CS 725 : Project Presentation

Topic : Adversarial Attacks and Defense using Fast Sign Gradient Method and Black Box Technique

Team Members:

1. Varad Bhatnagar (19305R007)
2. Prince Kumar (193050070)

Motivation

- Neural Networks are being increasingly used to automate detection and recognition tasks in a variety of fields such as Supply Chain Management, Law Enforcement, Camera Software and Biometrics.

- In the recent years, it has come to light that well trained and highly accurate Neural Networks are failing in some specific situations. Not only failing, they are often found taking a wrong decision with high confidence. Such data points are called ‘Adversarial Examples’.

- Such a failure can not be tolerated in some sensitive fields such as Law Enforcement and Self Driving Cars.

- Most often, human’s can’t distinguish between a Normal example and an Adversarial Example because the difference is very minute to the naked eye.

Goal

The goal of this project is to demonstrate :

(i) A well trained and highly accurate neural network fails on adversarial samples.

(ii) A neural network which has been trained on adversarial data, performs well on new adversarial samples.

(iii) How to generate adversarial samples :
- (a) When attacker knows the network architecture.
- (b) When the attacked doesn’t know anything about the network architecture.

(iv) Working of Fast Sign Gradient Method and Black Box technique.

Literature Survey

We studied the following papers before implementing the project:

1. Explaining and Harnessing Adversarial Examples by Ian J. Goodfellow, Jonathon Shlens, Christian Szegedy

Key Idea : Existence of Adversarial Examples can be explained by Linear behaviour in high dimensional spaces and is not because of extreme non-linearity of Deep Neural Nets.

2. Practical Black-Box Attacks against Machine Learning by Nicolas Papernot, Patrick McDaniel, Ian Goodfellow, Somesh Jha, Z. Berkay Celik, Ananthram Swami

Key Idea : Accessing labels O^* produced by the Deep Neural Net O is the only capability assumed in the threat model.

Approach

When model architecture is known to attacker:

1. Train a model using normal training dataset. We will call this the ‘Clean Model’.
2. Test the ‘Clean Model’ on a test dataset to ascertain accuracy. Repeat task 1 with adjusted hyperparameters if the accuracy is not found to be sufficient.
3. Test the ‘Clean Model’ on adversarial examples generated by Fast Sign Gradient Method.

4. Train a model using a mixture of normal and adversarial training dataset. We will call this the ‘Adversarial Model’.
5. Test the ‘Adversarial Model’ on a test dataset to ascertain accuracy.
6. Test the ‘Adversarial Model’ on adversarial examples generated by Fast Sign Gradient Method.

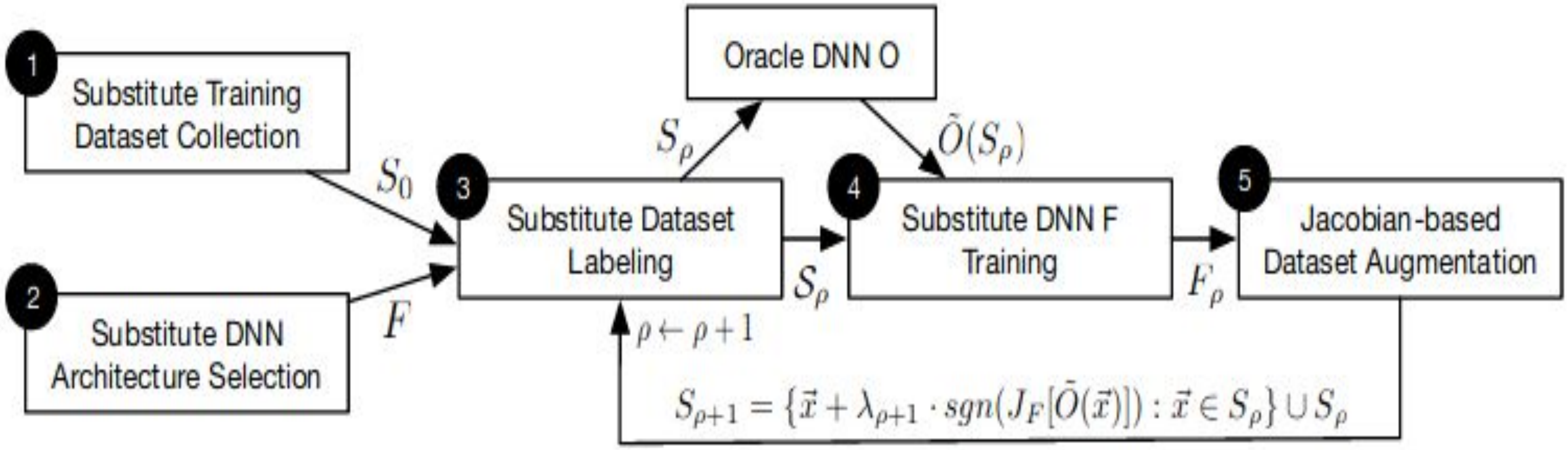
7. Report all the results to draw a conclusion.

