CSE 598: Machine Learning Security and Fairness

Assignment 1. Train and attack classifier

PART 1: Train a Fashion-MNIST Classification

Dataset:

The dataset used is Fashion-MNIST dataset. It consists of 10 classes. The 10 classes are T-shirt/Top, Trouser, Pullover, Dress, Coat, Sandal, Shirt, Sneaker, Bag, Ankle Boot.
The dataset consists of 60000 images in the training set and 10000 images in the test set. Each image is of size 28*28. Each image is a gray scale image.
The dataset is split into three parts: Training set, Validation set and Testing set.
The training set consists of 50000 images, validation set consists of 10000 images and test set consists of 10000 images.

Architecture of the network:

The Lenet network is used for training the Fashion-MNIST. Lenet consists of total 5 layers: 3 convolutional layers and 2 fully connected layers.

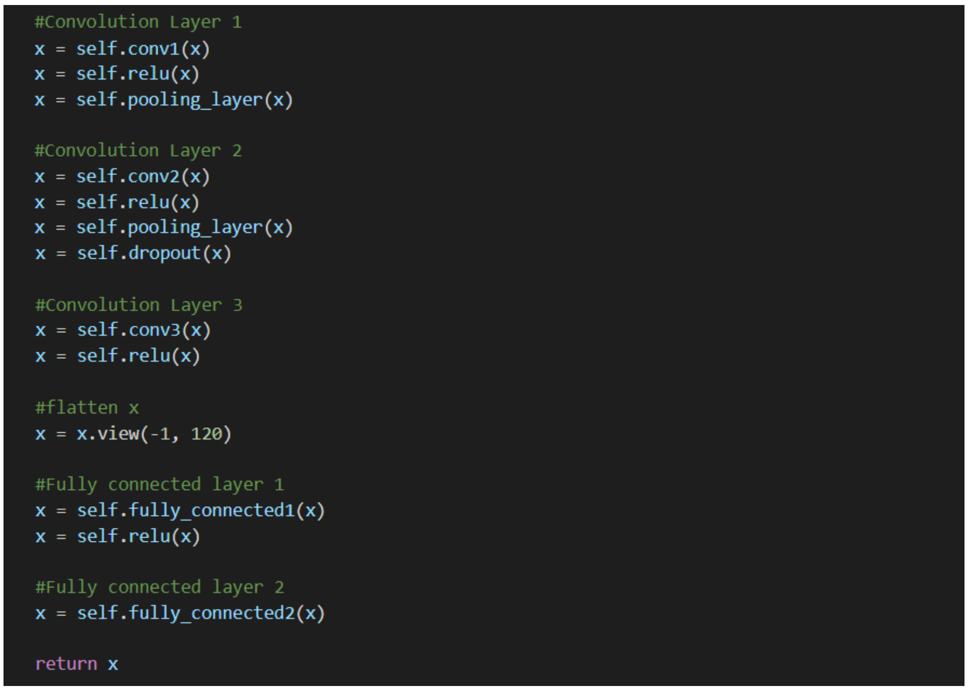
The network is defined in the forward function of the part1.py file.

Detailed description of each layer:

1. Convolution Layer 1:
    a. Input channels: 1
    b. Output channels: 6
    c. Kernel size: 5*5
    d. Stride: 1*1
2. Convolution Layer 2:
    a. Input channels: 6
    b. Output channels: 16
    c. Kernel size: 5*5
    d. Stride: 1*1
3. Convolution Layer 3:
    a. Input channels: 16
    b. Output channels: 120
    c. Kernel size: 4*4
    d. Stride: 1*1
4. Fully Connected layer 1:
    a. Input features: 120

    b. Output features: 84

5. Fully Connected layer 2:
    a. Input features: 84
    b. Output features: 10

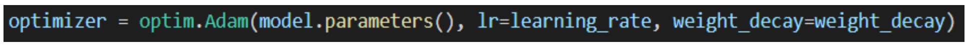


```
#Convolution Layer 1
x = self.conv1(x)
x = self.relu(x)
x = self.pooling_layer(x)

#Convolution Layer 2
x = self.conv2(x)
x = self.relu(x)
x = self.pooling_layer(x)
x = self.dropout(x)

#Convolution Layer 3
x = self.conv3(x)
x = self.relu(x)

#flatten x
x = x.view(-1, 120)

#Fully connected layer 1
x = self.fully_connected1(x)
x = self.relu(x)

#Fully connected layer 2
x = self.fully_connected2(x)

return x
```

Activation function used is ReLU.
A dropout layer is used after second convolution layer in the network with dropout percent set to 25%.

