| Fall 2024 | CS53331/4331 Adversarial Machine Learning | Assignment 1 |
|---|---|---|

|  |  |
|---|---|
| **Title** | **The Deep Learning Stuff** |
| **Due date** | Monday, October 14th, before the class. |
| **First Name** | Sruthi |
| **Last Name** | Mandalapu |
| **Student ID** | R11906160 |
| **Marks** | 100 |

Note: Please answer the following questions and submit them through Blackboard. Be sure to submit it to assignment 1. DO NOT write the report by hand and submit a scanned version. Just write the answers in a Word document and submit it. Both Word and PDF submissions are accepted.

## Submission Instruction (3 documents)

You are required to submit three documents:

1. ***Report.*** Just fill out the above report and submit it as a Word or PDF document.
2. ***Ipynb file.*** The code that you have written. Preferably in an ipynb document. You can submit it as a .py file as well.
3. ***Txt file of the code.*** We need your code in the .txt file as well. Use whatever way you prefer. The fastest would be to download the file as a .py file and change the extension to .txt

## Objectives

This assignment has three main objectives:

1. Implement untargeted attacks and check the results (FGSM, PGD, DeepFool, and C&W L2)
2. Implement targeted attacks and check the results (FGSM, PGD, and C&W L2)
3. Analyze defense using adversarial training.

## Get started

Download the assignment files from Blackboard. You will need the report (This file), the .ipynb file where you will put your code, the dataset, and the adversarial trained model.

## Dataset

For this assignment, we will use a traffic sign recognition dataset. The German Traffic Sign Recognition Benchmark (GTSRB) dataset consists of almost 51K images of traffic signs. There are 43 classes, and the size of their images is 32×32 pixels. Some of the images are shown below. Please download the dataset from Blackboard or here. You will need the starter code from Blackboard to preprocess it and build a VGG16 model you will attack. More information about the dataset can be found here.

As we did with Assignment 0, it is recommended that you upload the dataset into your personal Google Drive to follow the Colab instructions as they are. Of course, if you prefer to use other than Colab, you will need a similar preprocessing.

## Instruction for Colab (repeated from Assignment 0)

To get started with Google Colab, simply go to Google Colab, sign in with your Google account, and create a new notebook. You can write and execute Python code directly in the notebook. To access your dataset stored in your Google Drive (previous step), first run the following code to mount your Drive:

```python
from google.colab import drive
drive.mount('/content/drive')
```

Follow the authorization steps, and your Drive will be accessible at **/content/drive/My Drive/.** You can then load your dataset into the notebook by providing the correct file path. This part of the code is provided for you in the .ipynb file of Assignment 0. You will need to setup the drive connection and run the code.

To use the free GPU provided by Colab, you can change the runtime to access a GPU by clicking on **"Runtime" > "Change runtime type" and** selecting **"T4 GPU"** from the **Hardware accelerator** dropdown menu. You can always use higher GPU powers at a cost (Colab Pro is $10 per month), but you should be fine with the free version, considering that you start the assignment early enough.

Colab comes with many pre-installed libraries, but if you need to install additional Python packages, you can do so with pip. For example:

```
!pip install library_name
```

Remember to save your work frequently.

After you've completed your work in Google Colab, you can easily download your notebook from Google Colab, go to **"File"** > **"Download"** > **"Download.ipynb"**.

## Other than Colab

If you don't prefer Colab or notebook, you always have the option to run it on your computer (especially if it has a GPU) or access HPCC resources at TTU (needs an account with my permission).

## Additional resources

1. TensorFlow resource https://www.tensorflow.org/
2. PyTorch resources https://pytorch.org/get-started/pytorch-2.0/
3. Deep learning with Python https://dl-with-python.readthedocs.io/en/latest/
4. Get started with Colab https://colab.research.google.com/

## Task 1-untargeted attacks (40 pts)

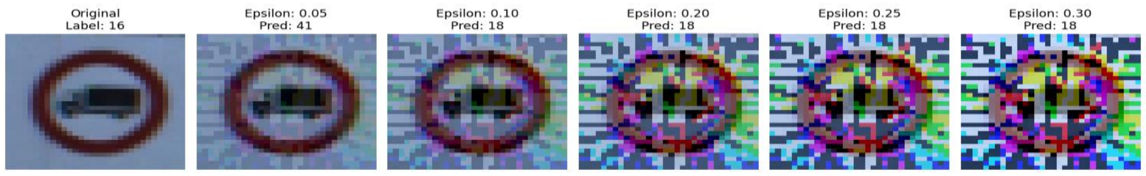
Let's start with basic **non-target white-box attacks**. First, we will implement some non-target white-box attacks we studied in class. Your downloaded code from Blackboard will build a VGG16 model for you. You will attack that deep learning model throughout the assignment.

This task aims to implement the following attacks: Fast Gradient Sign Method (FGSM), Projected Gradient Descent (PGD), Deep Fool, and C&W with L2 norm. You don't need to implement these attacks from scratch. Code for them can be in several libraries, including Adversarial Robustness Toolbox (ART), cleverhans, or scratchai (Just traditional libraries. Maybe there are better ones now). Using ART for this assignment is recommended, but using any other libraries of your choice is also acceptable. This notebook is a good start on how to use ART, and multiple other notebooks are available here.

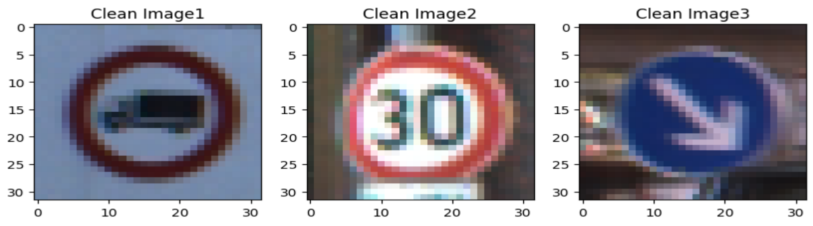
Your task is to apply attacks to create non-target adversarial examples using the first 500 images of your test set (check imgs_adv and labels_adv in your code). For FSGM, PGD, and DeepFool, apply perturbations magnitudes: $\epsilon$= [0.05, 0.1, 0.2, 0.25, 0.3]. For C&W, use the L2 norm to perform the attack. You should provide us with the following results.

**Note:** [10 pts] will go to your code for attack implementation. The comparison code evaluation will be included in its respective questions.

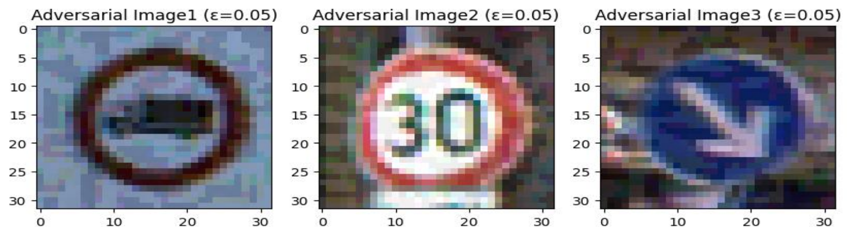
1. [4 pts] Plot the clean image vs adversarial image for three images of your choice. This should happen for all perturbation magnitudes (if any) and for all attacks. An example of FGSM is shown below. Do it for FGSM, PGD, DeepFool, and C&W (separate figure for each)



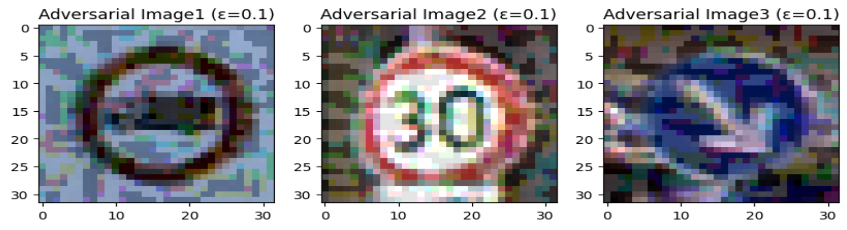
For FGSM:



