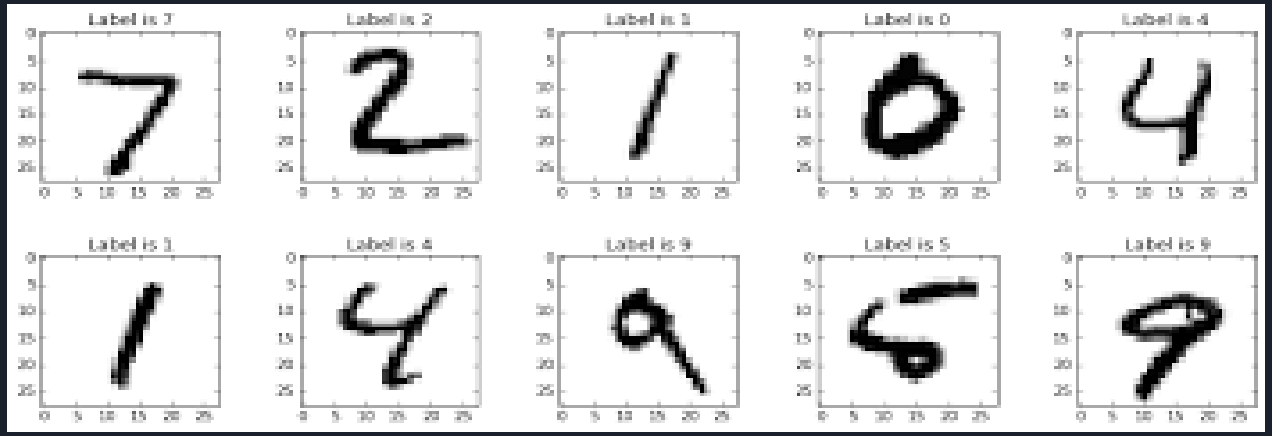
Workflow of the Project

1. Building a Deep CNN Model for MNIST Handwritten Digit Classification.
2. Evaluation of the accuracy of the above model.
3. Generation of Adversarial Examples using FGSM (Fast Gradient Sign Method)
4. Evaluation of the accuracy of the model on Adversarial examples
5. Retraining the model with adversarial examples
6. Comparing the robustness of the new model with the original

Note : We wanted to explore two different approaches for Adversarial Defense & retraining. One approach is to defend against black-box attack while other is to defend against white-box attack. Thus we have two different versions & we implemented them in different frameworks, one in Tensorflow & the other in PyTorch so as to learn both of them.

Mnist Dataset

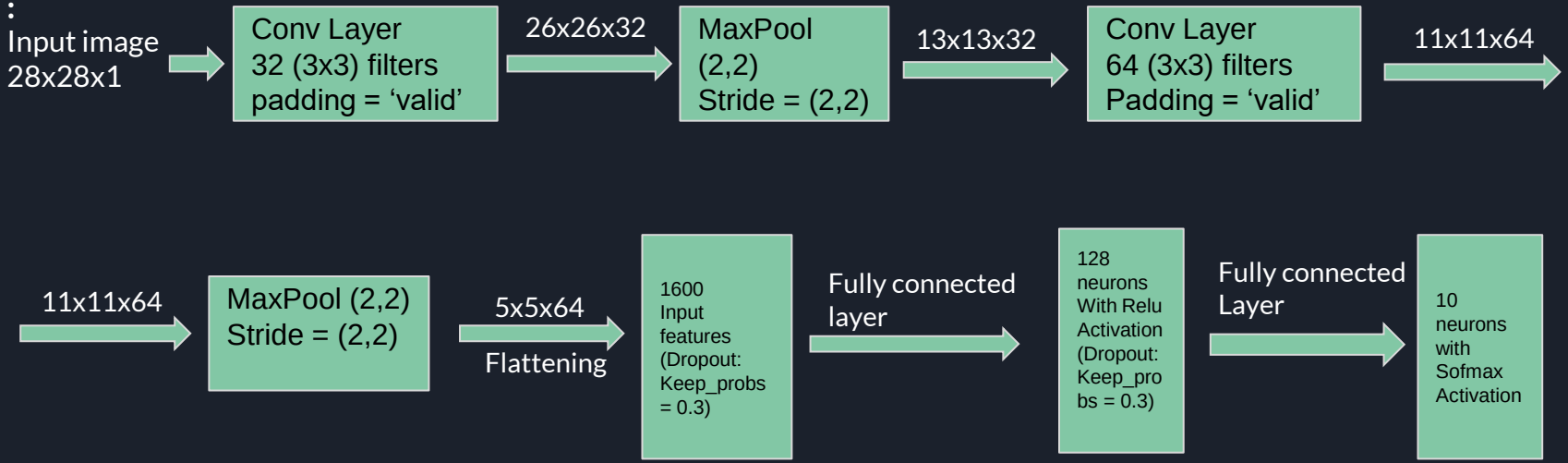
- Below are some of the examples from the Mnist dataset.
- Each image is a 28x28 pixels, grayscale image
- There are a total of 60000 images in the training set
- There are a total of 10000 images in the test set
- There are 10 output classes each representing a digit from 0 to 9.