Optimizers and its parameters:

The optimizer that is used is ADAM optimizer.
The parameters of the optimizers that are used for training the model are:
Learning Rate: 0.001
Weight Decay: 1e-4



```
optimizer = optim.Adam(model.parameters(), lr=learning_rate, weight_decay=weight_decay)
```

Training Parameters:

Batch size: 32
Epoch: 40
Learning Rate: 0.001

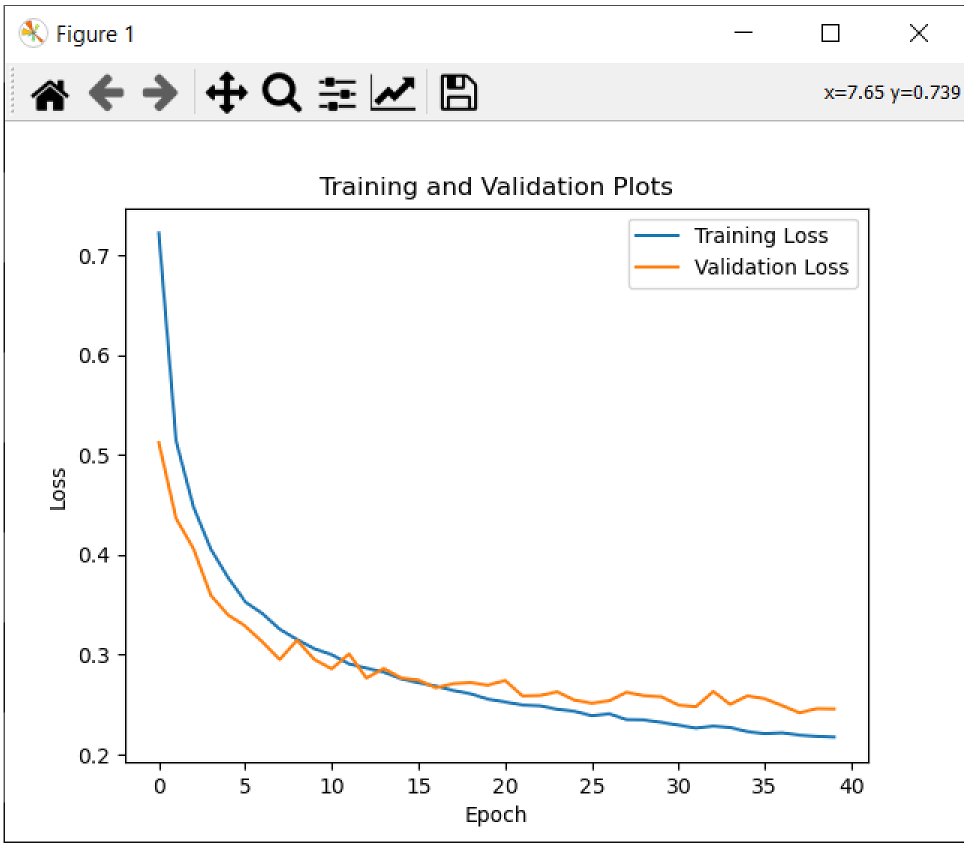
The model achieves best accuracy using the above-mentioned network, optimizer and training parameters.

Key Choices made:

1. Higher learning rate was not giving good accuracy on the test set even when it could achieve higher accuracy on training set.
2. Higher batch size for E.g.: 512, 1024 was not giving good accuracy. So a lower batch size was chosen for the best model.
3. Increasing the number of epochs to more than 100 was resulting in overfitting of the model. This was evident because as the epoch was increasing the training accuracy was increasing but the validation accuracy was decreasing.
4. Adam optimizer showed best result among SGD, RMSProp.
5. The model was overfitting so using a dropout layer was a way to deal with the overfitting of the model. Other method like augmenting the data using different augmentations like rotation at different angles which will increase the total number of training samples and reduce the overfitting problem can also be used.

