CS6220 HW2
Problem 1
Option 1

Name: Junyan Mao
GTID: 903343678

Environment:

- Windows 10 Pro 19043
- CPU: Intel® Core™ i7-9700k CPU @ 3.60GHz 3600 Mhz, 8 Core(s)
- RAM: 32.0GB
- GPU: NVIDIA GeForce RTX 2080

Code for this homework: https://github.com/jmao44/GTAttackPod

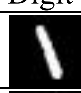
- I forked the GTAttackPod Repo, cloned it locally, and made modifications
- Added/Modified files:
    - datasets/datasets_utils.py
        - added time measurement for recording average time per example of the models under NO attack
    - attacks/lts4/deepfool.py
        - added code where I outputted the original image, the intermediate image within the loop, and the result image of the DeepFool attack. This is later commented out.
    - example_MNIST/
        - This folder contains 30 images of 10 examples during 3 stages of the DeepFool attack on the MNIST dataset (beginning, 2nd iteration, result). The model being attacked is CNN-7
    - example_CIFAR10/
        - This folder contains 30 images of 10 examples during 3 stages of the DeepFool attack on the CIFAR10 dataset (beginning, 1st iteration, result). The model being attacked is DenseNet-40
    - mnist_cnn_jmao44/MNIST_CNN_jmao44.ipynb
        - This is the iPython Notebook where I trained my own version of CNN on the MNIST dataset. CNN structure adopted from: https://github.com/yashk2810/MNIST-Keras
    - models/__init__.py
        - modified import statement so that the new model can be properly imported
    - models/MNIST_jmao44.py
        - Set up model, load weights, and compile model
    - models/weights/MNIST_jmao44.keras_weights.h5
        - Weights for CNN_jmao on the MNIST dataset
    - models/weights/CIFAR10_ResNet20v2.keras_weights.h5
        - Weights for ResNet-20 on the CIFAR-10 dataset
    - models/weights/CIFAR10_ResNet110v2.keras_weights.h5
        - Weights for ResNet-110 on the CIFAR-10 dataset
    - cifar10_resnet_jmao44/CIFAR10_Resnet_jmao44.ipynb

- iPython notebook where I trained two versions of ResNet (ResNet-20 and ResNet-110) on the CIFAR-10 dataset.
- attack_scripts/DeepFool-UA_CIFAR10_ResNet20.py
  - Script to attack ResNet-20 with DeepFool
- attack_scripts/DeepFool-UA_CIFAR10_ResNet110.py
  - Script to attack ResNet-110 with DeepFool
- attack_scripts/Transferability.py
  - Script to use the adversarial examples generated from attacking DenseNet-40 to attack ResNet-20

Input analysis (1): Provide a summary of your pre-trained models and datasets. For each dataset, provide 10 example inputs under five different classes, 2 per class.

Model: CNN-7
CNN refers to convolutional neural network, which is a type of artificial neural network and has a wide range of applications in computer vision tasks. There are several essential components to a CNN: convolutional layers to extract features; pooling layers to do down-sampling; ReLU layers to serve as activation functions; fully connected layers to help combine features and make everything into a model; and finally, Softmax function to produce the classification output. CNN-7 is simply a 7-layer (convolutional + dense layer) setup of such structure. State-of-the-art CNN models are able to achieve over 99% accuracy on the MNIST dataset.

Dataset: MNIST database (dataset)
MNIST stands for Modified National Institute of Standards and Technology. The MNIST database is a huge database of handwritten digits which is commonly used for training and testing number recognition machine learning models. The MNIST database consists of 60,000 training examples and 10,000 testing examples.

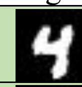
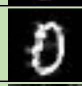
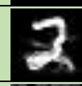
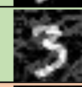
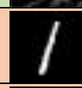
	Digit 0	Digit 1	Digit 2	Digit 3	Digit 4
Example 1					
Example 2					

Input analysis (2): Provide a summary of the two attack algorithm of your choice.

DeepFool is a simple and accurate method to fool deep neural networks, by efficiently computing perturbations. According to the authors of DeepFool, it is based on an iterative linearization of the classifier to generate minimal perturbations that are sufficient to change classification labels. DeepFool also tends to generate smaller perturbations than other methods, which makes it a valuable tool to estimate the robustness of classifiers.

PGD stands for Projected Gradient Descent. It's categorized as a "white-box" attack because the gradients of the model are exposed to attackers. PGD attempts to find the perturbation that maximizes the loss of a model on an input image, while keep the perturbation size under a specified threshold, called epsilon.

