```
import numpy as np
import matplotlib.pyplot as plt
from tqdm import tqdm # Displays a progress bar
from pylab import *
import torch
import torchvision
from torch import nn
from torch import optim
import torch.nn.functional as F
from torchvision import datasets, transforms, utils
from torch.utils.data import Dataset, Subset, DataLoader, random_split
```

```
# Load the dataset and train, val, test splits
print("Loading datasets...")
FASHION_transform = transforms.Compose([
    transforms.ToTensor(), # Transform from [0,255] uint8 to [0,1] float
])
FASHION_trainval = datasets.FashionMNIST('.', download=True, train=True, transform=FASHION_trans
FASHION_train = Subset(FASHION_trainval, range(50000))
FASHION_val = Subset(FASHION_trainval, range(50000,60000))
FASHION_test = datasets.FashionMNIST('.', download=True, train=False, transform=FASHION_transfor
print("Done!")
```

```
Loading datasets...
Done!
```

```
# Hyperparameters for the network
lr =  3e-4  # Learning Rate
batch_sz = 64 # Batch Size
num_epochs = 25 # Epochs
momentum=0.9 # Momentum Value
epsilon = 25/255
steps = [1,2,5,10]
alpha = 1e-2
```

```
#Plot Variables
training_loss = []
val_loss = []
pgd_adv_examples = []
pgd_label = []
original_label = []
```

```
# Create dataloaders
trainloader = DataLoader(FASHION_train, batch_size=batch_sz, shuffle=True)
valloader = DataLoader(FASHION_val, batch_size=batch_sz, shuffle=True)
testloader = DataLoader(FASHION_test, batch_size=batch_sz, shuffle=True)
input_sz = 784
```

```
class Network(nn.Module):
    def __init__(self, in_channels=1):
        super(Network, self).__init__()
        self.conv1 = nn.Conv2d(in_channels=1, out_channels=20, kernel_size=(5,5), stride=(1,1),
        self.pool = nn.MaxPool2d(kernel_size=(2,2), stride=(2,2))
        self.conv2 = nn.Conv2d(in_channels=20, out_channels=50, kernel_size=(5,5), stride=(1,1),
        self.fc1 = nn.Linear(2450,500)
        self.fc2 = nn.Linear(500,10)


    def forward(self,x):
        # TODO: Design your own network, implement forward pass here
        #x = x.view(-1,28*28) # Flatten each image in the batch
        x = F.relu(self.conv1(x))
        x = self.pool(x)
        x = F.relu(self.conv2(x))
        x = self.pool(x)
        x = x.reshape(x.shape[0],-1)
        x = F.relu(self.fc1(x))
        x = self.fc2(x)
        return x
```

```
device = "cuda" if torch.cuda.is_available() else "cpu" # Configure device
model = Network().to(device)
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=lr, weight_decay=1e-4)
```

```
def train(model, loader, num_epoch = num_epochs, valloader=valloader): # Train the model
    print("Start training...")
    model.train() # Set the model to training mode
    for i in range(num_epoch):
        model.train()
        running_loss = []
        for batch, label in tqdm(loader):
            batch = batch.to(device)
            label = label.to(device)
            optimizer.zero_grad() # Clear gradients from the previous iteration
            pred = model(batch) # This will call Network.forward() that you implement
            loss = criterion(pred, label) # Calculate the loss
            running_loss.append(loss.item())
            loss.backward() # Backprop gradients to all tensors in the network
            optimizer.step() # Update trainable weights
        training_loss.append(np.mean(running_loss))
        evaluate_val_loss(model, valloader, i)
        print("Train Epoch {} loss:{}".format(i+1,np.mean(running_loss))) # Print the average lo


    print("Done!")
```

```
def evaluate(model, loader): # Evaluate accuracy on validation / test set
    model.eval() # Set the model to evaluation mode
    correct = 0
    with torch.no_grad(): # Do not calculate gradient to speed up computation
```

```
        for batch, label in tqdm(loader):
            batch = batch.to(device)
            label = label.to(device)
            pred = model(batch)
            correct += (torch.argmax(pred,dim=1)==label).sum().item()
    acc = correct/len(loader.dataset)
    print("Evaluation accuracy: {}".format(acc))