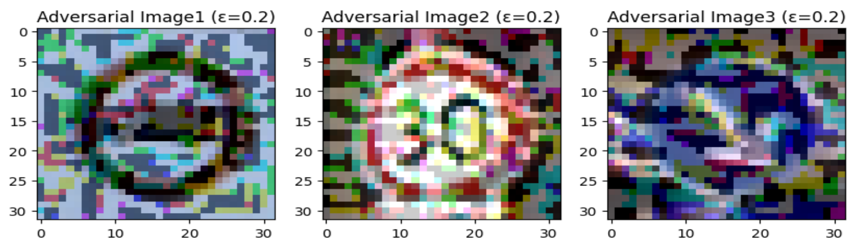
```
Accuracy on adversarial test data: 27.40%
Average perturbation: 0.05
```


Adversarial Image1 (ε=0.05)
Adversarial Image2 (ε=0.05)
Adversarial Image3 (ε=0.05)

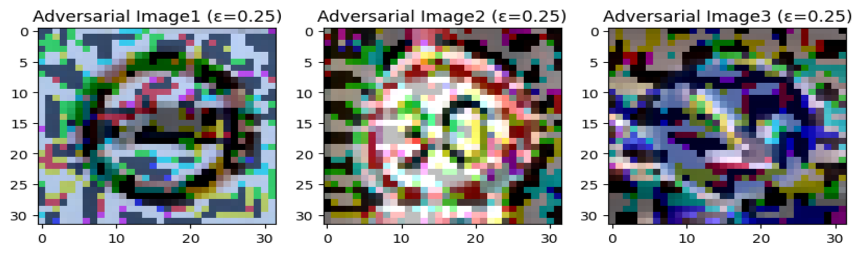
```
Accuracy on adversarial test data: 22.60%
Average perturbation: 0.09
```


Adversarial Image1 (ε=0.1)
Adversarial Image2 (ε=0.1)
Adversarial Image3 (ε=0.1)

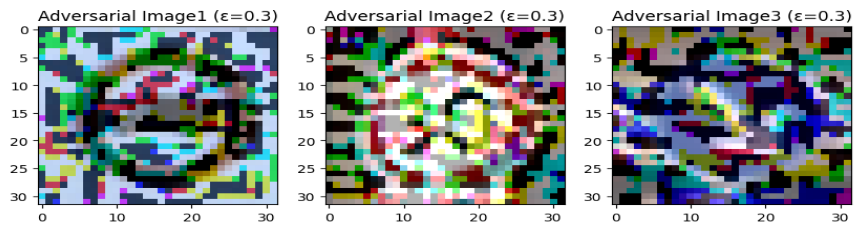
```
Accuracy on adversarial test data: 17.20%
Average perturbation: 0.17
```


Adversarial Image1 (ε=0.2)
Adversarial Image2 (ε=0.2)
Adversarial Image3 (ε=0.2)

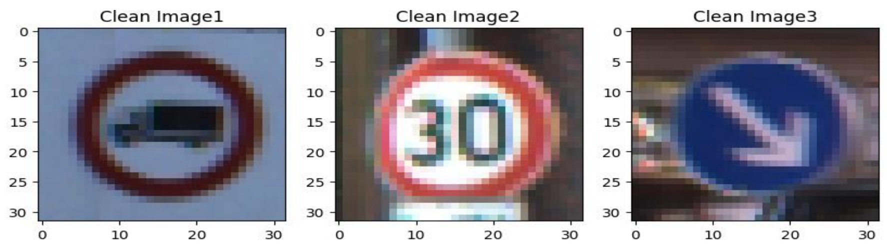
```
Accuracy on adversarial test data: 15.40%
Average perturbation: 0.21
```


Adversarial Image1 (ε=0.25)
Adversarial Image2 (ε=0.25)
Adversarial Image3 (ε=0.25)

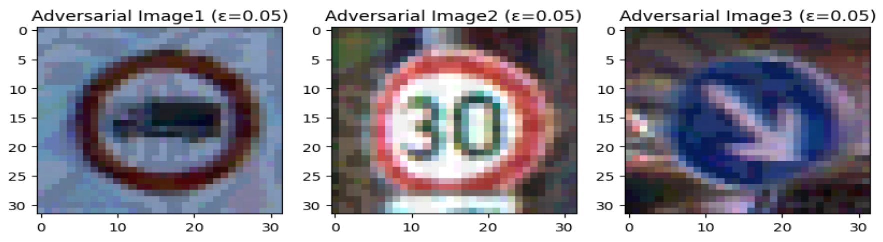
```
Accuracy on adversarial test data: 14.20%
Average perturbation: 0.24
```


Adversarial Image1 (ε=0.3)
Adversarial Image2 (ε=0.3)
Adversarial Image3 (ε=0.3)

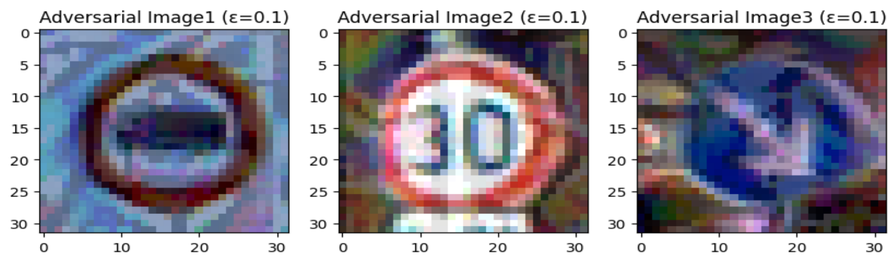
For PGD:



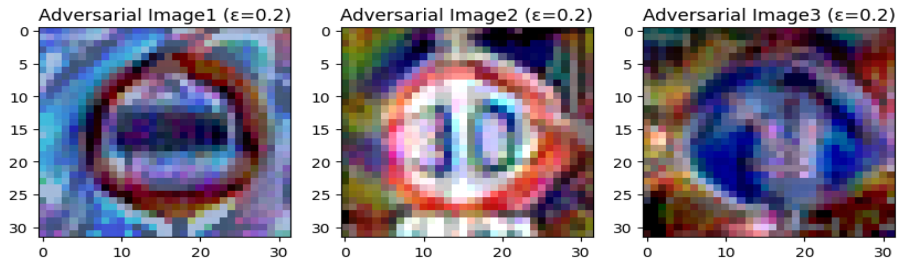
```
Accuracy on adversarial test data: 15.60%
Average perturbation: 0.04
```


Adversarial Image1 (ε=0.05)
Adversarial Image2 (ε=0.05)
Adversarial Image3 (ε=0.05)

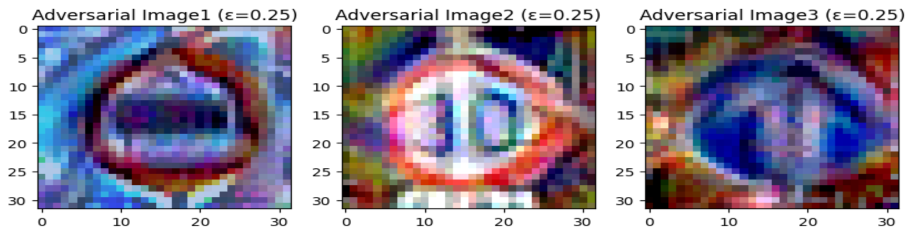
```
Accuracy on adversarial test data: 13.60%
Average perturbation: 0.07
```


Adversarial Image1 (ε=0.1)
Adversarial Image2 (ε=0.1)
Adversarial Image3 (ε=0.1)