Input analysis (3): Provide the attack examples you generated for the 10 examples you listed in 1).

	Digit 0	Digit 1	Digit 2	Digit 3	Digit 4
Example 1					
DeepFool					
PGD					
Example 2					
DeepFool					
PGD					

Analysis: From the above form, we can see that DeepFool is adding "less" perturbation to the image visually than PGD, which could mean that DeepFool is a more efficient attack algorithm and is able to misdirect the model with fewer human-perceivable changes to the images.

Output analysis (1):
Note: I have provided my answer to the questionnaire mentioned in requirement (2).

Output analysis (2)+(3)+(4):
I created a new CNN for this part of the assignment, we will be referencing it as "*CNN-jmao*" for convenience. Its structure is as follows:

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 26, 26, 32)        320
_________________________________________________________________
activation_1 (Activation)    (None, 26, 26, 32)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 24, 24, 32)        9248
_________________________________________________________________
activation_2 (Activation)    (None, 24, 24, 32)        0
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 12, 12, 32)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 10, 10, 64)        18496
_________________________________________________________________
activation_3 (Activation)    (None, 10, 10, 64)        0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 8, 8, 64)          36928
_________________________________________________________________
activation_4 (Activation)    (None, 8, 8, 64)          0
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 4, 4, 64)          0
_________________________________________________________________
flatten_1 (Flatten)          (None, 1024)              0
_________________________________________________________________
dense_1 (Dense)              (None, 512)               524800
_________________________________________________________________
activation_5 (Activation)    (None, 512)               0
_________________________________________________________________
dropout_1 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_2 (Dense)              (None, 10)                5130
_________________________________________________________________
activation_6 (Activation)    (None, 10)                0
=================================================================
Total params: 594,922
Trainable params: 594,922
Non-trainable params: 0
```

Here are the results of running different attacks on different models, on different datasets:

Dataset/Model/Attack	*Test Accuracy*	*Test Time Per Example/s*
MNIST/CNN-7/Benign	99.43%	0.00012614288330078125
MNIST/CNN-7/DeepFool	0% (misclassification 100%)	0.080392
MNIST/CNN-7/PGD	4% (misclassification 96%)	0.019253
MNIST/CNN-jmao/Benign	99.28%	0.00023227918148040772

MNIST/CNN-jmao/DeepFool	0% (misclassification 100%)	0.073408
MNIST/CNN-jmao/PGD	0% (misclassification 100%)	0.017287
CIFAR10/DenseNet-40/Benign	94.84%	0.021957121586799622
CIFAR10/DenseNet-40/DeepFool	0% (misclassification 100%)	1.656201
CIFAR10/DenseNet-40/PGD	9% (misclassification 91%)	0.797615

From the above form, it is obvious that DeepFool is a very strong attack algorithm, because the models attacked by it all achieved a test accuracy of 0%. PGD also has impressive performance. Both attack algorithms are causing the model to use more time to classify at test time, which means they are harming the image classification models' efficiency effectively.

Screenshot of running *DeepFool-UA_MNIST_CNN7.py*:

```
Loading the dataset...
Evaluating the target model...
Test accuracy on benign examples 99.43%
Mean confidence on ground truth classes 99.39%
Test time per example 0.00012614288330078125 seconds
Selected 100 examples.
Test accuracy on selected benign examples 100.00%
Mean confidence on ground truth classes, selected 100.00%

>> Compiling the gradient TensorFlow functions. This might take some time...
>> Computing gradient function...
  [>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>] DeepFool Attacking 100/100

---Statistics of DeepFool Attack (0.080392 seconds per sample)
Success rate: 100.00%, Misclassification rate: 100.00%, Mean confidence: 83.76%
Li dist: 0.5905, L2 dist: 2.2081, L0 dist: 48.1%
```

Screenshot of running *PGD-UA_MNIST_CNN7.py*:

```
Loading the dataset...
Evaluating the target model...
Test accuracy on benign examples 99.43%
Mean confidence on ground truth classes 99.39%
Test time per example 0.0001384028196334839 seconds
Selected 100 examples.
Test accuracy on selected benign examples 100.00%
Mean confidence on ground truth classes, selected 100.00%

