**Indian Institute of Technology, Jodhpur**

**Dependable AI | Assignment 2**

Topic : Adversarial Attacks - Attack, Detection and Mitigation

Submitted by: Sanyam Jain (P20QC001)

**Part (A):**

----------------------------------------------------------------------------------------------------------------

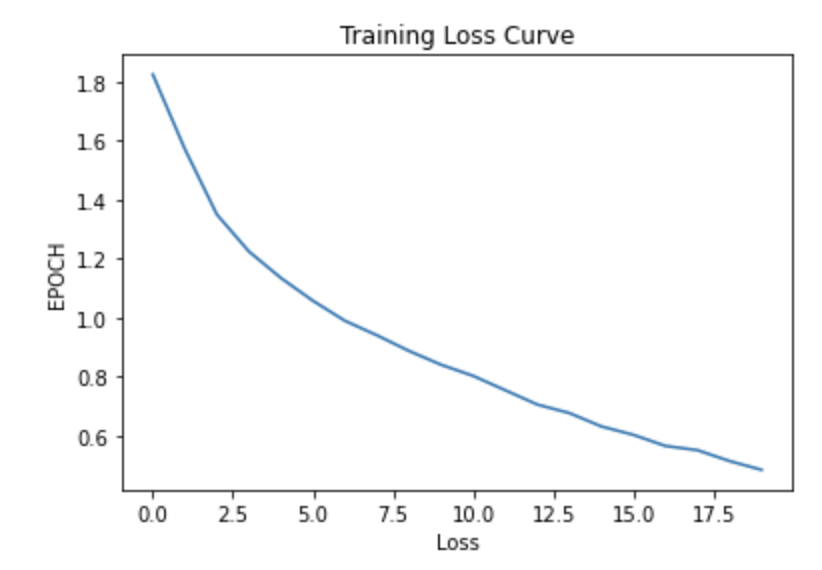
**Theory:**

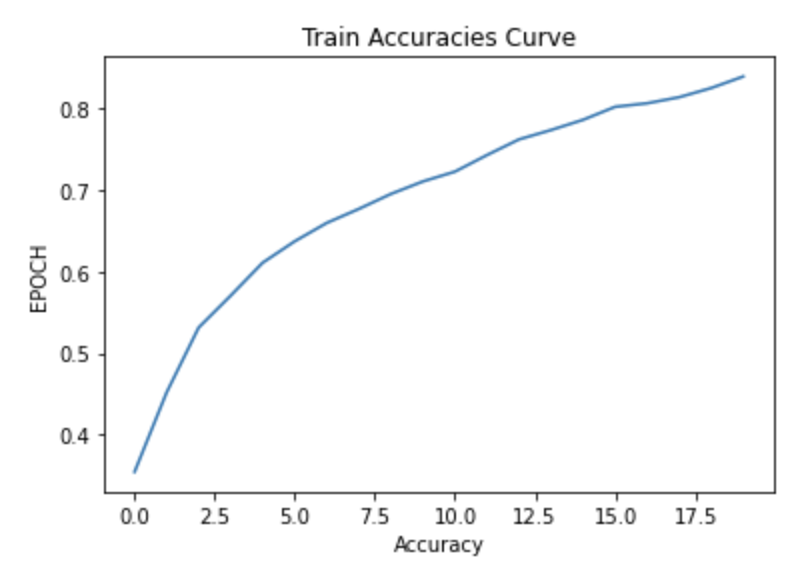
Pretrained model of ResNet50 is chosen as base model without weights. A new model is added to base model with Pooling and FC. This model is trained with our dataset. On 10477 samples of testing dataset, the model resulted a total loss of 71.08% and 77.48% accuracy. This is set procedure of importing image data from google drive and then training on our model which we have already seen in previous assignments too.

Report the accuracy, training-testing loss graph, classification report:

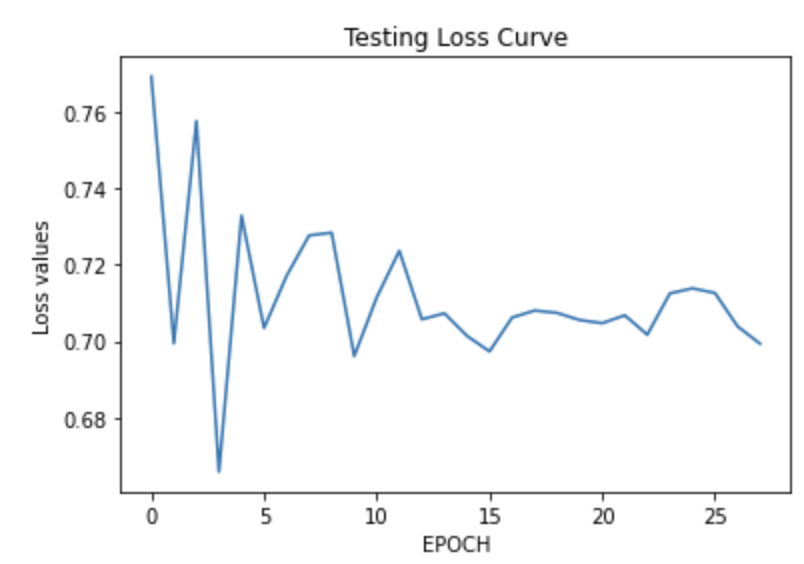
**Accuracy**: 84.41%

**Training loss and accuracy curve:**

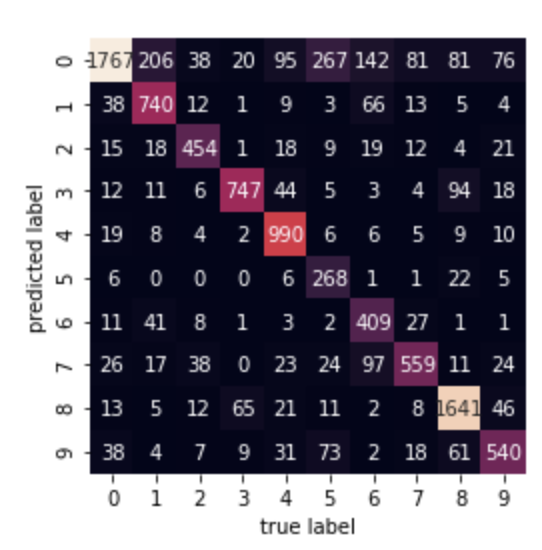




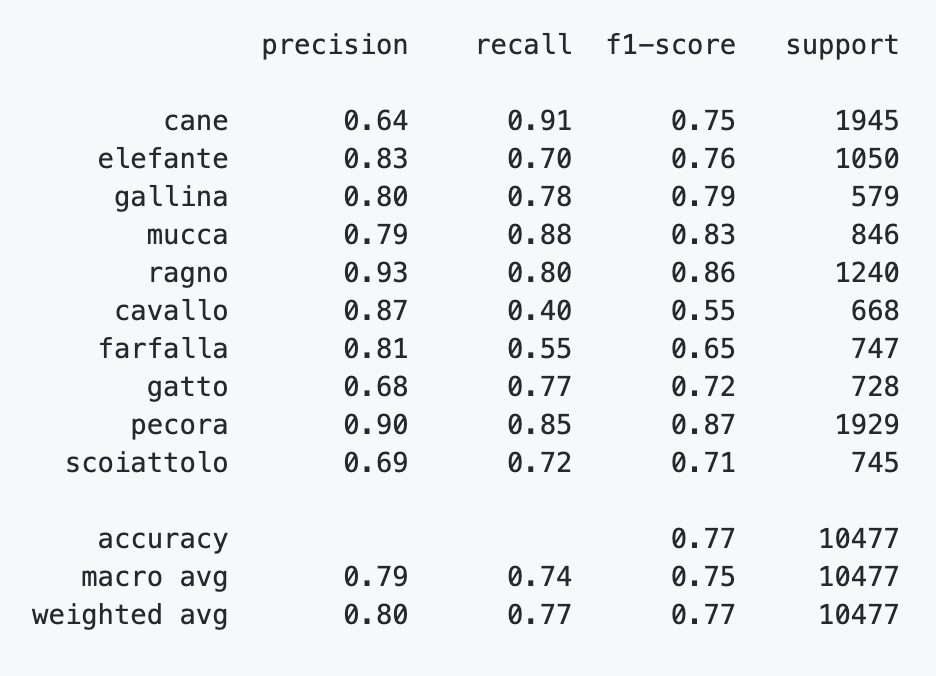
**Testing Loss Curve:**



**Confusion Matrix:**



**Classification Report:**

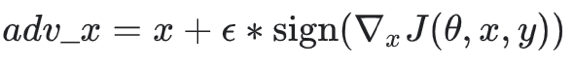


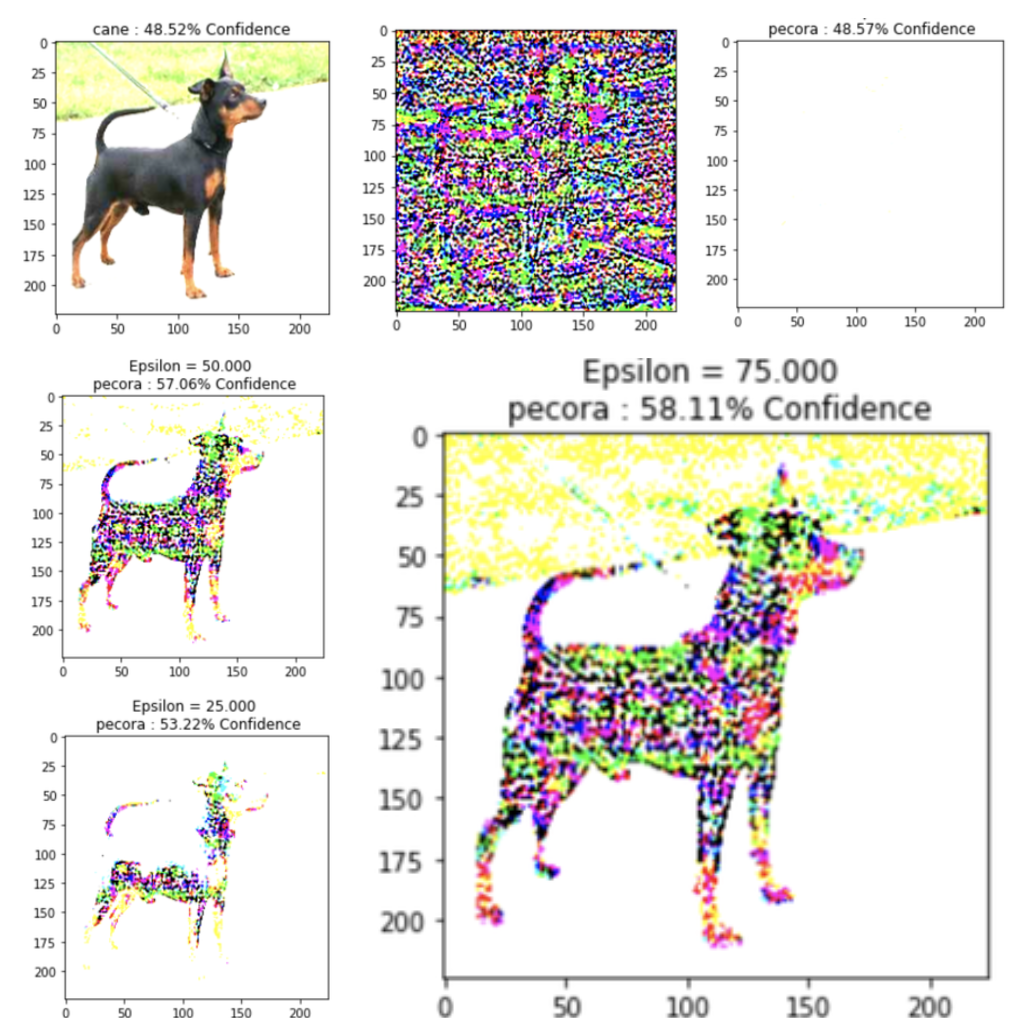
**Part (B):**

----------------------------------------------------------------------------------------------------------------

**Theory:** External Library chosen: CleverHans. I have performed this question in different methodologies.

**Approach 1: Only FGSM:** This is basic approach for FGSM attack available on TensorFlow documentation. Steps involved are follows:

1. Import and plot the image.
2. Perform prediction with the trained model from Part (A) on the imported image. In our case, it gave 48.52% confidence for “cane” on clean image.
3. Then create a perturbation vector / noise vector using create\_adversarial\_pattern() function. This is where the original logic of FGSM is performed. FGSM works on the principle of gradients. It tries to maximize the loss (negative to the gradient direction) and minimize the perturbation required such that the model misclassify the input example. This methodology calculates the important pixel which play key role in classification task that is how much each pixel is participating in giving loss or accuracy for that particular image. 
4. With varying epsilons, the prediction started misclassifying the label as “pecora” for different epsilon values.



**Approach 2:**

Since Cleverhans is one of the earlier libraries, it used to work great with python 2.x, So I retrained the model in python 2 environment. Versioning used in this question is:

Tensorflow Version: 2.1.0

Python Version: 2.2.1

Cleverhans Version: 3.0.1-9acbc88faf1d864c83e8d0ee0e71ba7f

('GPU Available: ', True)

However, the attribute error about graphkeys remains constant. Hence in this question I have just saved the model trained in tf 2.x and python 2.x in case in next process it is required.