Mnist Digit Classifier

- We use the below Convolutional Neural Network Architecture for our Model.
- Our Network Architecture is inspired from LeNet-5.
- Each activation function in the convolutional layer is ReLU
- We use Dropout Regularization technique in the Fully connected layers.

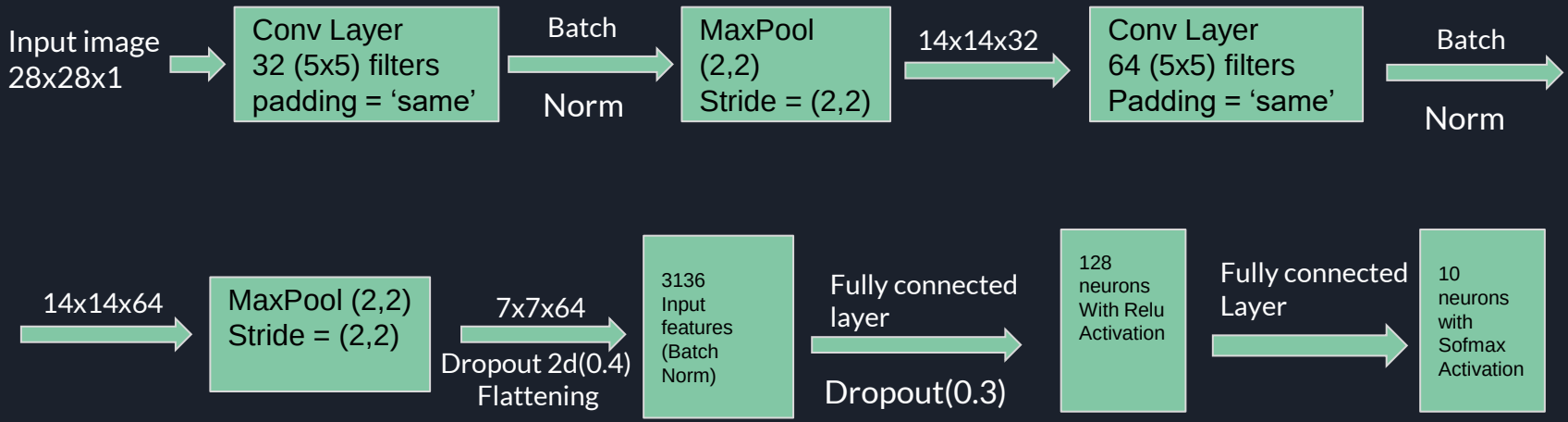
CNN Architecture used in Tensorflow



Mnist Digit Classifier

For this model, Data augmentation was used where each image in the training set is rotated by four different angles which were randomly chosen between -20° to 20°. Batch Normalization is also utilized.