Approach

When model architecture is not known to attacker:

1. Initial Collection : Collect a very small set S0 of inputs representative of the input domain.
2. Architecture Selection : Select an architecture to be trained as the substitute F. Again, this can be done using high-level knowledge of the classification task performed by the oracle.
3. Substitute Training : Iteratively train more accurate substitute DNNs F_ρ by repeating the following for $\rho \in 0..\rho_{max}$
4. Labeling : By querying for the labels $O(\tilde{x})$ output by oracle O, label each sample $\tilde{x} \in S_\rho$ in its initial substitute training set S_ρ.
5. Training : Train the architecture chosen at step (2) using substitute training set Sρ in conjunction with classical training techniques.
6. Augmentation and Reporting

Black Box Attack Flow


1
Substitute Training Dataset Collection
2
Substitute DNN Architecture Selection
S_0
F
3
Substitute Dataset Labeling
Oracle DNN O
S_ρ
$\tilde{O}(S_\rho)$
S_ρ
4
Substitute DNN F Training
F_ρ
5
Jacobian-based Dataset Augmentation
$\rho \leftarrow \rho + 1$
$S_{\rho+1} = \{\vec{x} + \lambda_{\rho+1} \cdot sgn(J_F[\tilde{O}(\vec{x})]) : \vec{x} \in S_\rho\} \cup S_\rho$

Datsets Used

We have used the following datasets:

1. MNIST – This is a well known dataset of images of handwritten digits from 0 to 9, number of output classes being 10. All images are of size 28*28 pixels and grayscale i.e. pixels can take value between 0 and 255. There are 70,000 samples in this dataset.

2. EMNIST – The extended version of MNIST. This dataset contains images of handwritten digits and characters from a-z, A-Z, 0-9. The number of output classes are 47 as some classes for some characters which have similar lowercase and uppercase representations (j, i, o, z, etc) have been combined. All images are of size 28*28 pixels and grayscale i.e. pixels can take value between 0 and 255. There are more than 240,000 samples in this dataset.

Architecture and Specifications

Model : Convolutional Neural Network
Source: Cleverhans Model Zoo

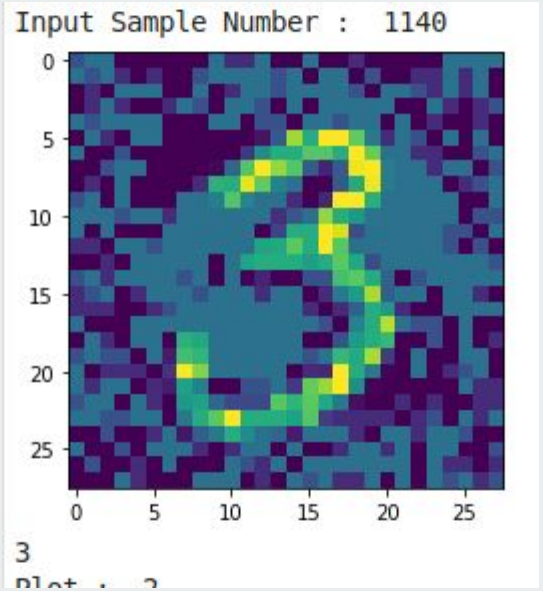
Initially we were using our own laptops to train the model but it was taking a lot of time as the computations were taking a lot of time due to RAM limitations. Then we shifted to Google Colab on which we used a free instance with 12 GB of RAM and a GPU. Training speed grew manifold after this.