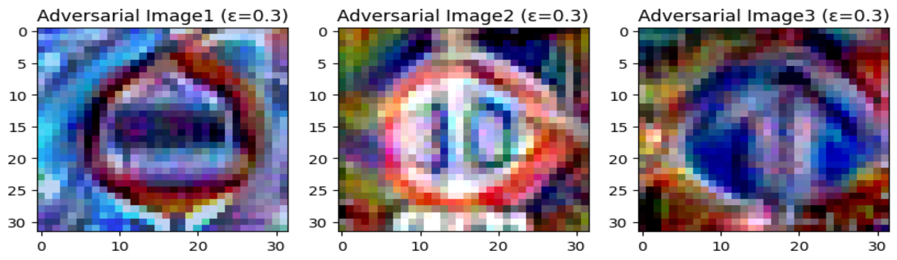
```
Accuracy on adversarial test data: 13.60%
Average perturbation: 0.11
```


Adversarial Image1 (ε=0.2)
Adversarial Image2 (ε=0.2)
Adversarial Image3 (ε=0.2)

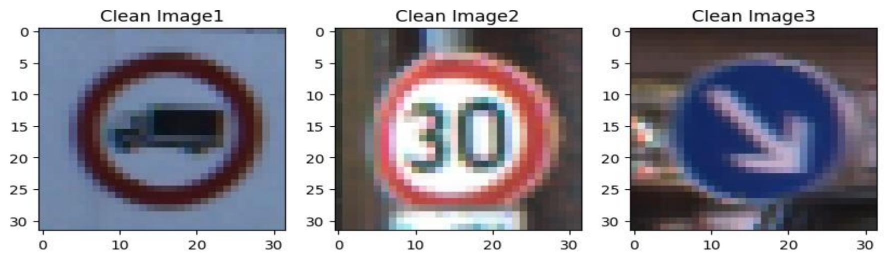
```
Accuracy on adversarial test data: 13.60%
Average perturbation: 0.12
```


Adversarial Image1 (ε=0.25)
Adversarial Image2 (ε=0.25)
Adversarial Image3 (ε=0.25)

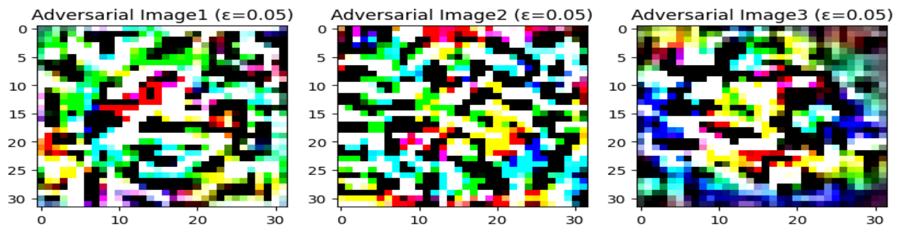
```
Accuracy on adversarial test data: 13.60%
Average perturbation: 0.12
```


Adversarial Image1 (ε=0.3)
Adversarial Image2 (ε=0.3)
Adversarial Image3 (ε=0.3)

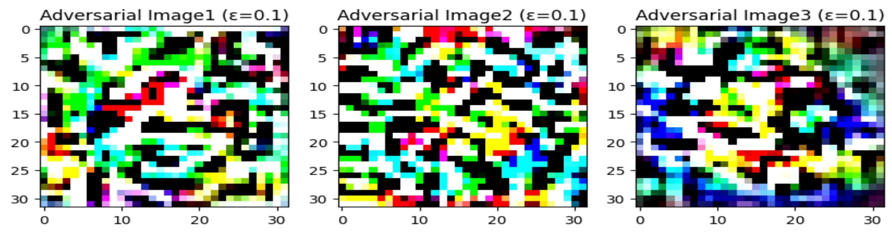
For Deepfool:



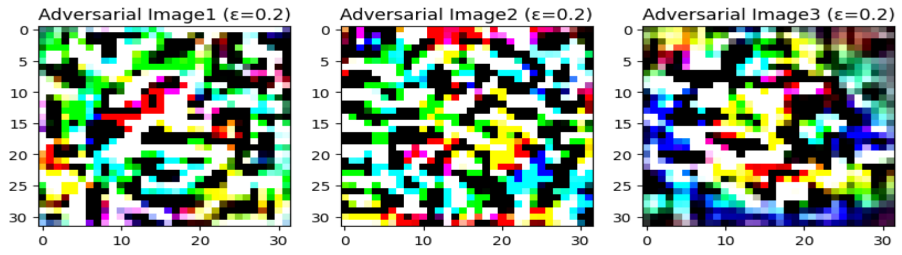
```
Accuracy on adversarial test data: 16.20%
Average perturbation: 0.17
```


Adversarial Image1 (ε=0.05)
Adversarial Image2 (ε=0.05)
Adversarial Image3 (ε=0.05)

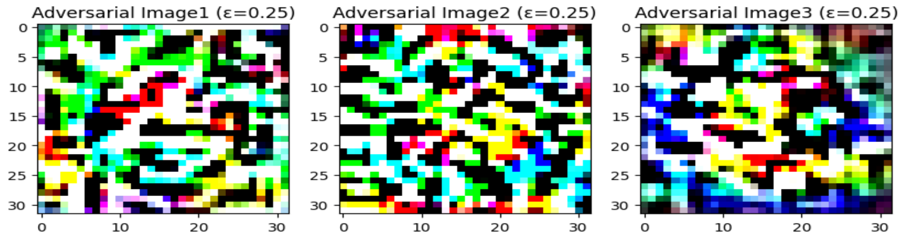
```
Accuracy on adversarial test data: 15.80%
Average perturbation: 0.17
```


Adversarial Image1 (ε=0.1)
Adversarial Image2 (ε=0.1)
Adversarial Image3 (ε=0.1)

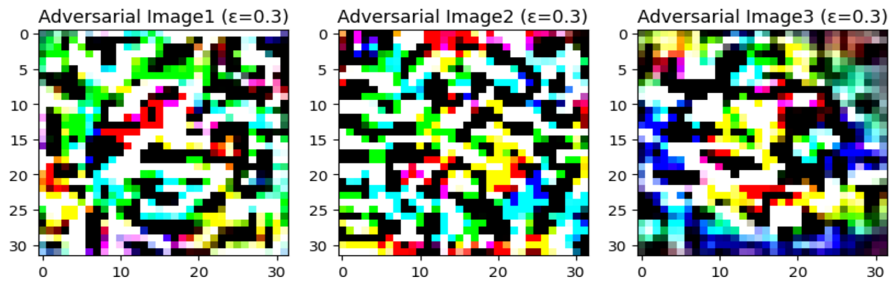
```
Accuracy on adversarial test data: 16.40%
Average perturbation: 0.18
```



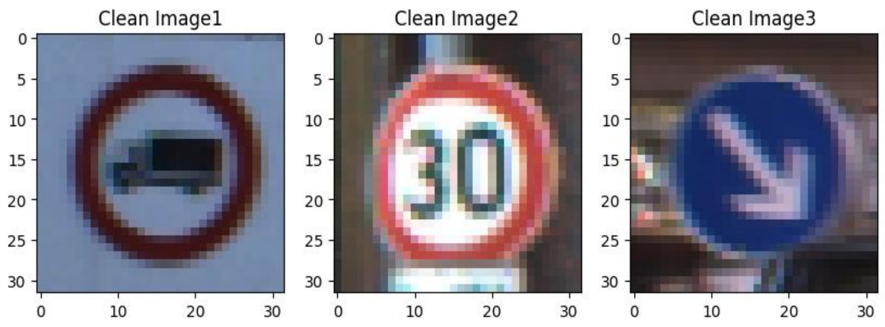
```
Accuracy on adversarial test data: 17.00%
Average perturbation: 0.18
```