CNN Architecture used in PyTorch :



Mnist Digit Classifier

- We have developed two different classifiers, one in Tensorflow+Keras (Model-1) and the other in Pytorch (Model-2) for classifying the digits .
- Adam optimizer is used for the minimization of the Loss Function.
- Training Accuracy :
- Model-1: 98.76 %
- Model-2 : 99.95%
- Test Accuracy :
- Model-1: 99.27 %
- Model-2: 99.33%

```
Epoch 5/5
1875/1875 [==============================]
313/313 [==============================] -
Test Accuracy= 99.27999973297119 %
```

```
Epoch No. = 50 cost = 3.76944899559021
Training Accuracy =  0.9995372486288848
Test Accuracy =  0.9933238636363636
```

Generating Adversarial Example

Fast Gradient Sign Method

The Fast gradient sign method works by using the gradients of the neural network to create an adversarial example. For an input image, the method uses the gradient of the loss with respect to the image to create a new image that maximizes the loss.

$$\text{Adversarial\_image} = x + \varepsilon * \text{sign}(\nabla_x J(\theta, x, y))$$

where,

x = Original Input Image

θ = Model Parameters

y = Original input label

ε = Multiplier to ensure perturbations are small

J = Loss function

Underlying Equations in FGSM

- First Order Taylor Approximation of Cost function for Adversarial Image

$$J(\tilde{\boldsymbol{x}}, \boldsymbol{\theta}) \approx J(\boldsymbol{x}, \boldsymbol{\theta}) + (\tilde{\boldsymbol{x}} - \boldsymbol{x})^\top \nabla_{\boldsymbol{x}} J(\boldsymbol{x}).$$

- Maximize

$$J(\boldsymbol{x}, \boldsymbol{\theta}) + (\tilde{\boldsymbol{x}} - \boldsymbol{x})^\top \nabla_{\boldsymbol{x}} J(\boldsymbol{x})$$

- Constraint & Solution

$$||\tilde{\boldsymbol{x}} - \boldsymbol{x}||_\infty \leq \epsilon$$

$$\Rightarrow \tilde{\boldsymbol{x}} = \boldsymbol{x} + \epsilon \text{sign}\left(\nabla_{\boldsymbol{x}} J(\boldsymbol{x})\right).$$

- Let x be the original image, y the class of x, θ be the weights of the network

  and $J(\theta,x,y)$ the loss function used to train the neural network.

- $\nabla_x J(\theta,x,y)$ is the gradient of the loss function with respect to the input image.

- We are only interested in the sign of the gradient to know if we want to increase or decrease the pixel value. We multiply this sign by a very small number ε to ensure that we do not go too far on the loss function surface and that the perturbation will be imperceptible.

- $\text{Perturbation} = \varepsilon * \text{sign}(\nabla_x J(\theta,x,y))$

  $X_{adv} = x + \text{Perturbation}$

Adversarial Example

Predicted: 4

Adversarial Image

=

Predicted: 9

Original Input Image

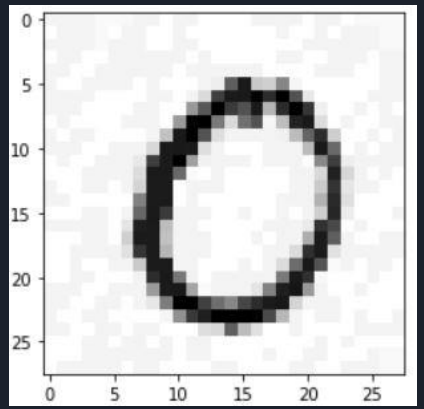
$+ (\varepsilon=0.1) *$

Perturbation

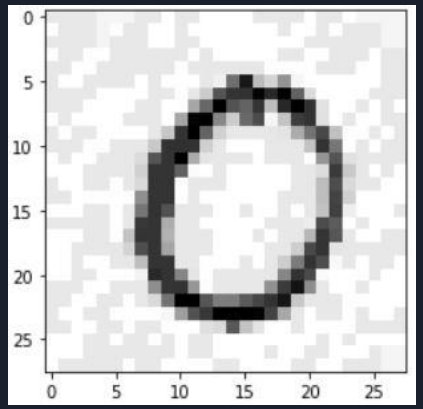
Types of Adversarial Attacks

- Classification 1:
  - **White-Box Attack** : In case of a white box attack, the attacker has complete access to the parameters & gradients of the model.
  - **Black-Box Attack** : The type of attack where the attacker has no information about the model, or has no access to the gradients/parameters of the model.
- Classification 2:
  - **Targeted Attack** : A targeted attack is one where the attacker perturbs the input image in a way such that the model predicts a specific target class.
  - **Untargeted Attack** : An untargeted attack is one where the attacker perturbs the input image such as to make the model predict any class other than the true class.