'final\_model\_python2.h5'

So nothing useful related our task (which is to run cleverhans) was done in approach 2.

**Approach 3:**

Then I tried again with tf 1.x and python 2, but this also did not work. Finally, in approach 3 I tried with python 3.x and tf 2.x and cleverhans version which is available on github specifically, it worked.

Cleverhans Version: 3.0.1-15447acccf2628751c1e44ee30e141ec

The steps for approach 3 are as follows:

1. Import the dataset and convert to trainloader and testloader. Also import the model in the workflow.
2. Import necessary attacks from cleverhans, in this approach I have imported FGSM and PGD.
3. With the epsilon value of 0.1 the attack was not able to achieve misclassification. With the epsilon value of 0.7, it started misclassifying lot of examples. One of the example for eps=0.1 and eps=0.7 is shown below.

With epsilon=0.7 (Tested for 50 examples)

[..............................] - ETA: 34:23test acc on clean examples (%): 100.000

test acc on FGM adversarial examples (%): 0.000

test acc on PGD adversarial examples (%): 0.000

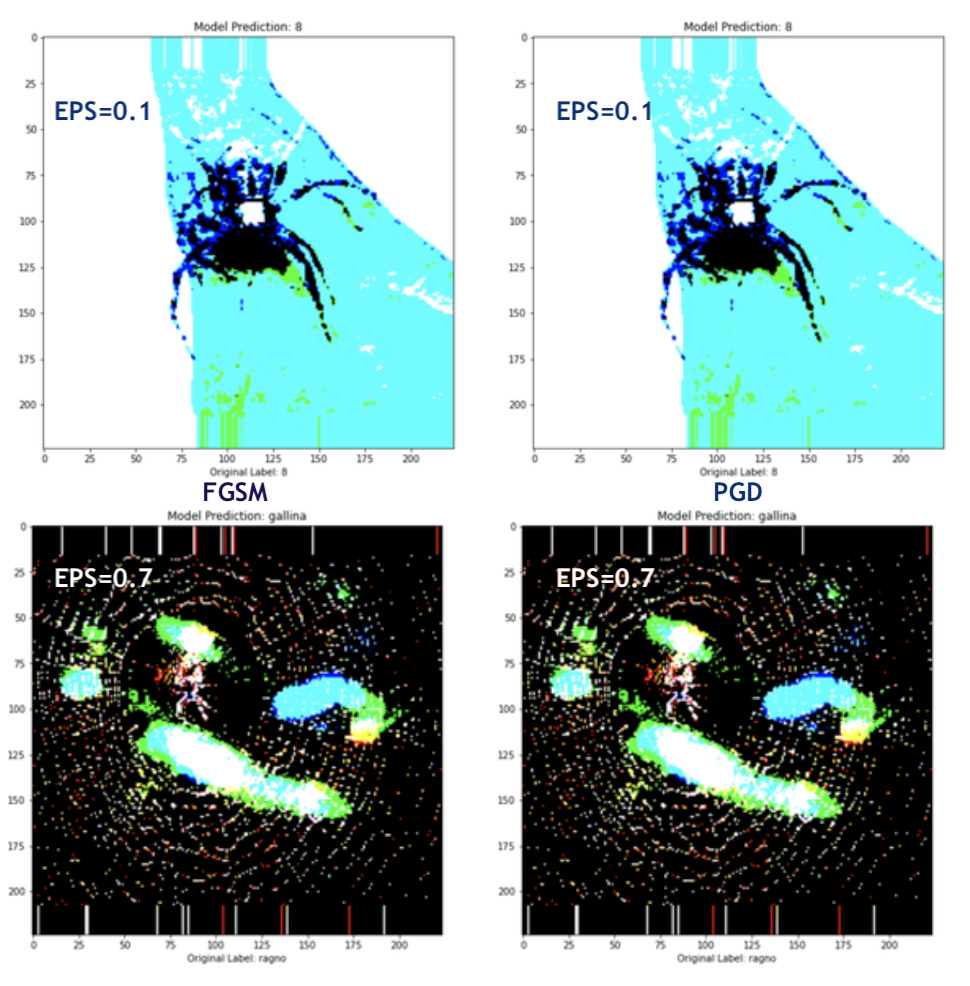
With epsilon=0.1 (Tested for 50 examples)

[..............................] - ETA: 57:47:56test acc on clean examples (%): 100.000

test acc on FGM adversarial examples (%): 90.000

test acc on PGD adversarial examples (%): 78.000

The higher test accuracy implies that the attack did not performed well. Thus in former case with e=0.7 we got 0 accuracy and latter case with e=0.1 we got higher test accuracy



**Approach 4:**

**FGSM/PGD/BIM/MIM:**

Continuing from approach 3 I have added one more attack in this approach. This approach total contains 3 attacks. Now that we have learned our sweet spot of epsilon=0.7 we will use it here with new random examples for all 3 attacks. The test results were: (for 79 eaxmples)

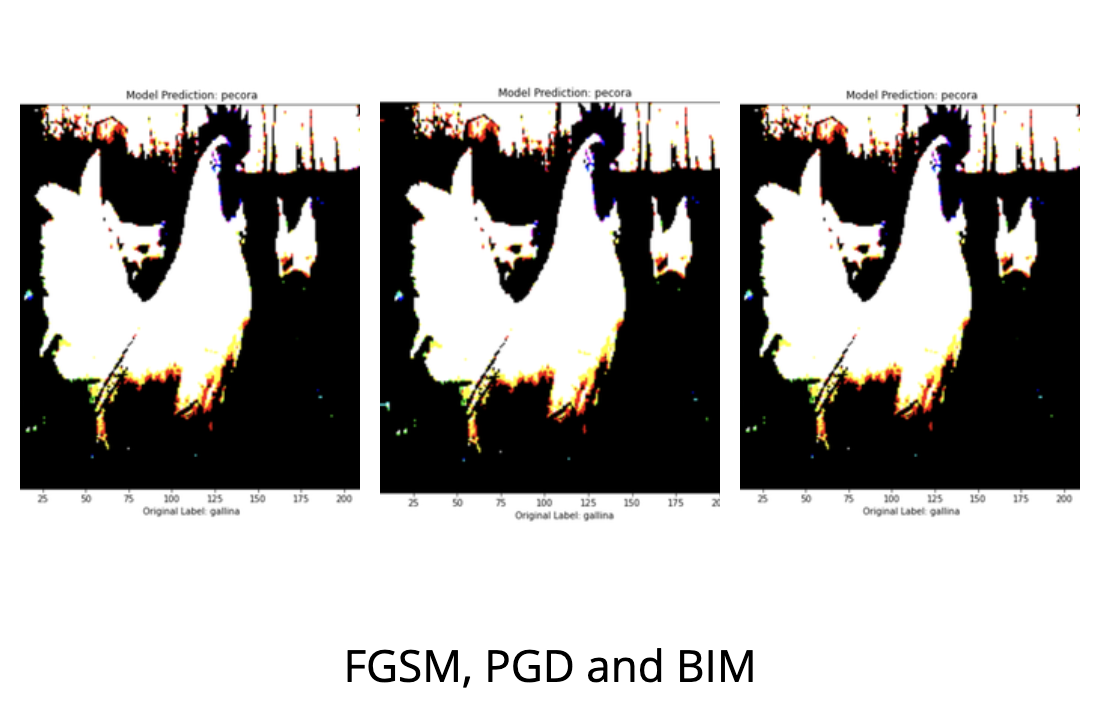
79/100 [..............................] - ETA: 1:05:53