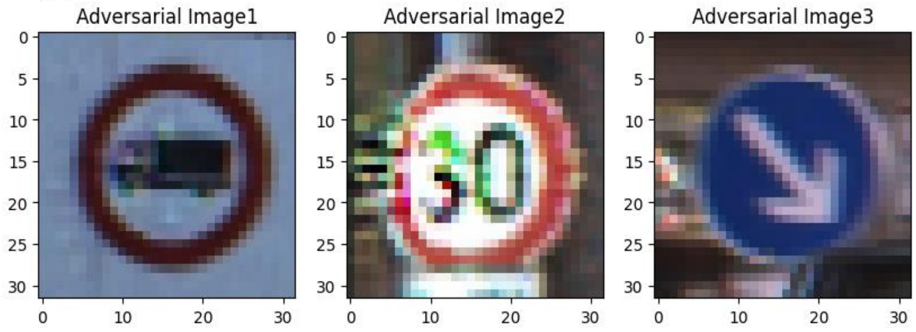
```
Accuracy on adversarial test data: 16.80%
Average perturbation: 0.18
```



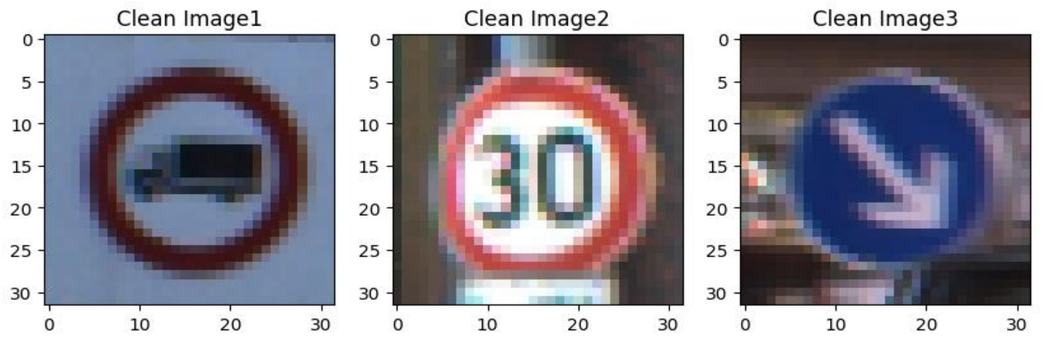
For C&W:



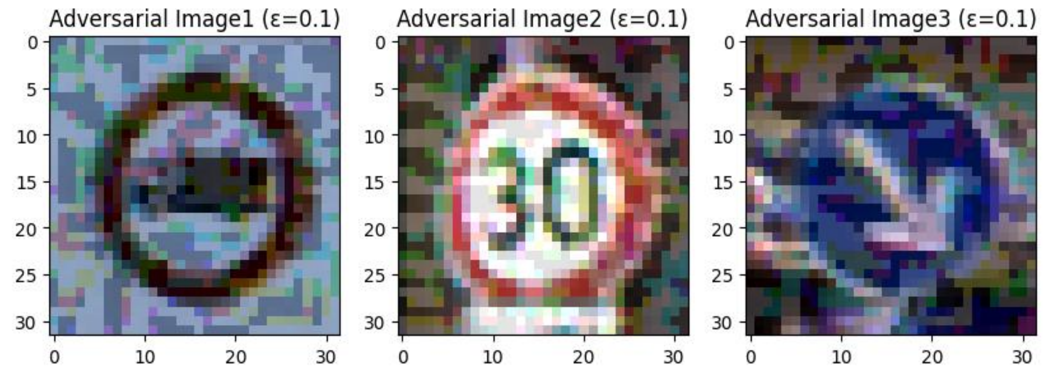
```
Accuracy on adversarial test data: 26.60%
Average perturbation: 0.01
```



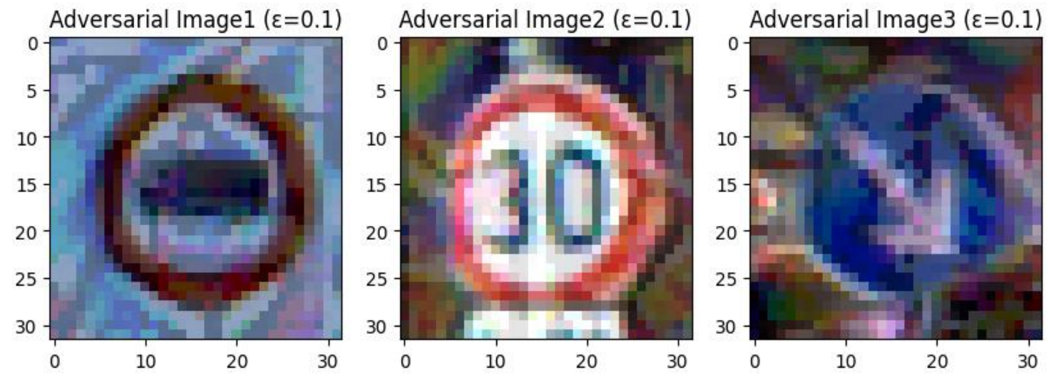
2. [6 pts] For two images of choice, plot the clean image vs. adversarial images of all attacks (if the perturbation is needed, choose $\epsilon$= [0.1]). You should produce one similar figure but with the four attacks and the original.



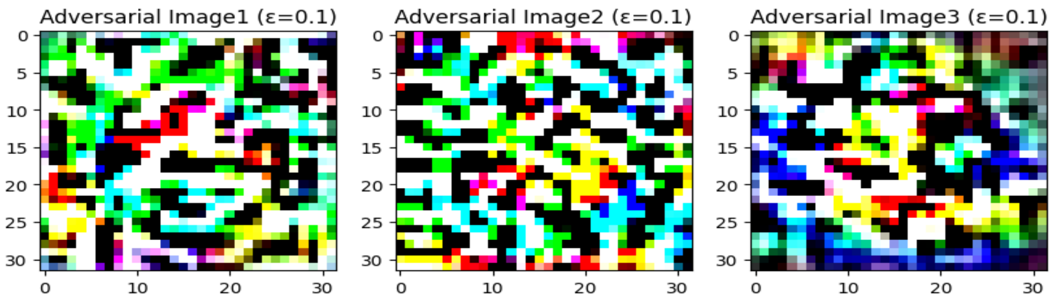
FGSM:


Adversarial Image1 (ε=0.1)
Adversarial Image2 (ε=0.1)
Adversarial Image3 (ε=0.1)

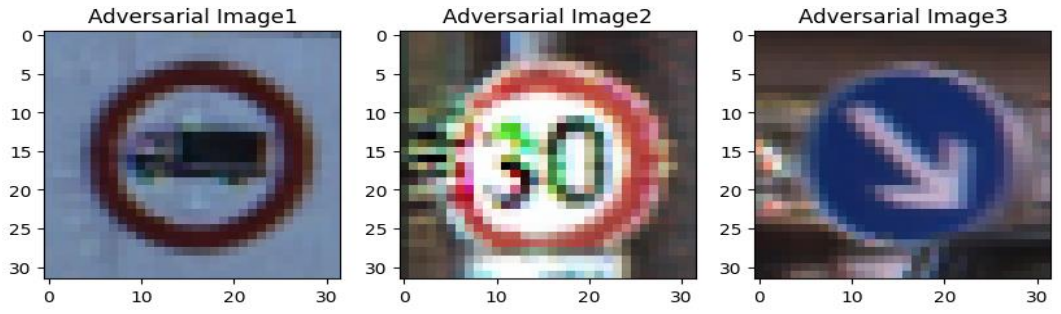
PGD:


Adversarial Image1 (ε=0.1)
Adversarial Image2 (ε=0.1)
Adversarial Image3 (ε=0.1)

Deep fool:


Adversarial Image1 (ε=0.1)
Adversarial Image2 (ε=0.1)
Adversarial Image3 (ε=0.1)

C & W:



3. [5 pts] Fill out Table 1 and Table 2 for accuracy and add the noise of each attack. Adversarial classification accuracy is the accuracy on adversarial samples only.