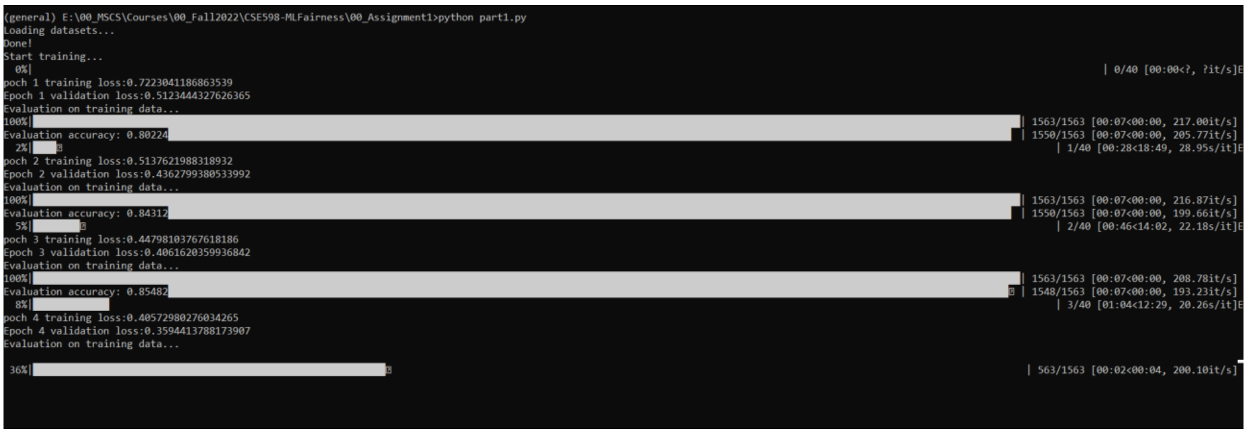
Loss Curves:

The graph below shows the training loss and validation loss for the best parameters.

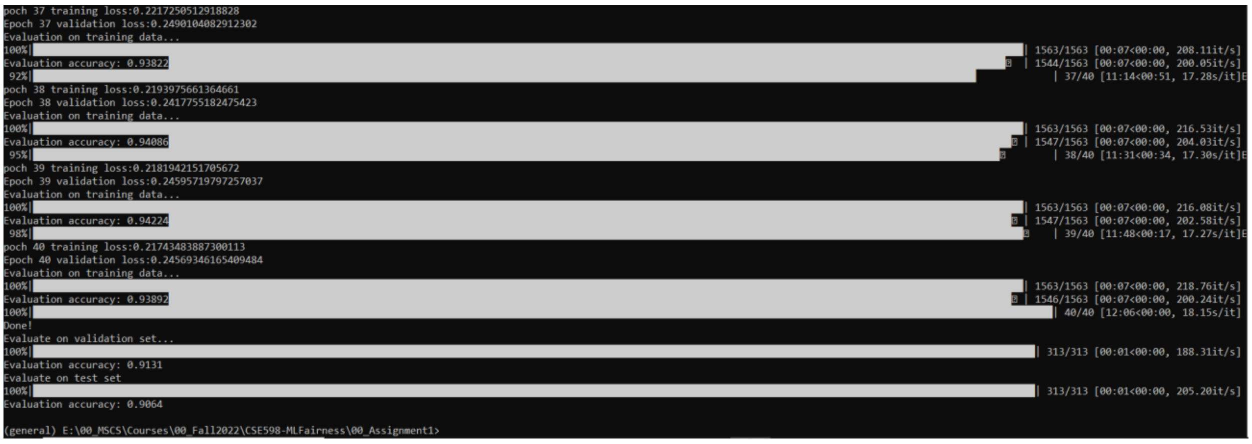


Training and Results:

The model is able to achieve an accuracy of 90.64% on the test dataset.
Test Accuracy: **90.64%**
Validation Accuracy: **91.31%**