Language : Python

Platform : Google Colab

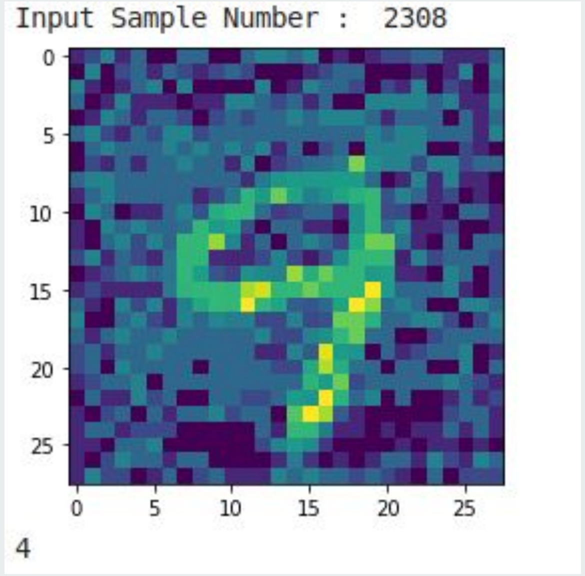
Libraries : Numpy, Tensorflow, Cleverhans, Matplotlib, functools, logging

Results



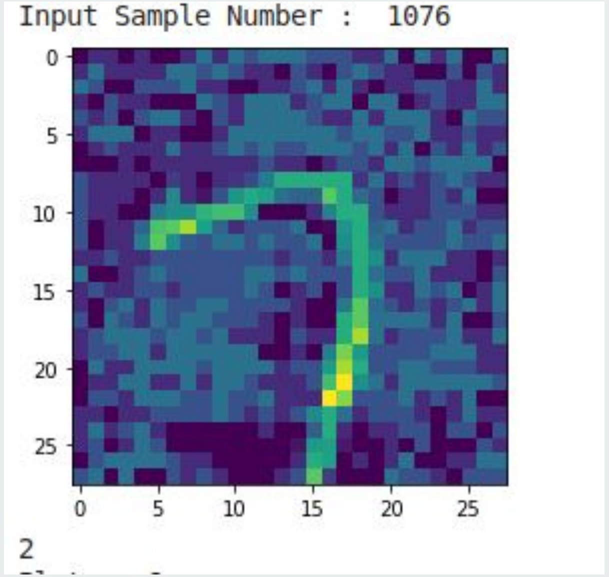
```
Input Sample Number :  1140
```

```
3
Plot : 2
```



```
Input Sample Number :  2308
```