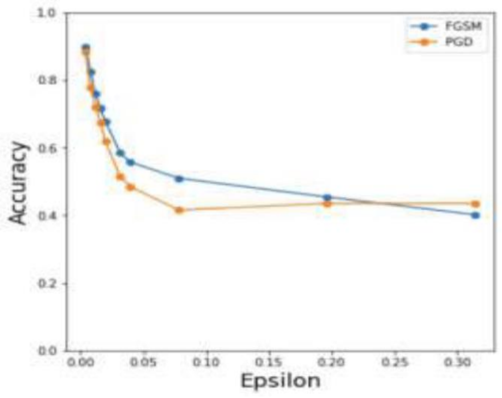
Table 1: Adversarial classification accuracy of the models

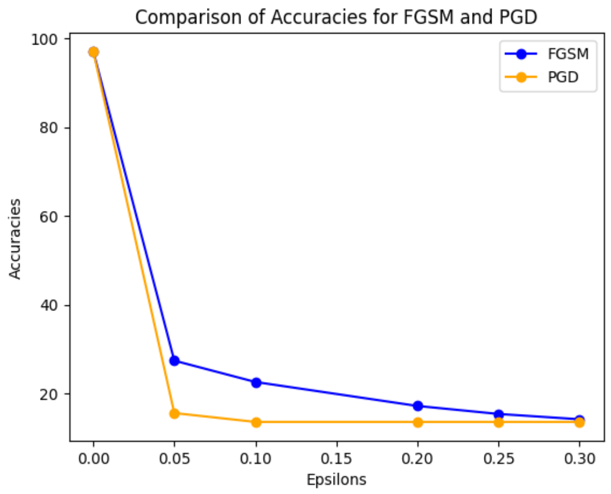
| **Model** | **Clean image** | **Adversarial images with $\epsilon$=0.05** | **Adversarial images with $\epsilon$=0.2** | **Adversarial images with $\epsilon$=0.3** |
|---|---|---|---|---|
| VGG16-FGSM | 97.05% | 27.40% | 17.20% | 14.20% |
| VGG16-PGD | 97.05% | 15.60% (for 40 iter) | 13.60% (for 40 iter) | 13.60% (for 40 iter) |
| VGG16-DeepFool | 97.05% | 16.20% (for 8 iter) | 16.40% (for 8 iter) | 16.80% (for 8 iter) |
| | **Clean image** | **Adversarial images** | | |
| C&W L2 | 97.05% | 26.60% (for 5 iterations) | | |

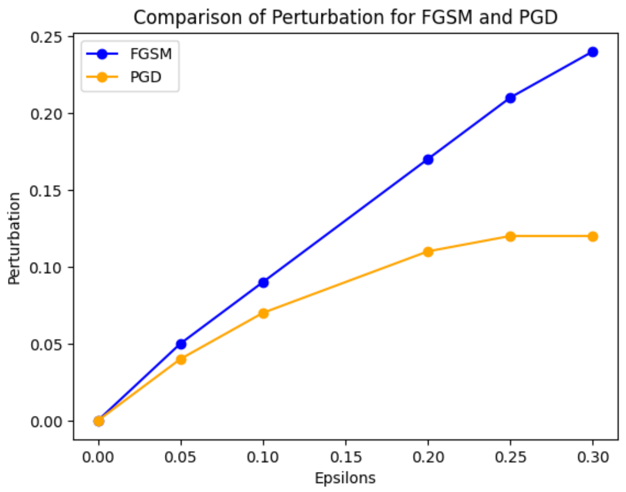
Table 2: Noise of the models

| **Model** | **Adversarial images with $\epsilon$=0.05** | **Adversarial images with $\epsilon$=0.2** | **Adversarial images with $\epsilon$=0.3** |
|---|---|---|---|
| VGG16-FGSM | 0.05 | 0.17 | 0.24 |
| VGG16-PGD | 0.04 | 0.11 | 0.12 |
| VGG16-DeepFool | 0.17 | 0.18 | 0.18 |
| | **Adversarial images** | | |
| C&W L2 | 0.01 | | |

4. [5 Points] Plot accuracy versus perturbation $\epsilon$ for FSGM and PGD adversarial attacks (similar to the example in the following figure).







5. [5 pts] Briefly provide insights on the model's performance under attacks and any other observations regarding the results.

   Overall, all the given attack techniques "FGSM, PGD, Deepfool, C&W L2" producing out lower accuracies i.e the attack performance is really good, coming to fgsm and pgd – the time complexity of both the attacks while running is lower when compared to deepfool, C&W. Based on the iterations, PGD was implemented on 40 iterations, Deepfool over 8 iterations, C&W over 5 iterations. The time complexity if directly proportional to the number of iterations, and is also proportional to adding more noise to the inputs. The more is the number of iterations the more is the amount of perturbation added to the data. As deepfool takes longer time, we can say perturbing more noise gives efficient results i.e giving more iterations. If we see the results, on increase in the epsilon value, the noisier image is generated which often produces wrong predictions. By looking through the above graph PGD achieves lower accuracy with lower epsilon values when compared to FGSM.

   Overall deductions used in code:

   - from art.estimators.classification import KerasClassifier – it is a module in Adversarial Robustness Toolbox (ART) is used to wrap Keras models for adversarial machine learning tasks. supports various types of adversarial attacks, such as Fast Gradient Sign Method (FGSM), Projected Gradient Descent (PGD). Enables adversarial and non-adversarial evaluation
   - from art.attacks.evasion import attack_method – the attack method can be FastGradientMethod, ProjectedGradientDescent, DeepFool, CarliniL2Method .. Produces various evasion attack techniques that can be applied to fool machine learning models by generating adversarial examples.
   - classifier = KerasClassifier(model=model, clip_values=(0, 1)) – this classifier will work seamlessly with ART's attack and evaluation methods. clip values refers range of values that the input data can take – Ensures the input data is normalized between 0 and 1.
   - attack_pgd = ProjectedGradientDescent(estimator=classifier, eps=i, eps_step=0.01, max_iter=40) – initializes PGD attack, uses KerasClassifier, eps describes the perturbation value, eps_step is the step size for the iteration (defined by max_iter -> as number of iterations)
   - imgs_adv_pgd = attack_pgd.generate(imgs_adv) – generates adversarial images by applying attacks
   - loss_test, accuracy_test = model.evaluate(imgs_adv_pgd, labels_adv) – Allows in model evaluation in terms of accuracy and loss.
   - perturbation = np.mean(np.abs((imgs_adv_pgd - imgs_adv))) – describes the amount of noise induced to the data.

## Task 2-targeted attacks (30 pts)

Now, let’s do some **target attacks** with white-box assumptions. Use the images with the Stop sign (label 14) from the overall test set for this task. There are around 270 images of that kind. Implement FGSM attacks on the Stop sign images to misclassify them as speed 30 sign images (label 1). Apply perturbations magnitudes: $\epsilon$= [0.05, 0.1, 0.2, 0.25, 0.3] for these attacks and report the classification accuracy on the Stop sign images and the Speed Limit 30 sign images.

Apply the same thing using PGD attacks and compare the results.