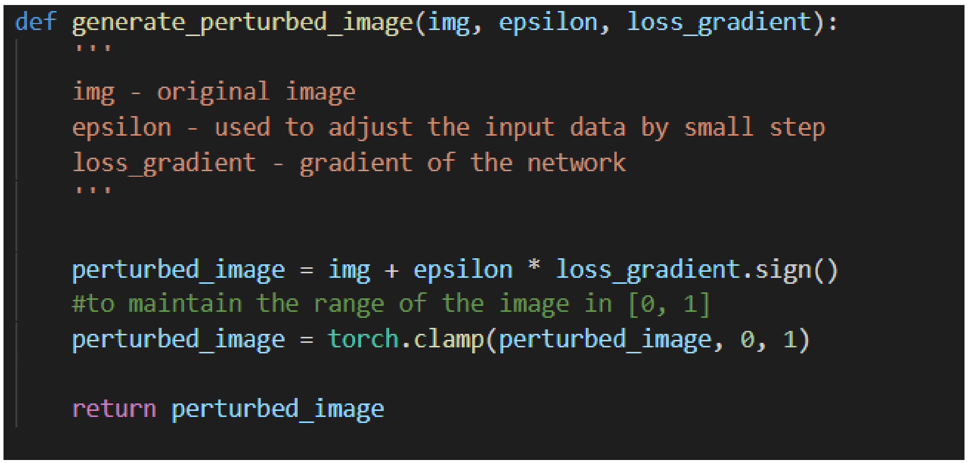
```
(general) E:\00_MSCS\Courses\00_Fall2022\CSE598-MLFairness\00_Assignment1>python part1.py
Loading datasets...
Done!
Start training...
  0%|                                                                     | 0/40 [00:00<?, ?it/s]E
poch 1 training loss:0.7223041186863539
Epoch 1 validation loss:0.5123444327626365
Evaluation on training data...
100%|                                                          | 1563/1563 [00:07<00:00, 217.00it/s]
Evaluation accuracy: 0.80224                                   | 1550/1563 [00:07<00:00, 205.77it/s]
  2%|                                                              | 1/40 [00:28<18:49, 28.95s/it]E
poch 2 training loss:0.5137621988318932
Epoch 2 validation loss:0.4362799380533992
Evaluation on training data...
100%|                                                          | 1563/1563 [00:07<00:00, 216.87it/s]
Evaluation accuracy: 0.84312                                   | 1550/1563 [00:07<00:00, 199.66it/s]
  5%|                                                              | 2/40 [00:46<14:02, 22.18s/it]E
poch 3 training loss:0.44798103767618186
Epoch 3 validation loss:0.4061620359936842
Evaluation on training data...
100%|                                                          | 1563/1563 [00:07<00:00, 208.78it/s]
Evaluation accuracy: 0.85482                                   | 1548/1563 [00:07<00:00, 193.23it/s]
  8%|                                                              | 3/40 [01:04<12:29, 20.26s/it]E
poch 4 training loss:0.40572980276034265
Epoch 4 validation loss:0.3594413788173907
Evaluation on training data...

 36%|                                                           | 563/1563 [00:02<00:04, 200.10it/s]
```



```
poch 37 training loss:0.2217250512918828
Epoch 37 validation loss:0.2490104082912302
Evaluation on training data...
100%|                                                          | 1563/1563 [00:07<00:00, 208.11it/s]
Evaluation accuracy: 0.93822                                   | 1544/1563 [00:07<00:00, 200.05it/s]
 92%|                                                             | 37/40 [11:14<00:51, 17.28s/it]E
poch 38 training loss:0.2193975661364661
Epoch 38 validation loss:0.2417755182475423
Evaluation on training data...
100%|                                                          | 1563/1563 [00:07<00:00, 216.53it/s]
Evaluation accuracy: 0.94086                                   | 1547/1563 [00:07<00:00, 204.03it/s]
 95%|                                                             | 38/40 [11:31<00:34, 17.30s/it]E
poch 39 training loss:0.2181942151705672
Epoch 39 validation loss:0.24595719797257037
Evaluation on training data...
100%|                                                          | 1563/1563 [00:07<00:00, 216.08it/s]
Evaluation accuracy: 0.94224                                   | 1547/1563 [00:07<00:00, 202.58it/s]
 98%|                                                             | 39/40 [11:48<00:17, 17.27s/it]E
poch 40 training loss:0.21743483887300113
Epoch 40 validation loss:0.24569346165409484
Evaluation on training data...
100%|                                                          | 1563/1563 [00:07<00:00, 218.76it/s]
Evaluation accuracy: 0.93892                                   | 1546/1563 [00:07<00:00, 200.24it/s]
100%|                                                             | 40/40 [12:06<00:00, 18.15s/it]
Done!
Evaluate on validation set...
100%|                                                            | 313/313 [00:01<00:00, 188.31it/s]
Evaluation accuracy: 0.9131
Evaluate on test set
100%|                                                            | 313/313 [00:01<00:00, 205.20it/s]
Evaluation accuracy: 0.9064

(general) E:\00_MSCS\Courses\00_Fall2022\CSE598-MLFairness\00_Assignment1>
```