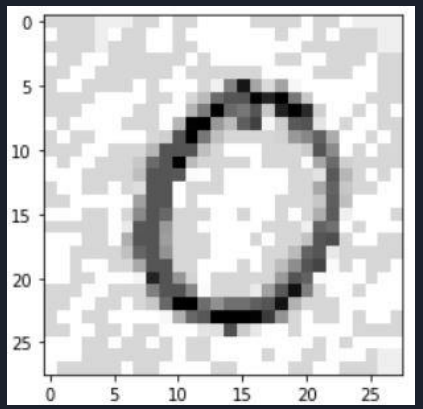
Adversarial Example for different values of Epsilon



Epsilon = 0.05



Epsilon = 0.1



Epsilon = 0.15

Evaluation of the Original Model on Adversarial Examples

- Adversarial examples were created for each example in the test set.
- For evaluation of original model, White-Box attack is carried out in both cases by accessing model parameters & taking gradient.
- The Model was tested for different values of epsilons (0.05, 0.10 and 0.15)
- Model's accuracy on the Adversarial examples of the test set is as follows:
  - ε = 0.05 (Model 1) → Accuracy = 93.72 %
  - ε = 0.10 (Model 1) → Accuracy = 77.88 %
  - ε = 0.15 (Model 1) → Accuracy = 55.98 %

ε = 0.05 (Model 2) → Accuracy = 92.19%

ε = 0.10 (Model 2) → Accuracy = 65.54%

ε = 0.15 (Model 2) → Accuracy = 34.47%

```
Accuracy for epsilon= 0.05 : 93.72
Accuracy for epsilon= 0.1 : 77.88
Accuracy for epsilon= 0.15 : 55.98
```

```
Epsilon: 0.05   0.9218571428571428
Epsilon: 0.1    0.6554285714285715
Epsilon: 0.15   0.3447142857142857
```

Observation and Inference :

- It can be observed that as the value of epsilon increases the Accuracy of the Model decreases significantly.
- As the value of epsilon increases, the addition of the noise to the original image increases, thus becoming difficult even for a human to recognize the digit. Hence, accuracy is expected to drop.
- For small epsilon the image is still the same for the human eyes but the model mis-classifies it.
- Hence, a more robust model is needed to rectify this error .

Adversarial training

- **Training of Model 1 (Tensorflow) (Black-Box Attack)** :
  - The adversarial examples of the original training set are generated using the FGSM for various values of ε .
  - The examples with $\varepsilon = 0.1$ are added to the original training set. Now we have 120,000 training examples.
  - New model is trained using this training set. This model is tested on the adversarial examples of the test set generated using previous model i.e Black Box Attack

- **Training of Model 2 (PyTorch) (White-Box Attack)** :
  - For this approach adversarial examples are generated for the entire training set for two different values of epsilon i.e. 0.05 & 0.1.
  - Then only the adversarial examples are used to fine tune the previously trained model so that it can also classify adversarial inputs.
  - Finally for testing, FGSM is again carried out with new parameters of the model i.e White-Box Attack.