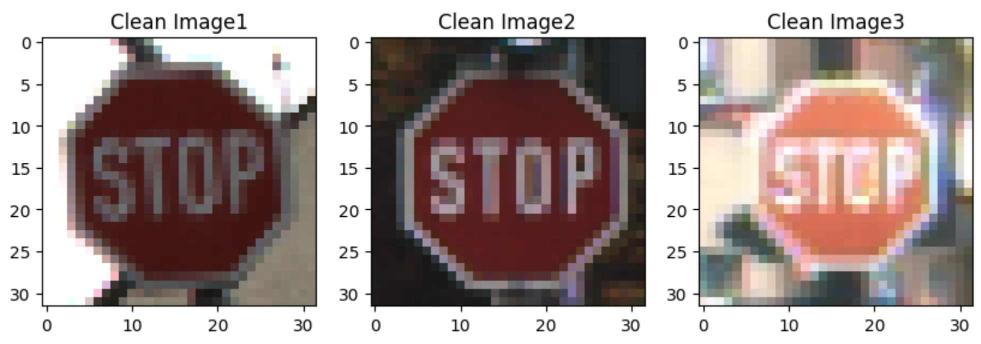
Apply the same thing using C&W L2 attacks (perturbations magnitudes) and compare the results.

Then, answer the following questions.

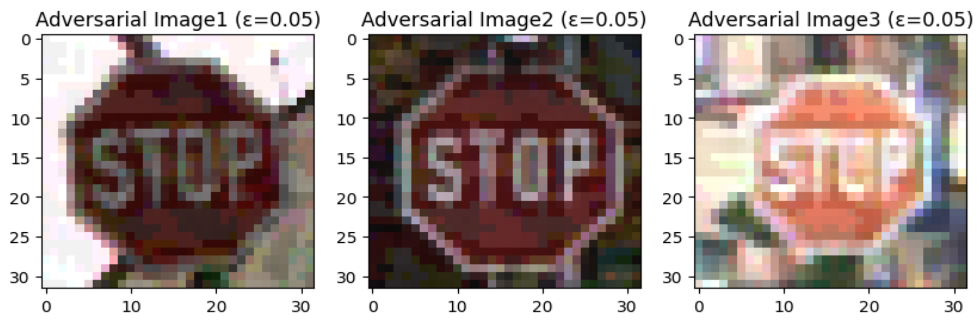
**Note:** [10 pts] will go to your code for attack implementation. The comparison code evaluation will be included in its respective questions.

1. [5 pts] Plot the clean image vs adversarial image for a Stop sign image of your choice. This should happen for all perturbation magnitudes (if any) and all attacks.

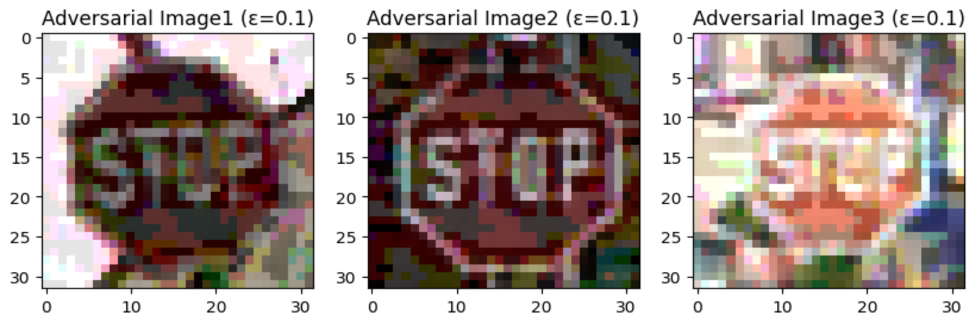
   For FGSM (Stop):



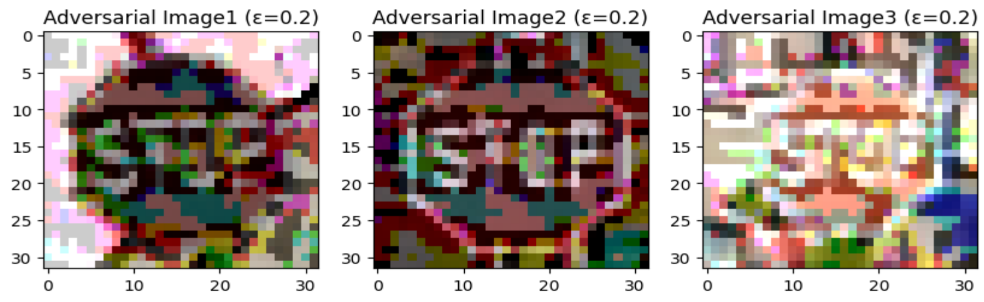
```
Accuracy on adversarial test data: 39.63%
Average perturbation: 0.05
```


Adversarial Image1 (ε=0.05)
Adversarial Image2 (ε=0.05)
Adversarial Image3 (ε=0.05)

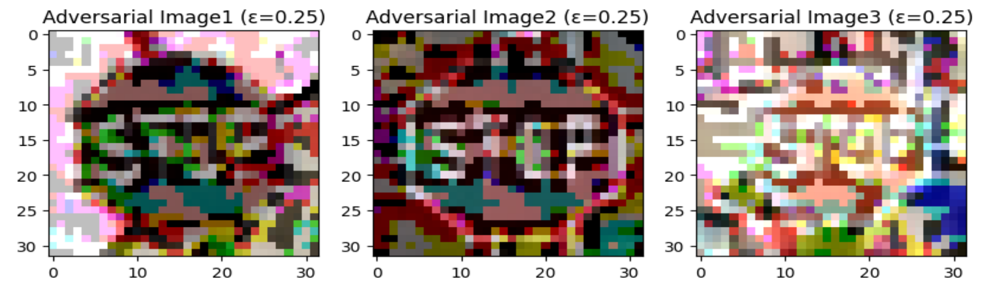
```
Accuracy on adversarial test data: 1.85%
Average perturbation: 0.09
```


Adversarial Image1 (ε=0.1)
Adversarial Image2 (ε=0.1)
Adversarial Image3 (ε=0.1)

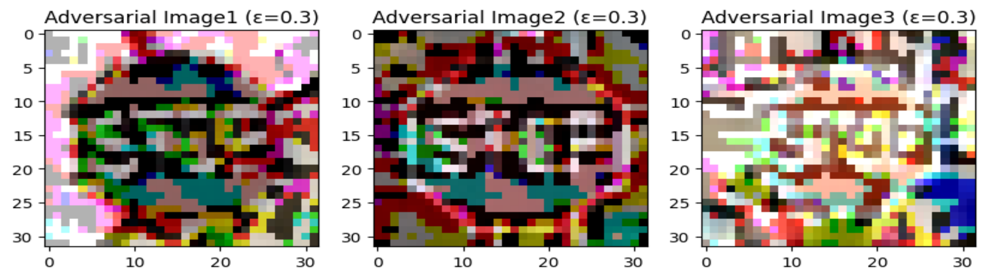
```
Accuracy on adversarial test data: 0.00%
Average perturbation: 0.18
```


Adversarial Image1 (ε=0.2)
Adversarial Image2 (ε=0.2)
Adversarial Image3 (ε=0.2)

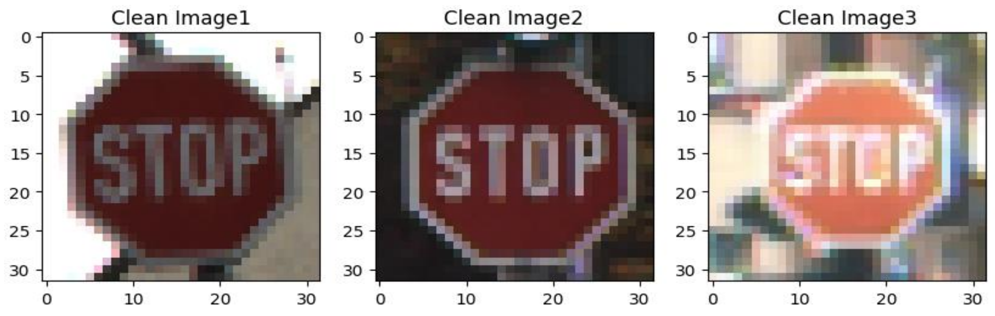
```
Accuracy on adversarial test data: 0.00%
Average perturbation: 0.21
```