  [>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>] PGD Attacking 100/100

---Statistics of PGD Attack (0.019253 seconds per sample)
Success rate: 96.00%, Misclassification rate: 96.00%, Mean confidence: 99.99%
Li dist: 0.3020, L2 dist: 5.4795, L0 dist: 73.7%
```

Screenshot of running *DeepFool-UA_MNIST_CNNjmao.py*:

```
Loading the dataset...
Evaluating the target model...
Test accuracy on benign examples 99.28%
Mean confidence on ground truth classes 99.06%
Test time per example 0.00021606359481811524 seconds
Selected 100 examples.
Test accuracy on selected benign examples 100.00%
Mean confidence on ground truth classes, selected 99.38%

>> Compiling the gradient TensorFlow functions. This might take some time...
>> Computing gradient function...
  [>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>] DeepFool Attacking 100/100

---Statistics of DeepFool Attack (0.074331 seconds per sample)
Success rate: 100.00%, Misclassification rate: 100.00%, Mean confidence: 81.16%
Li dist: 0.5381, L2 dist: 2.0100, L0 dist: 48.8%
```

Screenshot of running *PGD-UA_MNIST_CNNjmao.py*

```
Loading the dataset...
Evaluating the target model...
Test accuracy on benign examples 99.28%
Mean confidence on ground truth classes 99.06%
Test time per example 0.0001975504875183105 6 seconds
Selected 100 examples.
Test accuracy on selected benign examples 100.00%
Mean confidence on ground truth classes, selected 99.38%

  [>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>] PGD Attacking 100/100

---Statistics of PGD Attack (0.017287 seconds per sample)
Success rate: 100.00%, Misclassification rate: 100.00%, Mean confidence: 99.94%
Li dist: 0.3020, L2 dist: 5.1207, L0 dist: 73.0%
```

Screenshot of running *DeepFool-UA_CIFAR10_DenseNet40.py*:

```
Loading the dataset...
Evaluating the target model...
Test accuracy on benign examples 94.84%
Mean confidence on ground truth classes 92.15%
Test time per example 0.021957121586799622 seconds
Selected 100 examples.
Test accuracy on selected benign examples 100.00%
Mean confidence on ground truth classes, selected 95.55%

>> Compiling the gradient TensorFlow functions. This might take some time...
>> Computing gradient function...
  [>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>] DeepFool Attacking 100/100

---Statistics of DeepFool Attack (1.656201 seconds per sample)
Success rate: 100.00%, Misclassification rate: 100.00%, Mean confidence: 85.81%
Li dist: 0.0275, L2 dist: 0.2307, L0 dist: 99.1%
```

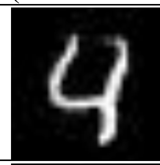
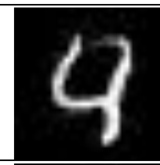
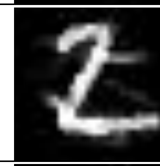
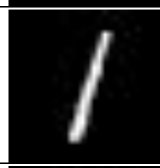
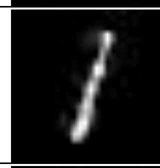
Screenshot of running *PGD-UA_CIFAR10_DenseNet40.py*:

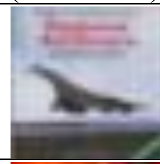
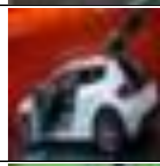
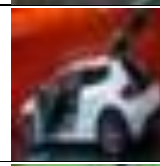
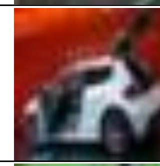
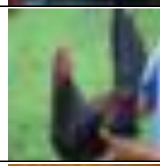
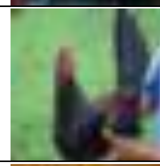
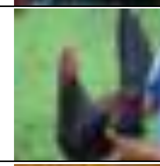
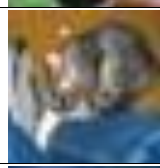
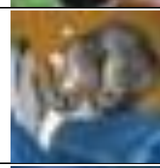
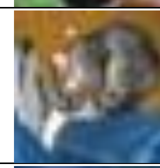
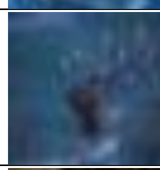
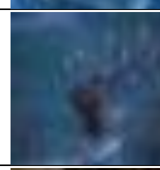
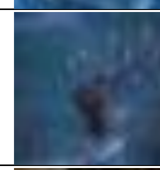
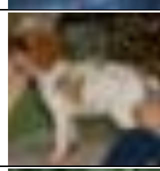
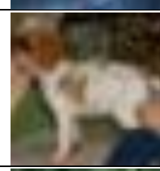
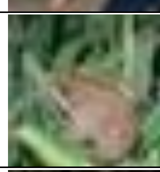
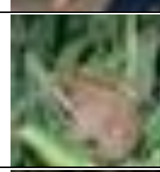
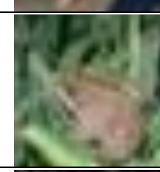
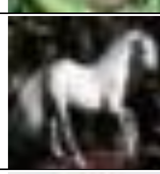
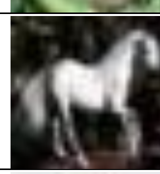
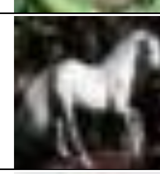
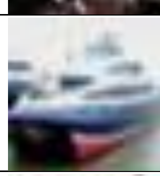
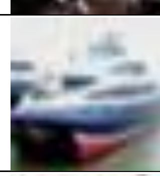
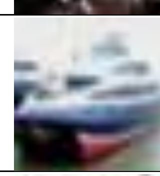
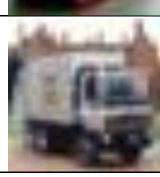
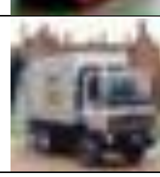
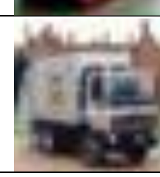
```
Loading the dataset...
Evaluating the target model...
Test accuracy on benign examples 94.84%
Mean confidence on ground truth classes 92.15%
Test time per example 0.021277007842063905 seconds
Selected 100 examples.
Test accuracy on selected benign examples 100.00%
Mean confidence on ground truth classes, selected 95.55%