PART 2 Implement attacks for Fashion-MNIST Classification:

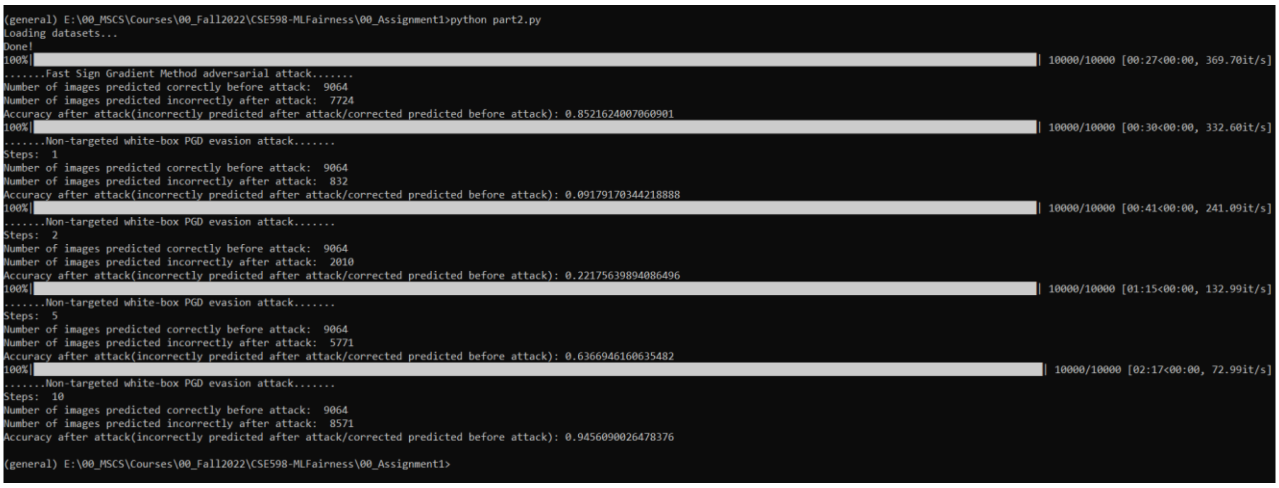
The implementation of both the attacks are present in part2.py file.
The python function generates perturbed images and performs the attack on the test dataset only.



```python
def generate_perturbed_image(img, epsilon, loss_gradient):
    '''
    img - original image
    epsilon - used to adjust the input data by small step
    loss_gradient - gradient of the network
    '''

    perturbed_image = img + epsilon * loss_gradient.sign()
    #to maintain the range of the image in [0, 1]
    perturbed_image = torch.clamp(perturbed_image, 0, 1)

    return perturbed_image
```

The functions evaluate_fsgm() and evaluate_pgd() has the implementation of fast sign gradient method and PDG evasion attack.