test acc on clean examples (%): 98.642

test acc on FGM adversarial examples (%): 2.379

test acc on PGD adversarial examples (%): 5.394

test acc on BIM adversarial examples (%): 1.032

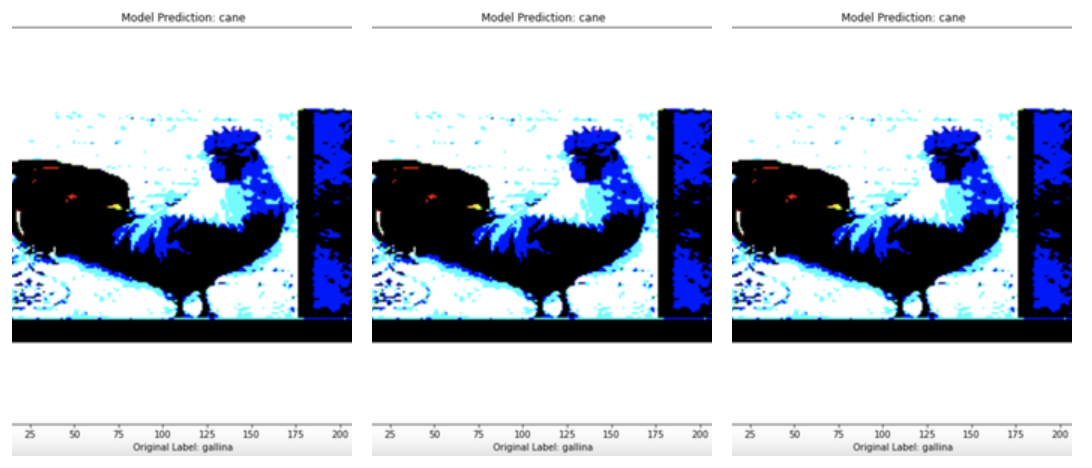


**Part (C):** **SSIM**

----------------------------------------------------------------------------------------------------------------

Continuing from previous part, that is part B, here I am going to report Structural Similarity in this part. (**Here I have taken only one example to look for SSIM**)The purpose of choosing only one sample from test set to calculate SSIM is because here I want to find how perturbations have affected. In other words, I want to calculate, how adversarial attack has changed my original image or clean image. The Similarity measure between 2 images is called SSIM. I have passed clean image and perturbed image to the SSIM function which then returns the value of similarity between both. The higher value returned implies both the images are similar and the perturbed image look similar in the perception However the lower SSIM value indicates that we have successfully achieved the attacking procedure and the images may or may not look perceptibly similar but it is perturbed in the pixel level. So in our example I have calculated SSIM for each of the attacks.

Example 1:



FGM SSIM: 0.4033331655939359 for FGM

PGD SSIM: 0.3631802415370079 for PGD

BIM SSIM: 0.3233245693048583 for BIM

1/100 [..............................] - ETA: 1:07:12

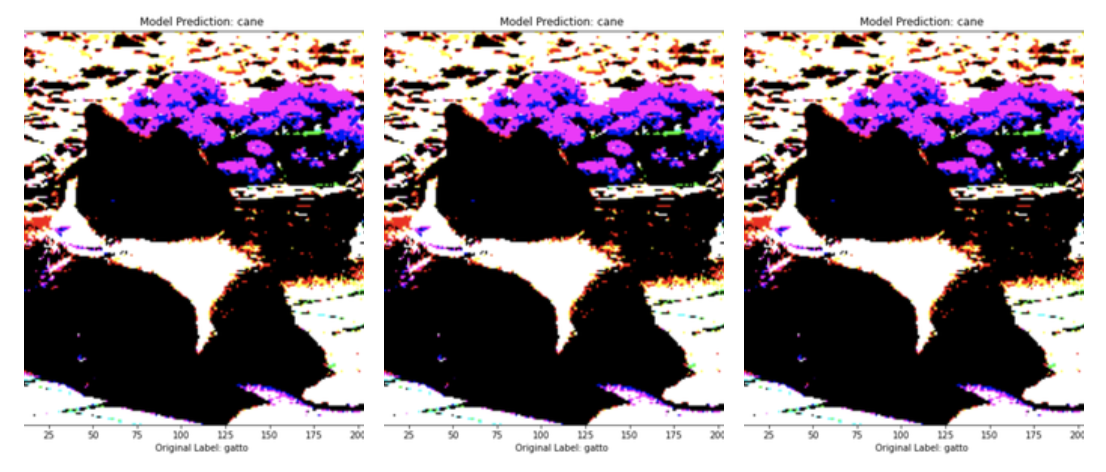
test acc on clean examples (%): 100.000

test acc on FGM adversarial examples (%): 0.000

test acc on PGD adversarial examples (%): 0.000

test acc on BIM adversarial examples (%): 0.000

Example 2:



FGM SSIM: 0.3028960843766728 for FGM

PGD SSIM: 0.3429613126004061 for PGD

MIM SSIM: 0.3354730333004546 for MIM

1/100 [..............................] - ETA: 1:07:34

test acc on clean examples (%): 0.000

test acc on FGM adversarial examples (%): 0.000

test acc on PGD adversarial examples (%): 0.000

test acc on MIM adversarial examples (%): 0.000

**Inferences**:

1. In example 1 you have seen that SSIM is approx. 30% to 40% for each attack. Remember we are performing this for only one example. And it is also expected that since it is attacked the classification is changed. The test accuracy on clean image is 100% which implies that our single clean sample is successfully correctly classified with the model, however the perturbed images are not thus giving 0% accuracy.
2. In example 2 you have seen that SSIM is approx. 30% to 35% for each attack. Here test accuracy for clean image is also 0 which implies even model is not correct on classifying the clean image.