Adversarial Image1 (ε=0.25)
Adversarial Image2 (ε=0.25)
Adversarial Image3 (ε=0.25)

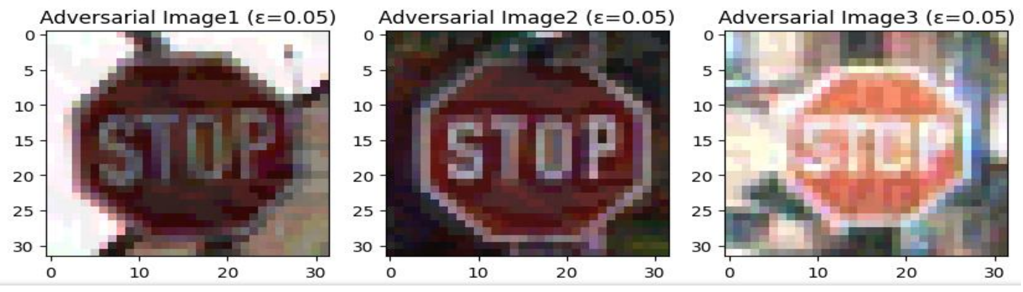
```
Accuracy on adversarial test data: 0.00%
Average perturbation: 0.25
```


Adversarial Image1 (ε=0.3)
Adversarial Image2 (ε=0.3)
Adversarial Image3 (ε=0.3)

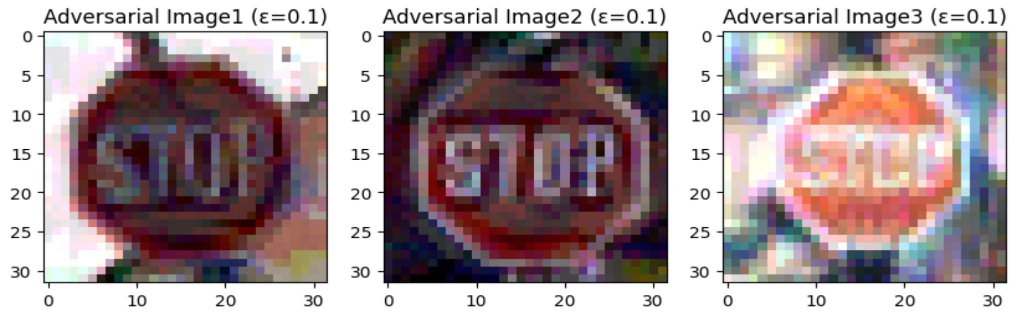
For PGD (Stop):



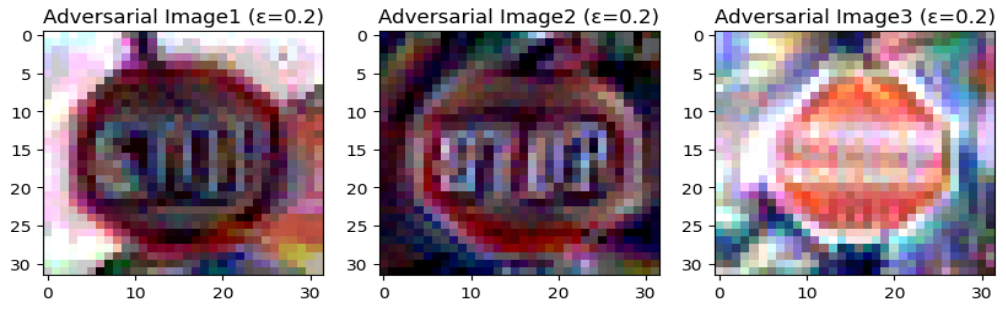
```
Accuracy on adversarial test data: 4.07%
Average perturbation: 0.04
```


Adversarial Image1 (ε=0.05)
Adversarial Image2 (ε=0.05)
Adversarial Image3 (ε=0.05)

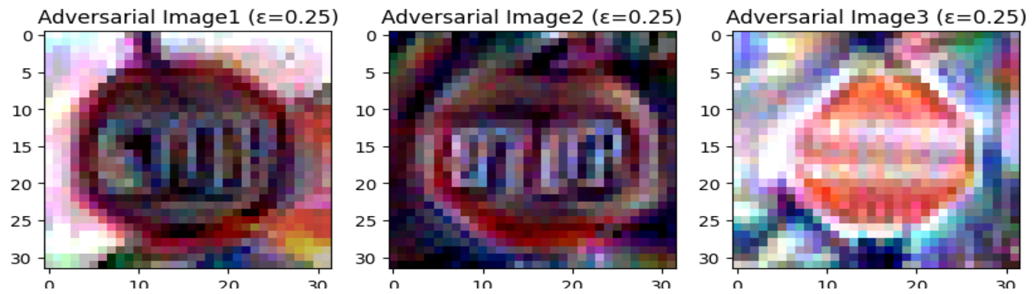
```
Accuracy on adversarial test data: 2.22%
Average perturbation: 0.07
```


Adversarial Image1 (ε=0.1)
Adversarial Image2 (ε=0.1)
Adversarial Image3 (ε=0.1)

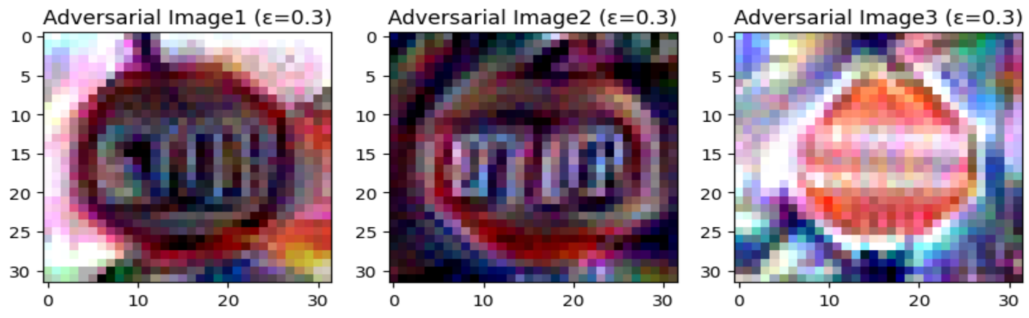
```
Accuracy on adversarial test data: 2.22%
Average perturbation: 0.11
```


Adversarial Image1 (ε=0.2)
Adversarial Image2 (ε=0.2)
Adversarial Image3 (ε=0.2)

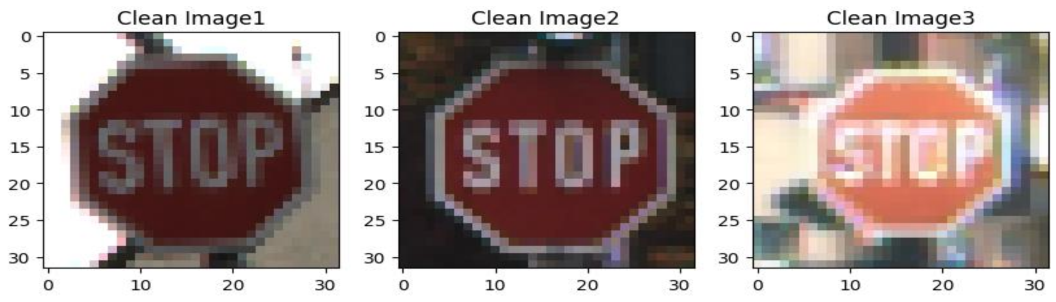
```
Accuracy on adversarial test data: 2.22%
Average perturbation: 0.12
```