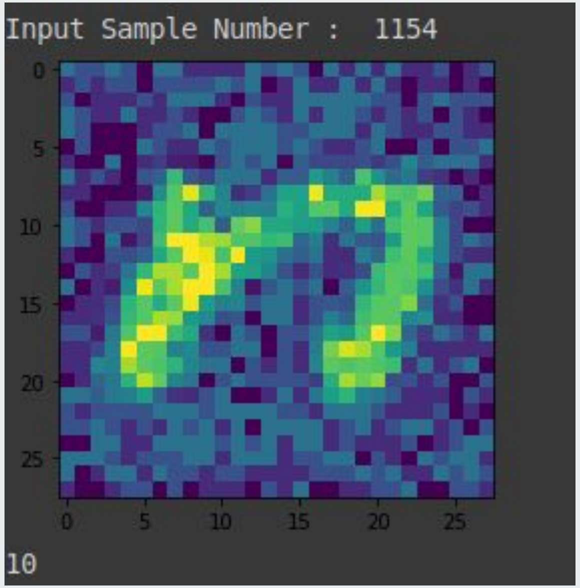
```
4
```



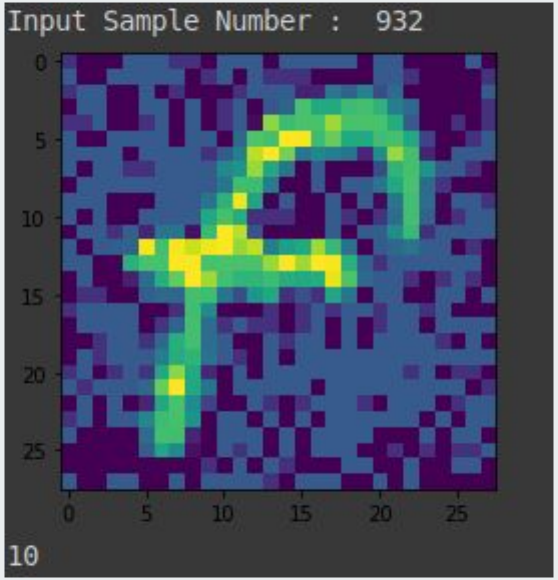
```
Input Sample Number :  1076
```

```
2
```