**Part (D):** **Detection Measure (i)**

----------------------------------------------------------------------------------------------------------------

Detecting Adversarial Examples through Image Transformation Authors: Shixin Tian, Guolei Yang, Ying Cai, Paper Link - http://bit.ly/daipaper

Classification results of Clean images are immune to transformations.

Classification results of Adversarial images are prone to transformations.

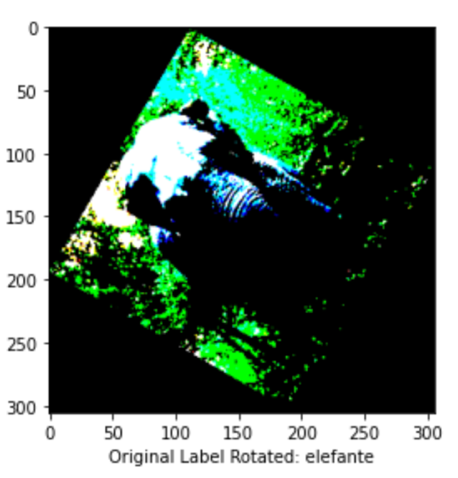
The goal is to build a detector that is able to distinguish adversarial examples from the normal ones. The image is a normal image and the classifier can classify it correctly. The image is an adversarial example and it can attack the classifier successfully. You can classify the image as adversarial or normal image as Classification results of Adversarial images are prone to transformations. Transforming adversarial examples with small rotations and shifts may get you correct classifications as well. You do not always get this correct classification on rotations and transformations, in fact there is debate on contrary that rotating the adversarial images leads to more problems but were not considering the SOTA CW attack. Author also mentioned this detection mechanism works properly when there are different patterns between normal images and adversarial examples.

------------FINAL RESULTS DETECTION-------------------

Without Rotation Original Label: elefante

Without Rotation Predicted Label: elefante

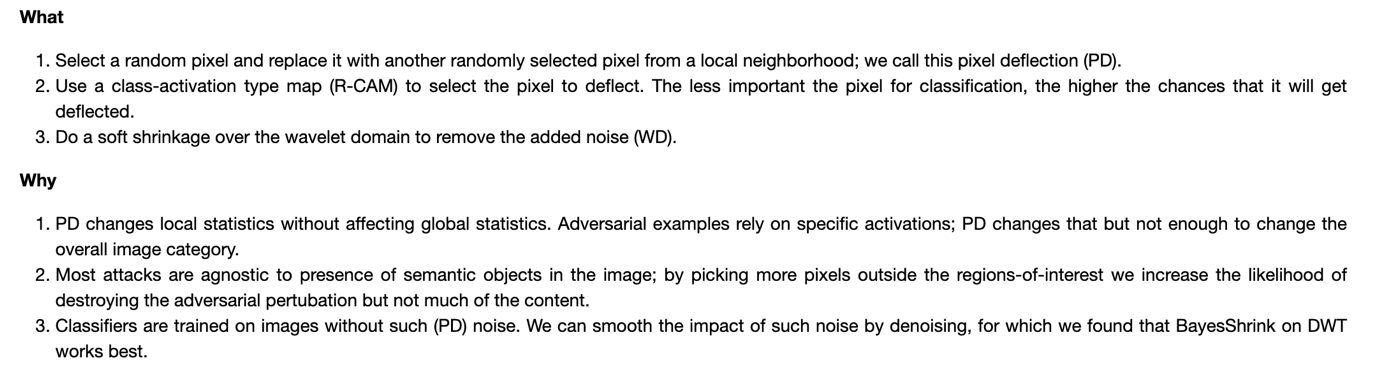
With Rotation Predicted Label: ragno

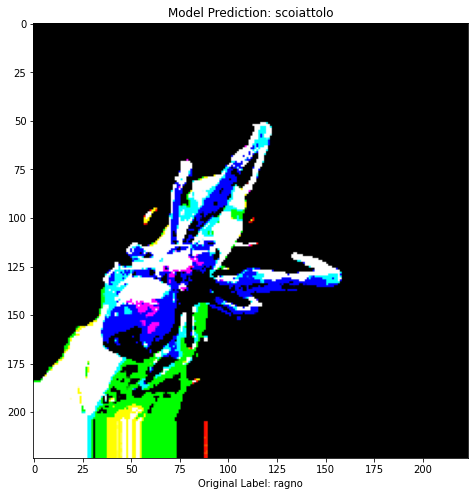


**Part (D):** **Detection Measure (ii)**

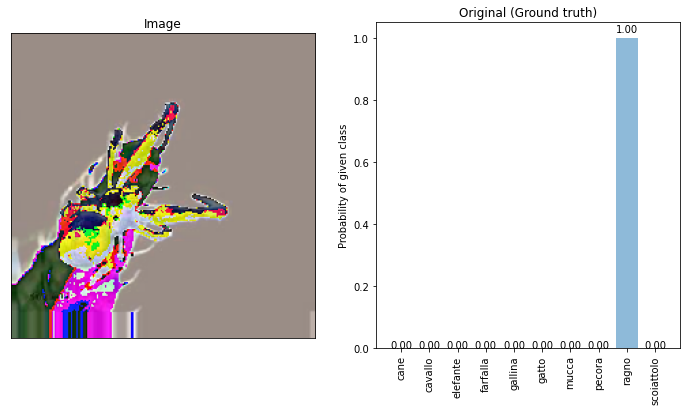
----------------------------------------------------------------------------------------------------------------

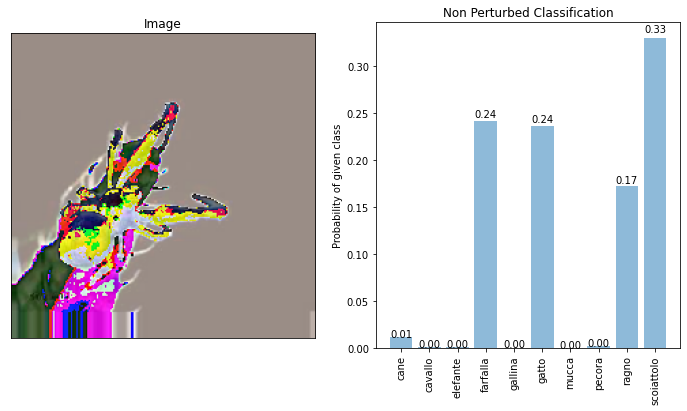
Pixel Deflection ref: <https://github.com/iamaaditya/pixel-deflection/blob/master/demo.ipynb>