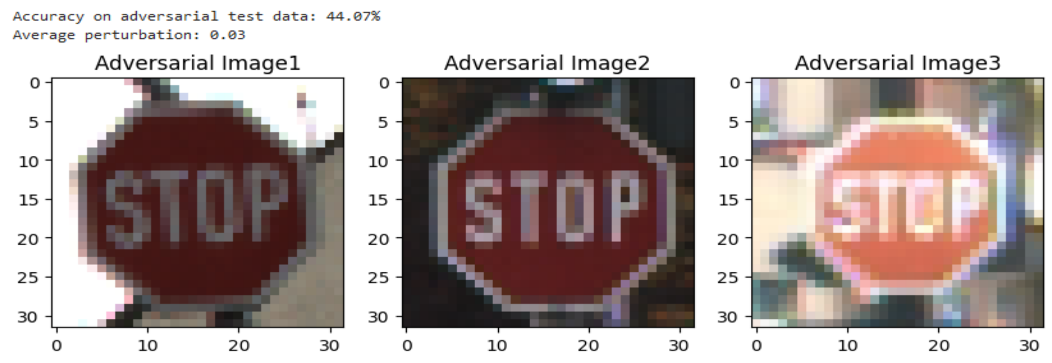

Adversarial Image1 (ε=0.25)
Adversarial Image2 (ε=0.25)
Adversarial Image3 (ε=0.25)

```
Accuracy on adversarial test data: 2.22%
Average perturbation: 0.12
```


Adversarial Image1 (ε=0.3)
Adversarial Image2 (ε=0.3)
Adversarial Image3 (ε=0.3)

For C & W (Stop):





```
Accuracy on adversarial test data: 44.07%
Average perturbation: 0.03
```

2. [10 pts] Fill out Table 3 for accuracies.

Table 3: Classification accuracy of the model on adversarial original and target class images.

| $\epsilon$ | **FGSM attack – Stop sign images** | **FGSM attack – Speed Limit 30 sign images** | **PGD attack – Stop sign images** | **PGD attack – Speed Limit 30 sign images** |
|---|---|---|---|---|
| 0.05 | 39.63% | 30.69% | 4.07% | 3.89% |
| 0.1 | 1.85% | 2.08% | 2.22% | 0.00% |
| 0.2 | 0.00% | 1.53% | 2.22% | 0.00% |
| 0.25 | 0.00% | 1.39% | 2.22% | 0.00% |
| 0.3 | 0.00% | 1.39% | 2.22% | 0.00% |
| | **C&W attack – Stop sign images** | | **C&W attack – Speed Limit 30 sign images** | |
| - | 44.07% | | 2.08% | 2.08% |

3. [5 pts] Briefly provide insights on the model's performance under attacks and any other observations regarding the results.
As, we can see the targeted attacks performs more efficiently when compared to non-targeted attacks. It was almost leaving up with accuracy 0% as the amount of perturbation into the dataset is increased. As the increase in the noise "the model is leaving up with 0% accuracy". That means the attack is working more efficiently in scenario "Targeted" compared with "Non-targeted". In this targeted attack from the overall GTSRB dataset we are consuming data which contains STOP sign images, and Speed Limit 30km/hr images. On each of dataset it is induced with perturbation values varying between ranges 0.05 and 0.3, as the increase in epsilon value the accuracy keeps on reducing and reaching to a point of 0 accuracy, which explains about efficiency of the attack. If we see C&W attack – it should be executed over good number of iterations to validate the accuracies – since the time complexity for the C & W is too long we have used less number of iterations. Coming to comparison STOP and Speed Limit – Speed limit have more dataset (which allowed lesser accuracy).
Modifications in dataset: To get STOP and Speed Limit
   - STOP_SIGN_LABEL = np.where(signnames == 'Stop')[0][0] – allows in extracting the label for the stop sign. If signnames == 'Speed limit (30km/h)' it gives out the speed limit of 30km/hr images.
   - stop_sign_indices = np.where(labels_test == STOP_SIGN_LABEL)[0] – produces all the indexes which contain stop signs
   - imgs_stop_signs = imgs_test[stop_sign_indices], labels_stop_signs = labels_test[stop_sign_indices] – this allows in producing out all the images and labels having respective indices.

To conclude, Targeted attacks produces almost 0% accuracy on perturbation, which deduces the attack is more efficient compared to that of results in Task1.

## Task 3- Adversarial Defense (20 pts)

Now, let's defend against adversarial attacks. This notebook is a good start for adversarial training. Due to its complexity and the time taken to build an adversarial-trained model, we decided to give the model to you, and your only task will be to analyze it. The VGG classifier from earlier parts is used to build the adversarial model. New adversarial images were generated and used to build the new model. This model is provided in your assignment, and you will need to analyze it. Your tasks for this part will

- Analyze the code (commented) provided to build the adversarial model and answer question 1.
- Measure the performance of the trained robust model and compare it to the original VGG model used in earlier parts. For the attack part, we will use FGSM and PGD on the samples we reserved for adversarial generation in the earlier parts (use eps=0.1). You will probably need to implement your evaluation for things to work here. Record the classification accuracies that have been attacked.

1. [5 pts] Answer the following questions about the provided model or code for it.
   a. What attacks were used to build the model
      The model is built as a defense for PGD attack.
   b. What perturbation (if any) was used to build the attack samples?
      The attack samples were built on epsilon = 0.1
   c. What percentage of training samples was used to generate the adversarial samples used in the built model?
      50 % of training samples
   d. How many epochs were used?
      Number of epochs – 25
   e. What needs to be changed in the code to generate an adversarial-trained model based on C&W attacks? Write the code for it as an answer.
      #attackPgd = ProjectedGradientDescent(estimator = robustClassifier,eps=0.1, max_iter=40, batch_size=64) - This line to be modified with: attack_cw = CarliniL2Method(classifier=classifier,max_iter=5,learning_rate=0.1, initial_const=1e0)

2. [10 pts] Fill out Table 4 for accuracies.

Table 4: Classification accuracy of the traditional and adversarial trained model

| **Setup** | **VGG16** | **Adversarial trained model** |
|---|---|---|
| **Clean train images** | 97.05% | 99.73% |
| **Clean subset of test images** | 93.80% | 96.60% |
| **FGSM attacked subset of test images** | 46.80% | 44.20% |
| **PGD attacked subset of test images** | 43.60% | 81.20% |

3. [5 pts] Briefly provide insights on the model's performance under attacks and any other observations regarding the results.
   This section explains about a Adversarially built defense technique on evasion attacks. The defense model is built against PGD attack – which works really well in removing the noise and procures the accuracy almost equal to 80 percent. When there is no defense, it produced lesser accuracy which is almost (40%). As this model is completely built against PGD defense technique – the model performs poorly in case of FGSM – lesser accuracy. So, as we observe the pgd attack is prevented, while leaving FGSM with almost same accuracies before and after application of adversarial model. It almost procures – good improvement in accuracies (working as pgd defence). Discussing about model parameters – the epsilon value is taken as 0.1. By default the pgd takes 100 iterations – and it is implemented by using respective parameters. Though, there is noise induced by using PGD defense – the model gives more accuracy compared to the past results – specifying the prominence of defense.
   FGSM attack → Lower Accuracy Values (Attack performance is good – no defense)
   PGD Attack → Higher Accuracy Values (Attack performance is low – specifying implicating a defense mechanism)

## Submission Instruction (3 documents)

You are required to submit three documents:

1. ***Report.*** Just fill out the above report and submit it as a Word or PDF document.
2. ***Ipynb file.*** The code that you have written. Preferably in an ipynb document. You can submit it as a .py file as well.
3. ***Txt file of the code.*** We need your code in the .txt file as well. Use whatever way you prefer. The fastest would be to download the file as a .py file and change the extension to .txt