  [>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>] PGD Attacking 100/100

---Statistics of PGD Attack (0.797615 seconds per sample)
Success rate: 91.00%, Misclassification rate: 91.00%, Mean confidence: 98.25%
Li dist: 0.0078, L2 dist: 0.3613, L0 dist: 99.8%
```

Output analysis (5):

MNIST+CNN-7+DeepFool	Original Image	Intermediate Image (2 iterations)	Final Image
Example 1			
Example 2			
Example 3			
Example 4			
Example 5			
Example 6			
Example 7			
Example 8			
Example 9			
Example 10			

CIFAR10+DenseNet-40+DeepFool	Original Image	Intermediate Image (1 iteration)	Final Image
Example 1			
Example 2			
Example 3			
Example 4			
Example 5			
Example 6			
Example 7			
Example 8			
Example 9			
Example 10			

Requirements (6) a. Adverse effect on different depths of CNNs.

I chose two variable-depth ResNets for this part of the assignment, ResNet-20 and ResNet-110. ResNet is short for Residual Network. A special feature in a residual neural network is that it utilizes "skip connections", which means that it will jump over some layers while training. By doing this, a ResNet can avoid the problem of vanishing gradients and mitigate the accuracy saturation problem.

Training parameters:

- batch size = 32
- epochs = 200 for ResNet-20, 150 for ResNet-110

ResNet-20

- Model statistics:
  - ```
    ================================
    Total params: 574,090
    Trainable params: 570,602
    Non-trainable params: 3,488
    ________________________________
    ResNet20v2
    ```
- First 3 epochs:
  - ```
    Epoch 1/200
    Learning rate:  0.001
    1562/1562 [==============================] - 41s 26ms/step - loss: 1.7367 - acc: 0.4919 - val_loss: 1.4401 - val_acc: 0.5813

    Epoch 00001: val_acc improved from -inf to 0.58130, saving model to ../models/weights/CIFAR10_ResNet20v2.keras_weights.h5
    Epoch 2/200
    Learning rate:  0.001
    1562/1562 [==============================] - 35s 23ms/step - loss: 1.3220 - acc: 0.6209 - val_loss: 1.9770 - val_acc: 0.5097

    Epoch 00002: val_acc did not improve from 0.58130
    Epoch 3/200
    Learning rate:  0.001
    1562/1562 [==============================] - 35s 22ms/step - loss: 1.1632 - acc: 0.6733 - val_loss: 1.2739 - val_acc: 0.6459

    Epoch 00003: val_acc improved from 0.58130 to 0.64590, saving model to ../models/weights/CIFAR10_ResNet20v2.keras_weights.h5
    ```
- Last 3 epochs:
  - ```
    Epoch 198/200
    Learning rate:  5e-07
    1562/1562 [==============================] - 35s 22ms/step - loss: 0.1893 - acc: 0.9786 - val_loss: 0.4264 - val_acc: 0.9134