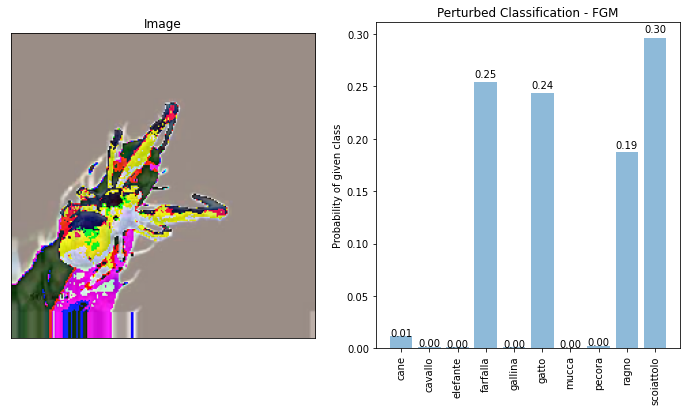




The concept is, try to replace multiple pixels selected randomly from the less important parts of the pixel distribution which takes less participation in the classification criteria. (This can be understood by activation maps) A randomly less important pixel is selected and perturbed. The expected behavior is that adding noise to less important pixels in the less important region of the image will bring heat map / activation map from more important feature of the image to that perturbed pixel region making it to misbehave for the classification algorithm. Below I have shown three charts, How the clean label, model prediction perturbed image prediction looks like. Checking last two graphs we get to know that performing FGM attack, it changes the prediction of the model probability values. By this we can check that something has changed in our image otherwise both the bottom two graphs should have looked same.







**Part (E – OnePixel Attack)** **Approach 1:**

Since I have already computed major attacks, I have tried one-pixel attack. It is fast, not resource hungry. Major drawback of this attack is that it is of the least effective attack In approach 1, I have just calculated for a small amount of dataset, how much of attacks are getting successful and attacking accuracy. In this approach I have used random 3144 examples.

One random example: (Failed to attack)

True Label: scoiattolo

True Image Predictions Label/Class: pecora

Perturbed Image Predictions Label/Class: pecora

Number of Times attack was successful: 12

Attack accuracy: 0.38167938931297707

**Part (E)** **Approach 2:**

I have tried to add more pixels such that it increases the attacking power. To make it multi-pixel attack.

image\_id = s\_number # Image index in the test set

pixel1 = np.array([64, 64, 255, 255, 0]) # pixel = x,y,r,g,b

pixel2 = np.array([64, 64, 0, 255, 255]) # pixel = x,y,r,g,b

pixel3 = np.array([124, 124, 225, 0, 255]) # pixel = x,y,r,g,b

pixel4 = np.array([64, 64, 255, 255, 0]) # pixel = x,y,r,g,b

pixel5 = np.array([34, 34, 255, 255, 0]) # pixel = x,y,r,g,b

pixel6 = np.array([34, 34, 255, 255, 0]) # pixel = x,y,r,g,b

pixel7 = np.array([34, 34, 255, 255, 0]) # pixel = x,y,r,g,b

pixel8 = np.array([34, 34, 255, 255, 0]) # pixel = x,y,r,g,b

image\_perturbed = perturb\_image(pixel1, test\_loader[image\_id][0][0])[0]

image\_perturbed = perturb\_image(pixel2, image\_perturbed)[0]

image\_perturbed = perturb\_image(pixel3, image\_perturbed)[0]

image\_perturbed = perturb\_image(pixel4, image\_perturbed)[0]

image\_perturbed = perturb\_image(pixel5, image\_perturbed)[0]

image\_perturbed = perturb\_image(pixel6, image\_perturbed)[0]

image\_perturbed = perturb\_image(pixel7, image\_perturbed)[0]

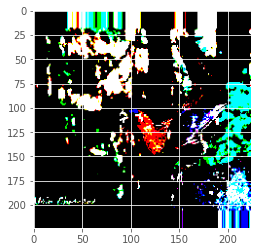
image\_perturbed = perturb\_image(pixel8, image\_perturbed)[0]

Example 1:

True Label: gallina

True Image Predictions Label/Class: scoiattolo

Perturbed Image Predictions Label/Class: ragno



Example 2:

In example 1 model clean image prediction and perturbed image prediction both are different with original label. In example 2 model clean image prediction and ground truth prediction are same however the perturbed image prediction is different. This means that the attack was successful. In this approach now I have started getting better attack performance I have tried with 10477 examples. And this was the resultant output of the code:

Number of Times attack was successful: 1252

Attack accuracy: 11.9499856829245



**Complete Solution for Part (E): look for file : *[12]DAI\_ASSIGNMENT\_2\_Part\_E(ii).ipynb***

For simplicity and speed up, I have taken a total of 1000 samples from different classes randomly. Then I have perturbed them. Splitting them with a shape of

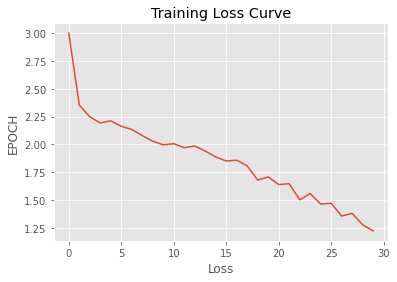
Xtrain.shape (1000, 224, 224, 3)

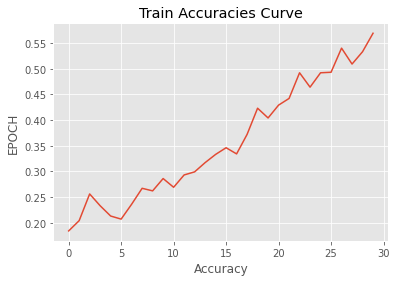
Ytrain.shape (1000, 10)

Xtest.shape (200, 224, 224, 3)

Ytest.shape (200, 10)

Now performing adversarial training.





Testting randomly for 7 examples.

7/7 [==============================] - 2s 56ms/step - loss: 2.1939 - accuracy: 0.4250

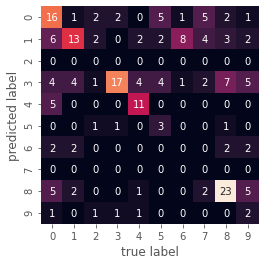
test loss:

2.1939125061035156

test accu:

0.42500001192092896

Confusion Matrix:



Classification report:

precision recall f1-score support

cane 0.46 0.41 0.43 39

cavallo 0.31 0.59 0.41 22

elefante 0.00 0.00 0.00 7

farfalla 0.35 0.81 0.49 21

gallina 0.69 0.58 0.63 19

gatto 0.50 0.21 0.30 14

mucca 0.00 0.00 0.00 10

pecora 0.00 0.00 0.00 13

ragno 0.61 0.61 0.61 38

scoiattolo 0.33 0.12 0.17 17

accuracy 0.42 200

macro avg 0.32 0.33 0.30 200

weighted avg 0.40 0.42 0.39 200

**Part (F): Mitigation and JPEG Compression**

Now that we have two models, model\_new and model\_old as:

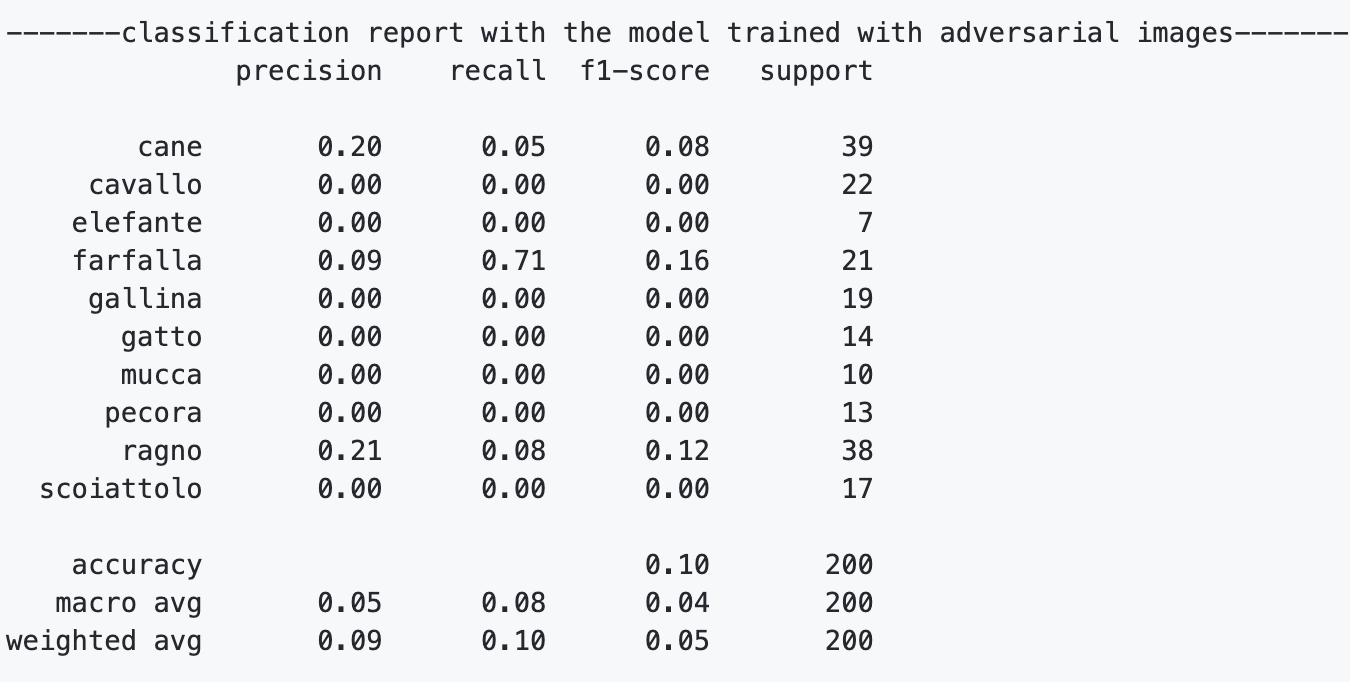
model\_new = model # Model with adversarial training

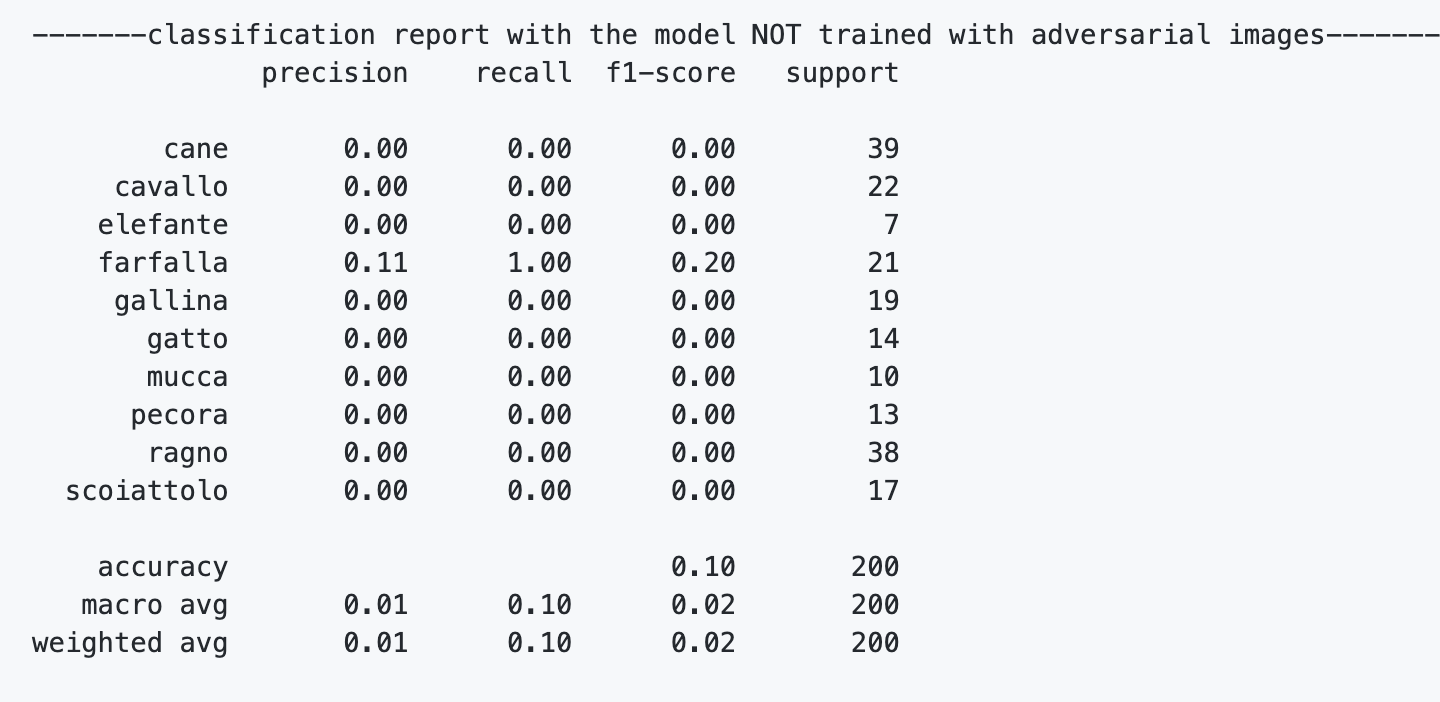
model\_old = keras.models.load\_model('final\_model.h5') # My Saved model (Trained with Python 3 & TF 2.X) # model without adversarial training

Lets do classification metrics using moth models on JPEG compression and see its effect.