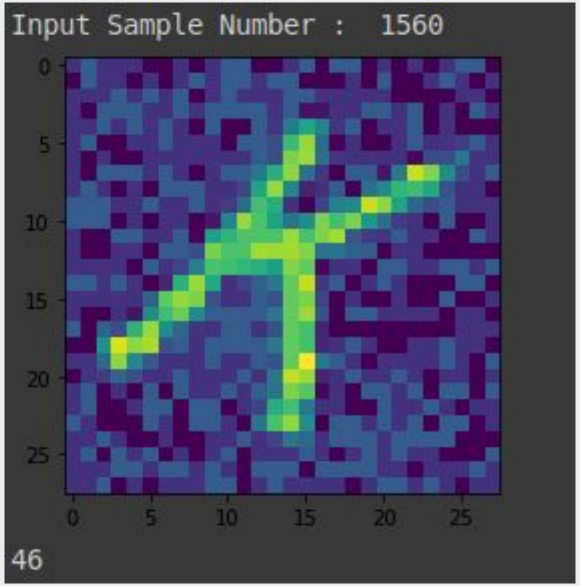
Results


Input Sample Number : 1154

10

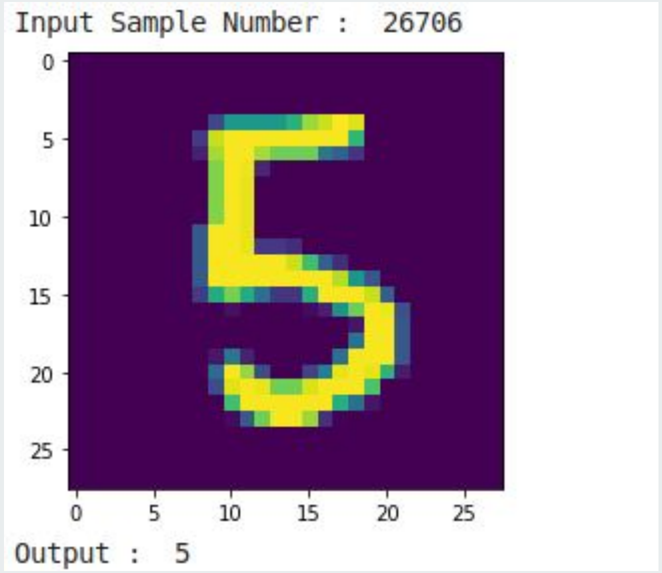

Input Sample Number : 932

10


Input Sample Number : 1560

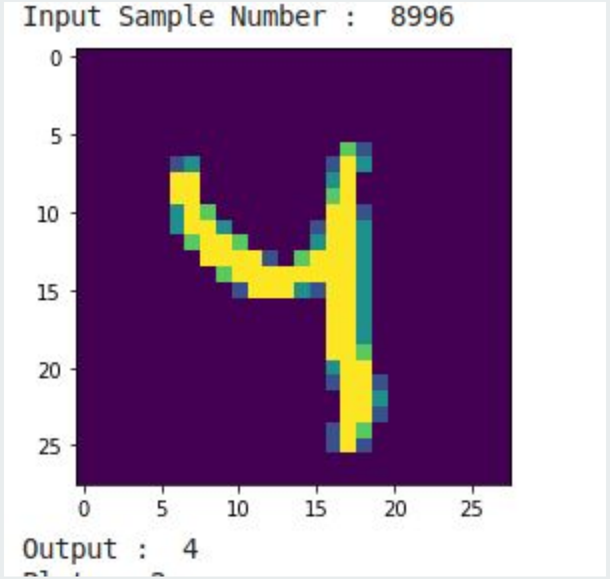
46

Results



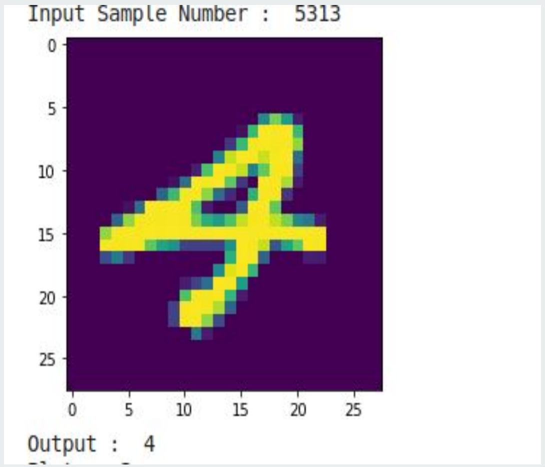
```
Input Sample Number :  26706
```

```
Output :  5
```



```
Input Sample Number :  8996
```

```
Output :  4
```

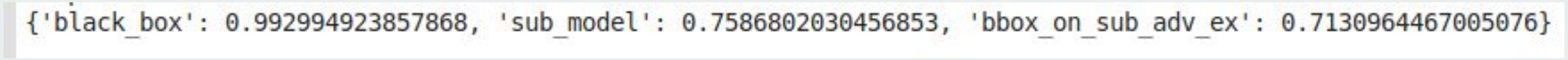


```
Input Sample Number :  5313
```

```
Output :  4
```

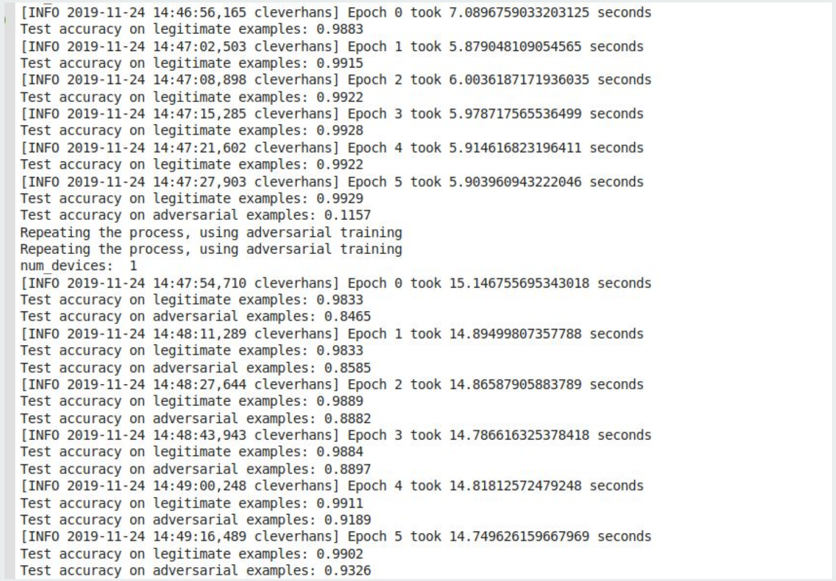
Results

BB Accuracy



```
{'black_box': 0.992994923857868, 'sub_model': 0.7586802030456853, 'bbox_on_sub_adv_ex': 0.7130964467005076}
```