    Epoch 00198: val_acc did not improve from 0.91390
    Epoch 199/200
    Learning rate:  5e-07
    1562/1562 [==============================] - 35s 22ms/step - loss: 0.1907 - acc: 0.9780 - val_loss: 0.4255 - val_acc: 0.9132

    Epoch 00199: val_acc did not improve from 0.91390
    Epoch 200/200
    Learning rate:  5e-07
    1562/1562 [==============================] - 35s 22ms/step - loss: 0.1910 - acc: 0.9785 - val_loss: 0.4263 - val_acc: 0.9126

    Epoch 00200: val_acc did not improve from 0.91390
    ```
    ```

ResNet-110

- Model statistics:
    - ```
      Total params: 3,323,210
      Trainable params: 3,302,442
      Non-trainable params: 20,768
      ______________________________
      ResNet110v2
      ```
- First 3 epochs:
    - ```
      Epoch 1/150
      Learning rate:  0.001
      1562/1562 [==============================] - 222s 142ms/step - loss: 2.3873 - acc: 0.4887 - val_loss: 1.7991 - val_acc: 0.5507

      Epoch 00001: val_acc improved from -inf to 0.55070, saving model to ../models/weights/CIFAR10_ResNet110v2.keras_weights.h5
      Epoch 2/150
      Learning rate:  0.001
      1562/1562 [==============================] - 184s 118ms/step - loss: 1.5250 - acc: 0.6202 - val_loss: 1.7509 - val_acc: 0.5713

      Epoch 00002: val_acc improved from 0.55070 to 0.57130, saving model to ../models/weights/CIFAR10_ResNet110v2.keras_weights.h5
      Epoch 3/150
      Learning rate:  0.001
      1562/1562 [==============================] - 183s 117ms/step - loss: 1.2880 - acc: 0.6821 - val_loss: 1.3985 - val_acc: 0.6394

      Epoch 00003: val_acc improved from 0.57130 to 0.63940, saving model to ../models/weights/CIFAR10_ResNet110v2.keras_weights.h5
      ```
- Last 3 epochs:
    - ```
      Epoch 148/150
      Learning rate:  1e-05
      1562/1562 [==============================] - 184s 118ms/step - loss: 0.1653 - acc: 0.9870 - val_loss: 0.4024 - val_acc: 0.9286

      Epoch 00148: val_acc improved from 0.92790 to 0.92860, saving model to ../models/weights/CIFAR10_ResNet110v2.keras_weights.h5
      Epoch 149/150
      Learning rate:  1e-05
      1562/1562 [==============================] - 183s 117ms/step - loss: 0.1652 - acc: 0.9873 - val_loss: 0.4042 - val_acc: 0.9282

      Epoch 00149: val_acc did not improve from 0.92860
      Epoch 150/150
      Learning rate:  1e-05
      1562/1562 [==============================] - 183s 117ms/step - loss: 0.1635 - acc: 0.9878 - val_loss: 0.4041 - val_acc: 0.9285

      Epoch 00150: val_acc did not improve from 0.92860
      ```

Comparison: ResNet-20 VS. ResNet-110 (under DeepFool attack)
ResNet-20:
      ```

- ```
  Loading the dataset...
  Evaluating the target model...
  Test accuracy on benign examples 84.03%
  Mean confidence on ground truth classes 81.35%
  Test time per example 0.0002786757230758667 seconds
  Selected 100 examples.
  Test accuracy on selected benign examples 100.00%
  Mean confidence on ground truth classes, selected 94.44%

  >> Compiling the gradient TensorFlow functions. This might take some time...
  >> Computing gradient function...
    [>>>>>>>>>-------------------------------] DeepFool Attacking 23/100  00:03:
  rning: divide by zero encountered in float_scalars
    pert_k = abs(f_k)/np.linalg.norm(w_k.flatten())
    [>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>] DeepFool Attacking 100/100

  ---Statistics of DeepFool Attack (0.626428 seconds per sample)
  Success rate: 93.00%, Misclassification rate: 93.00%, Mean confidence: 90.10%
  Li dist: 0.2987, L2 dist: 2.9928, L0 dist: 99.5%
  ```