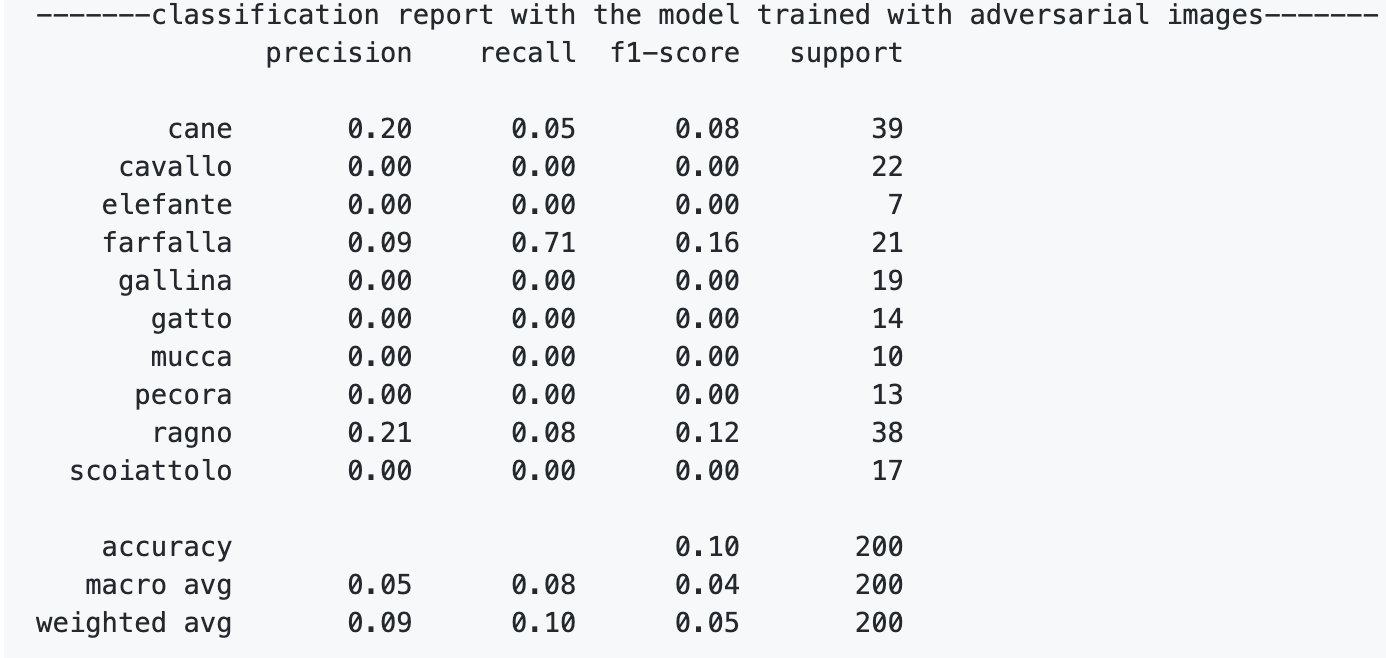
(Performed for 200 examples)

**Metrics for JPEG Compression at 50%**





**Metrics for JPEG Compression at 10%**





**References:**

**1. https://colab.research.google.com/github/shreehari117/ML-2/blob/master/Sureshbabu\_Shree\_Hari\_aml.ipynb**

**2. https://colab.research.google.com/github/andantillon/cleverhans/blob/master/tutorials/future/tf2/notebook\_tutorials/mnist\_fgsm\_tutorial.ipynb**

**3. https://github.com/Fuu3214/Grad\_Proj/blob/4adc3147fdaa5a0acfca61c67188608b07880048/attack\_and\_generate\_attr.ipynb**

**4. https://github.com/Carco-git/CW\_Attack\_on\_MNIST/blob/master/CW\_Attack\_l2.ipynb**

**5.** [**https://github.com/ocatak-zz/adversarial-ml-training/blob/5e6984a28e9375fb40362b17ffd931f0b8da2e70/adversarial-machine-learning-attacks-and-mitigations.ipynb**](https://github.com/ocatak-zz/adversarial-ml-training/blob/5e6984a28e9375fb40362b17ffd931f0b8da2e70/adversarial-machine-learning-attacks-and-mitigations.ipynb)

**6.** [**https://www.tensorflow.org/tutorials/generative/adversarial\_fgsm**](https://www.tensorflow.org/tutorials/generative/adversarial_fgsm)

**7.** [**http://media.nips.cc/nipsbooks/nipspapers/paper\_files/nips32/reviews/1853.html**](http://media.nips.cc/nipsbooks/nipspapers/paper_files/nips32/reviews/1853.html)

**8.** [**https://ourcodeworld.com/articles/read/991/how-to-calculate-the-structural-similarity-index-ssim-between-two-images-with-python**](https://ourcodeworld.com/articles/read/991/how-to-calculate-the-structural-similarity-index-ssim-between-two-images-with-python)

**9.** [**https://github.com/iamaaditya/pixel-deflection/blob/master/demo.ipynb**](https://github.com/iamaaditya/pixel-deflection/blob/master/demo.ipynb)

**10.** [**https://stackoverflow.com/questions/30771652/how-to-perform-jpeg-compression-in-python-without-writing-reading**](https://stackoverflow.com/questions/30771652/how-to-perform-jpeg-compression-in-python-without-writing-reading)

**Structure of the Deliverables:**

The Uploaded zip contains:

[0][First\_training]DAI\_ASSIGNMENT\_2\_Part\_A.ipynb

[1]DAI\_ASSIGNMENT\_2\_Part\_B\_FGSM\_.ipynb

[2]DAI\_ASSIGNMENT\_2\_Part\_B\_FGSM.ipynb

[3]DAI\_ASSIGNMENT\_2\_Part\_B.ipynb

[4]DAI\_ASSIGNMENT\_2\_Part\_B\_FGSM\_PGD.ipynb

[5]DAI\_ASSIGNMENT\_2\_Part\_B\_FGSM\_PGD\_BIM.ipynb

[6]DAI\_ASSIGNMENT\_2\_Part\_C\_FGSM\_PGD\_BIM\_MIM\_SSIM.ipynb

[7]DAI\_ASSIGNMENT\_2\_Part\_D\_Detection\_Measure\_1.ipynb

[8]DAI\_ASSIGNMENT\_2\_Part\_D\_Detection\_Measure\_2.ipynb

[9]DAI\_ASSIGNMENT\_2\_Part\_E\_OPA\_ATTACK.ipynb

[10]DAI\_ASSIGNMENT\_2\_Part\_E\_OPA\_ATTACK\_AND\_TRAINING.ipynb

[11]DAI\_ASSIGNMENT\_2\_Part\_E(i).ipynb

[12]DAI\_ASSIGNMENT\_2\_Part\_E(ii).ipynb

[13]DAI\_ASSIGNMENT\_2\_Part\_F.ipynb