FSGM Accuracy



```
[INFO 2019-11-24 14:46:56,165 cleverhans] Epoch 0 took 7.0896759033203125 seconds
Test accuracy on legitimate examples: 0.9883
[INFO 2019-11-24 14:47:02,503 cleverhans] Epoch 1 took 5.879048109054565 seconds
Test accuracy on legitimate examples: 0.9915
[INFO 2019-11-24 14:47:08,898 cleverhans] Epoch 2 took 6.0036187171936035 seconds
Test accuracy on legitimate examples: 0.9922
[INFO 2019-11-24 14:47:15,285 cleverhans] Epoch 3 took 5.978717565536499 seconds
Test accuracy on legitimate examples: 0.9928
[INFO 2019-11-24 14:47:21,602 cleverhans] Epoch 4 took 5.914616823196411 seconds
Test accuracy on legitimate examples: 0.9922
[INFO 2019-11-24 14:47:27,903 cleverhans] Epoch 5 took 5.903960943222046 seconds
Test accuracy on legitimate examples: 0.9929
Test accuracy on adversarial examples: 0.1157
Repeating the process, using adversarial training
Repeating the process, using adversarial training
num_devices:  1
[INFO 2019-11-24 14:47:54,710 cleverhans] Epoch 0 took 15.146755695343018 seconds
Test accuracy on legitimate examples: 0.9833
Test accuracy on adversarial examples: 0.8465
[INFO 2019-11-24 14:48:11,289 cleverhans] Epoch 1 took 14.89499807357788 seconds
Test accuracy on legitimate examples: 0.9833
Test accuracy on adversarial examples: 0.8585
[INFO 2019-11-24 14:48:27,644 cleverhans] Epoch 2 took 14.86587905883789 seconds
Test accuracy on legitimate examples: 0.9889
Test accuracy on adversarial examples: 0.8882
[INFO 2019-11-24 14:48:43,943 cleverhans] Epoch 3 took 14.786616325378418 seconds
Test accuracy on legitimate examples: 0.9884
Test accuracy on adversarial examples: 0.8897
[INFO 2019-11-24 14:49:00,248 cleverhans] Epoch 4 took 14.81812572479248 seconds
Test accuracy on legitimate examples: 0.9911
Test accuracy on adversarial examples: 0.9189
[INFO 2019-11-24 14:49:16,489 cleverhans] Epoch 5 took 14.749626159667969 seconds
Test accuracy on legitimate examples: 0.9902
Test accuracy on adversarial examples: 0.9326
```

Code

Final code can be viewed at :
https://github.com/varadhbhatnagar/Adversarial-CNN-using-FSGM-and-BlackBox

There are two files in total:
(i) BlackBox.py – This file contains the implementation of the Black Box technique.
(ii) FSGM.py – This file contains the implementation of the FSGM Technique.

Google Colab was used to train the CNN. Link to colab environment : [Prince Sir]

As we were using python libraries such as Numpy and Cleverhans, many functions were already implemented and we just had to use the API calls.

Total lines of Submitted Code ~ 500.
Total lines of Code (Including Unsuccessful Appoach) ~ 800

Conclusion

We explored the theory behind existence of adversarial examples and their generation using the Fast Sign Gradient Method and Black Box Techniques successfully. These two methods correspond to the two scenarios when :

(i) The attacker knows about the underlying architecture of the Neural Net.
(ii) The attacker doesn’t know about the underlying architecture of the Neural Net.

We also demonstrated the impact of such adversarial examples on well trained and highly accurate Neural Network and how easily they get fooled. Not only do they predict the wrong class, they do so with a high degree of confidence.

References

Blogs:
https://www.kaggle.com/allunia/how-to-attack-a-machine-learning-model
https://www.kaggle.com/benhamner/adversarial-learning-challenges-getting-started

Datasets:
https://www.nist.gov/node/1298471/emnist-dataset
http://yann.lecun.com/exdb/mnist/

Libraries:
https://github.com/tensorflow/cleverhans
https://github.com/bethgelab/foolbox
https://adversarial-robustness-toolbox.readthedocs.io/en/latest/

References

Papers:

‘Explaining and Harnessing Adversarial Examples’ by Ian J. Goodfellow, Jonathon Shlens, Christian Szegedy (https://arxiv.org/abs/1511.07528)

‘Practical Black-Box Attacks against Machine Learning’ by Nicolas Papernot, Patrick McDaniel, Ian Goodfellow, Somesh Jha, Z. Berkay Celik, Ananthram Swami (https://arxiv.org/abs/1412.6572)

Code Base:
http://www.robots.ox.ac.uk/~aarnab/adversarial_robustness.html
https://github.com/hmph/adversarial-attacks
https://github.com/asranand7/Adverserial_Attack