ResNet-110

- ```
  Loading the dataset...
  Evaluating the target model...
  Test accuracy on benign examples 84.39%
  Mean confidence on ground truth classes 82.78%
  Test time per example 0.000787515115737915 seconds
  Selected 100 examples.
  Test accuracy on selected benign examples 100.00%
  Mean confidence on ground truth classes, selected 94.77%

  >> Compiling the gradient TensorFlow functions. This might take some time...
  >> Computing gradient function...
    [----------------------------------------] DeepFool Attacking 1/100  00:56:1
  ning: divide by zero encountered in float_scalars
    pert_k = abs(f_k)/np.linalg.norm(w_k.flatten())
    [>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>] DeepFool Attacking 100/100

  ---Statistics of DeepFool Attack (2.203672 seconds per sample)
  Success rate: 97.00%, Misclassification rate: 97.00%, Mean confidence: 87.00%
  Li dist: 0.3362, L2 dist: 3.2478, L0 dist: 99.5%
  (attack) PS D:\fall2021\cs6220\GTAttackPod>
  ```

Analysis: Comparing ResNet-20 and ResNet-110 under the same DeepFool attack, we can find that, the deeper neural network is not necessarily more resilient to attacks, as the ResNet-110 has a misclassification rate of 97% and the ResNet-20 only has a misclassification rate of 93%. However, we can see that the attack algorithm is obviously taking longer to attack ResNet-110 (2.2 seconds) versus ResNet-20 (0.6 seconds). This is because as the neural network gets deeper, there would naturally be more gradients for the attack algorithm to compute.

Requirements (6) b. Test transferability of your generated adversarial examples:

- Using adversarial examples generated by attacking DenseNet-40 on CIFAR-10 to attack ResNet-20

- Attacking DenseNet-40 with DeepFool

```
Loading the dataset...
Evaluating the target model...
Test accuracy on benign examples 94.84%
Mean confidence on ground truth classes 92.15%
Test time per example 0.0006198992729187011 seconds
Selected 100 examples.
Test accuracy on selected benign examples 100.00%
Mean confidence on ground truth classes, selected 95.55%

>> Compiling the gradient TensorFlow functions. This might take some time...
>> Computing gradient function...
  [>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>] DeepFool Attacking 100/100

---Statistics of DeepFool Attack (0.533981 seconds per sample)
Success rate: 100.00%, Misclassification rate: 100.00%, Mean confidence: 85.75%
Li dist: 0.0275, L2 dist: 0.2306, L0 dist: 99.1%
```

- Attacking ResNet-20 with the adversarial examples generated from attacking DenseNet-40

```
Loading the dataset...
Evaluating the target model...
Test accuracy on benign examples 94.84%
Mean confidence on ground truth classes 92.15%
Test time per example 0.0006412990808486939 seconds
Selected 100 examples.
Test accuracy on selected benign examples 100.00%
Mean confidence on ground truth classes, selected 95.55%

>> Compiling the gradient TensorFlow functions. This might take some time...
>> Computing gradient function...
  [>>>>>>>>>--------------------------------] DeepFool Attacking 23/100  00:03:1
rning: divide by zero encountered in float_scalars
  pert_k = abs(f_k)/np.linalg.norm(w_k.flatten())
  [>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>] DeepFool Attacking 100/100

---Statistics of DeepFool Attack (0.661609 seconds per sample)
Success rate: 91.00%, Misclassification rate: 91.00%, Mean confidence: 91.68%
Li dist: 0.3192, L2 dist: 3.4612, L0 dist: 99.5%
```

Analysis: From the above screenshots, we can see that DeepFool achieved 100% success rate on attacking DenseNet-40. While as we use the same adversarial examples to attack ResNet-20, which is also trained on CIFAR-10, it only achieves a 91% success rate. The takeaway from this is that these DeepFool attacks are very gradient-dependent. Since DenseNet-40 and ResNet-20 have very different structures and gradients, those gradient-oriented attacks may not achieve as good a performance on another model. However, we are still seeing a 91% success rate – this indicates that DeepFool has a good transferability between models, it's able to deliver comparable attack results even when the same images generated for one model are applied to attacking another.

References

https://en.wikipedia.org/wiki/Convolutional_neural_network

https://en.wikipedia.org/wiki/MNIST_database

https://arxiv.org/abs/1511.04599

https://towardsdatascience.com/know-your-enemy-7f7c5038bdf3

https://en.wikipedia.org/wiki/Residual_neural_network