Evaluation of New Models on Adversarial Examples

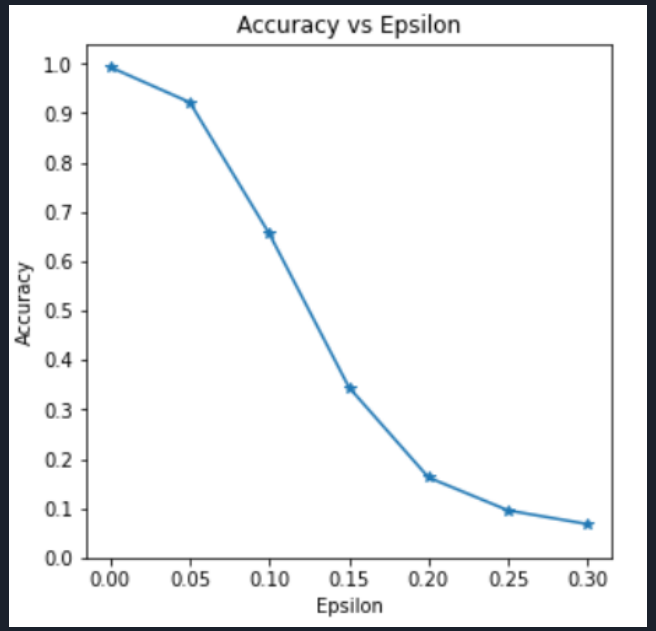
- Model's accuracy on the Adversarial examples of the test set is as follows:
  - **Black-Box Attack** **White-Box Attack**
  - ε = 0.05 (Model 1) → Accuracy = 99.65 % ε = 0.05 (Model 2) → Accuracy = 95.57%
  - ε = 0.10 (Model 1) → Accuracy = 99.77% ε = 0.10 (Model 2) → Accuracy = 86.21%
  - ε = 0.15 (Model 1) → Accuracy = 99.7% ε = 0.15 (Model 2) → Accuracy = 66.89%

```
Accuracy for epsilon= 0.05 : 99.65
Accuracy for epsilon= 0.1 : 99.77
Accuracy for epsilon= 0.15 : 99.7
```

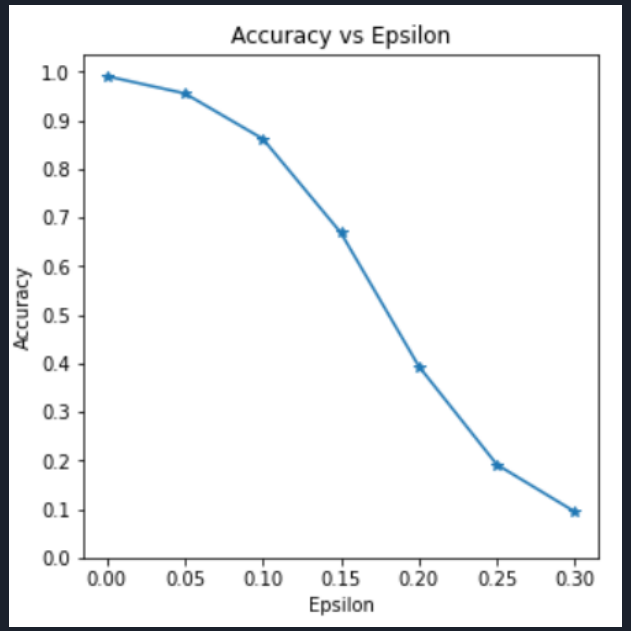
```
Epsilon: 0.05 0.9557142857142857
Epsilon: 0.1  0.8621428571428571
Epsilon: 0.15 0.6688571428571428
```

Accuracy vs Epsilon Plots(White-Box Attack)

- Initial



- Final



Observation & Conclusion

- In case of White-Box Adversarial Attack on original model, as expected, the accuracy quickly decreases as epsilon is increased & the models are not naturally robust.
- By including Adversarial examples in training, it is seen that the new model is quite resistant to Black-Box Attack giving an accuracy greater than 99% for epsilon values of 0.05, 0.1 & 0.15.
- On the other hand for White-Box attack, after fine tuning on adversarial examples, though the model gives a higher accuracy than the previous case for every epsilon>0, the improvement is lesser as compared to Black-Box Attack.
- As expected, Black-Box Attacks are easier to defend than White-Box attacks & there is a need to explore more advanced techniques for protection against White-Box attacks.

Team Contributions

1. **Yash Gadhia** : Creation of Adversarial Examples and Adversarial training (White Box Attack) in PyTorch, including presentation making work.
2. **Prashi Patil** : Creation of Adversarial Examples and Adversarial training (Black Box Attack) in Tensorflow, including presentation making work.
3. **Jay Sawant** : Architecture of CNN Model, training of the Original Model in Tensorflow, including presentation making work.
4. **Lokesh Pawar** : Training of the Original Model in PyTorch, including presentation making work.