The following epsilon and alpha values are used:
Epsilon: 25/255
Alpha: 2/255

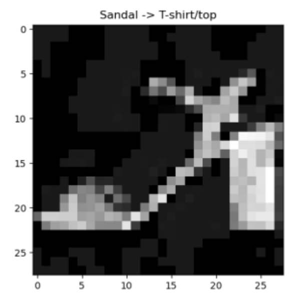


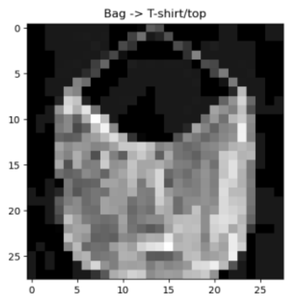
```
(general) E:\00_MSCS\Courses\00_Fall2022\CSE598-MLFairness\00_Assignment1>python part2.py
Loading datasets...
Done!
100%|██████████████████████████████████████████████████| 10000/10000 [00:27<00:00, 369.70it/s]
.......Fast Sign Gradient Method adversarial attack.......
Number of images predicted correctly before attack:  9064
Number of images predicted incorrectly after attack:  7724
Accuracy after attack(incorrectly predicted after attack/corrected predicted before attack): 0.8521624007060901
100%|██████████████████████████████████████████████████| 10000/10000 [00:30<00:00, 332.60it/s]
.......Non-targeted white-box PGD evasion attack.......
Steps:  1
Number of images predicted correctly before attack:  9064
Number of images predicted incorrectly after attack:  832
Accuracy after attack(incorrectly predicted after attack/corrected predicted before attack): 0.09179170344218888
100%|██████████████████████████████████████████████████| 10000/10000 [00:41<00:00, 241.09it/s]
.......Non-targeted white-box PGD evasion attack.......
Steps:  2
Number of images predicted correctly before attack:  9064
Number of images predicted incorrectly after attack:  2010
Accuracy after attack(incorrectly predicted after attack/corrected predicted before attack): 0.22175639894086496
100%|██████████████████████████████████████████████████| 10000/10000 [01:15<00:00, 132.99it/s]
.......Non-targeted white-box PGD evasion attack.......
Steps:  5
Number of images predicted correctly before attack:  9064
Number of images predicted incorrectly after attack:  5771
Accuracy after attack(incorrectly predicted after attack/corrected predicted before attack): 0.6366946160635482
100%|███████████████████████████████████████████████████| 10000/10000 [02:17<00:00, 72.99it/s]
.......Non-targeted white-box PGD evasion attack.......
Steps:  10
Number of images predicted correctly before attack:  9064
Number of images predicted incorrectly after attack:  8571
Accuracy after attack(incorrectly predicted after attack/corrected predicted before attack): 0.9456090026478376

(general) E:\00_MSCS\Courses\00_Fall2022\CSE598-MLFairness\00_Assignment1>
```

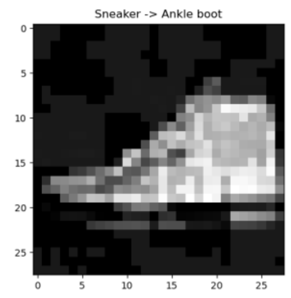
Fast Sign Gradient Method:

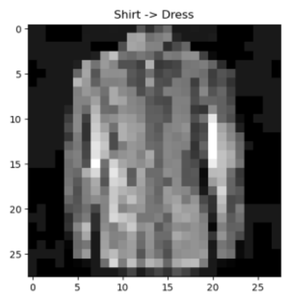
The attack accuracy of the fast sign gradient method is: **85.21%**

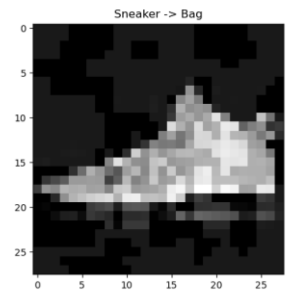
Sample Adversarial Images and its Prediction for FSGM attack:

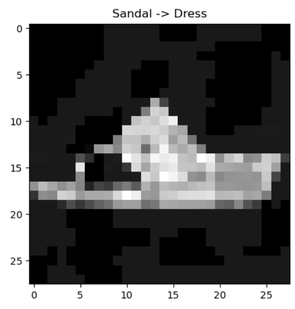


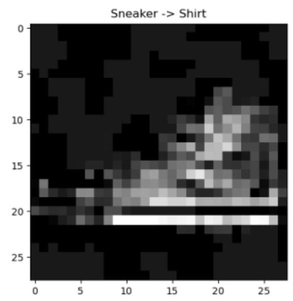


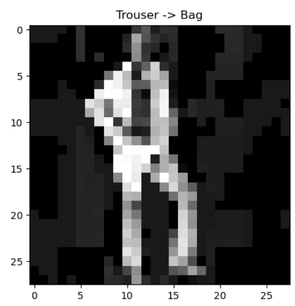


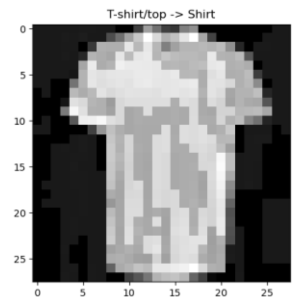


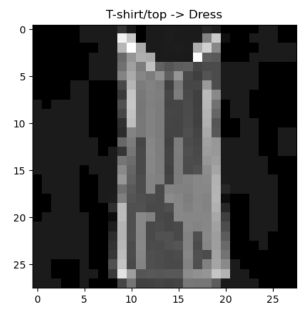










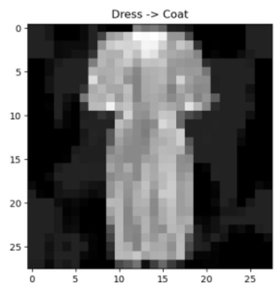


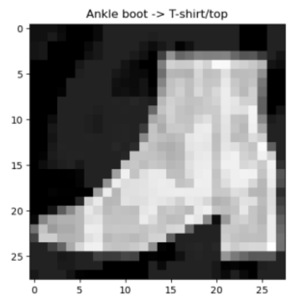
PGD Evasion Attack:

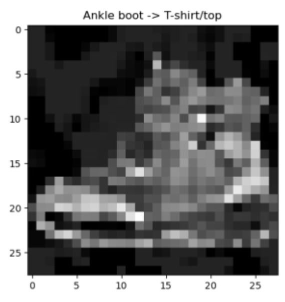
Step sizes used are 1, 2, 5, 10.

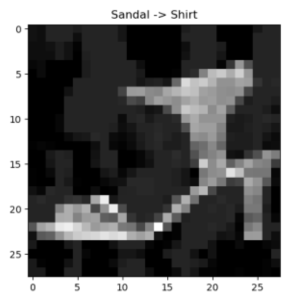
Accuracy for Step 1: **9.17%**
Accuracy for Step 2: **22.17%**
Accuracy for Step 5: **63.67%**
Accuracy for Step 10: **94.56%**

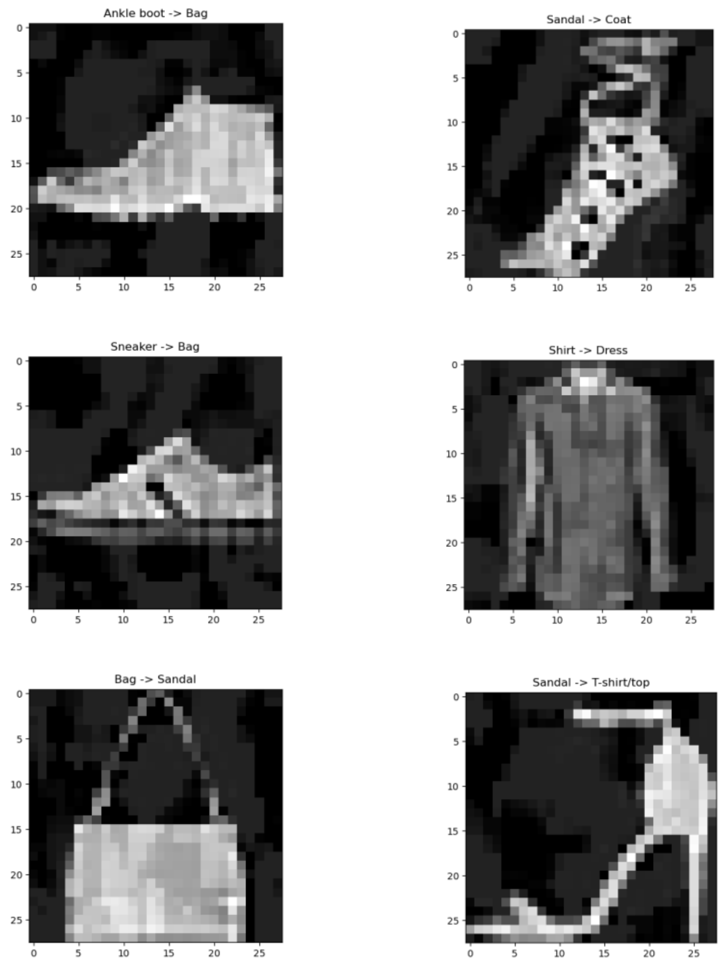
Sample Adversarial images for PGD Evasion Attack with step size 10:










Ankle boot -> Bag
Sandal -> Coat
Sneaker -> Bag
Shirt -> Dress
Bag -> Sandal
Sandal -> T-shirt/top
0
5
10
15
20
25