    return acc
```

```
def evaluate_val_loss(model, loader, i):
    model.eval()
    running_loss = []
    for batch, label in tqdm(loader):
        batch = batch.to(device)
        label = label.to(device)
        optimizer.zero_grad() # Clear gradients from the previous iteration
        pred = model(batch) # This will call Network.forward() that you implement
        loss = criterion(pred, label) # Calculate the loss
        running_loss.append(loss.item())
    val_loss.append(np.mean(running_loss))
    print("Val Epoch {} loss:{}".format(i+1,np.mean(running_loss)))
```

```
def plot_loss(val_loss, train_loss):

    # Training Loss
    plt.figure(figsize=(10,5))
    plt.title("Training Loss")
    plt.plot(train_loss,label="train")
    plt.xlabel("iterations")
    plt.ylabel("Train Loss")
    plt.legend()
    plt.savefig("trainloss.png")

    # Validation Loss
    plt.figure(figsize=(10,5))
    plt.title("Validation Loss")
    plt.plot(val_loss,label="val")
    plt.xlabel("iterations")
    plt.ylabel("Validation Loss")
    plt.legend()
    plt.savefig("valloss.png")
```

```
def fgsm_attack(image, epsilon, batch_grad):
    sign_batch_grad = batch_grad.sign()
    perturbed_image = image + epsilon*sign_batch_grad
    perturbed_image = torch.clamp(perturbed_image, 0, 1)
    return perturbed_image
```

```
def pgd_attack(images, labels, n, eps=epsilon):
    clamp_max = 255
    for i in range(n):
        images.requires_grad = True
        outputs = model(images)

        model.zero_grad()
        cost = criterion(outputs, labels).to(device)
        cost.backward()

        attack_images = images + alpha*images.grad.sign()
        a = torch.clamp(images - eps, min=0)
        b = (attack_images>=a).float()*attack_images + (a>attack_images).float()*a
        c = (b > images+eps).float()*(images+eps) + (images+eps >= b).float()*b
        images = torch.clamp(c,max=clamp_max).detach_()

    return images
```

```
def test(model, device, test_loader, epsilon):

    # Accuracy counter
    c_pgd = 0
    w_pgd = 0
    correct = 0
    wrong = 0

    # Loop over all examples in test set
    for batch, label in tqdm(test_loader):
        batch, label = batch.to(device), label.to(device)
        # Set requires_grad attribute of tensor. Important for Attack
        batch.requires_grad = True
        # Forward pass the batch through the model
        output = model(batch)

        # Calculate the loss
        loss = criterion(output, label)

        # Zero all existing gradients
        model.zero_grad()

        # Calculate gradients of model in backward pass
        loss.backward()

        # Collect datagrad
        batch_grad = batch.grad.data

        # Call FGSM Attack
        perturbed_batch = fgsm_attack(batch, epsilon, batch_grad)
        # Re-classify the perturbed image
        pred = model(perturbed_batch)
        # Check for success
        correct += (torch.argmax(pred,dim=1)==label).sum().item()
        wrong += (torch.argmax(pred,dim=1)!=label).sum().item()

        # Call PGD Attack and metrics
        pgd_batch = pgd_attack(batch,label,steps[3],epsilon) #Change index of steps list here t
        pred_pgd = model(pgd_batch)
        # Save adv batch for later use
        pgd_adv_examples.append(pgd_batch)
        pgd_label.append(pred_pgd)
        original_label.append(label)
        c_pgd += (torch.argmax(pred_pgd,dim=1)==label).sum().item()
        w_pgd += (torch.argmax(pred_pgd,dim=1)!=label).sum().item()

    acc = correct/len(testloader.dataset)
    attack_acc = wrong/len(testloader.dataset)
    attack_pgd = w_pgd/len(testloader.dataset)
    print("Test accuracy on Attacked Test Data is: {} and Attack Success Rate (accuracy) is: {}"
    print("Attack Success Rate (accuracy) for PGD on Test Data is: {}".format(attack_pgd))
```

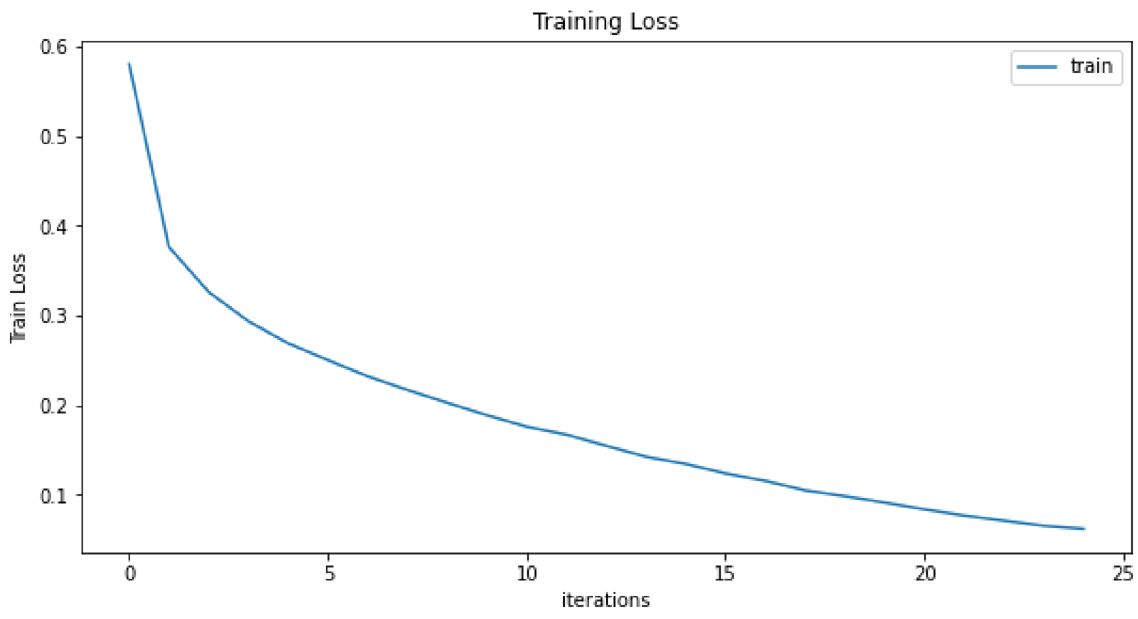
```
def imshow(img, title,i):
    npimg = img.numpy()
    plt.figure(figsize = (15, 10))
    plt.imshow(np.transpose(npimg,(1,2,0)))
    plt.title(title)
    plt.tight_layout()
    plt.savefig("Plot{}.png".format(i))
```

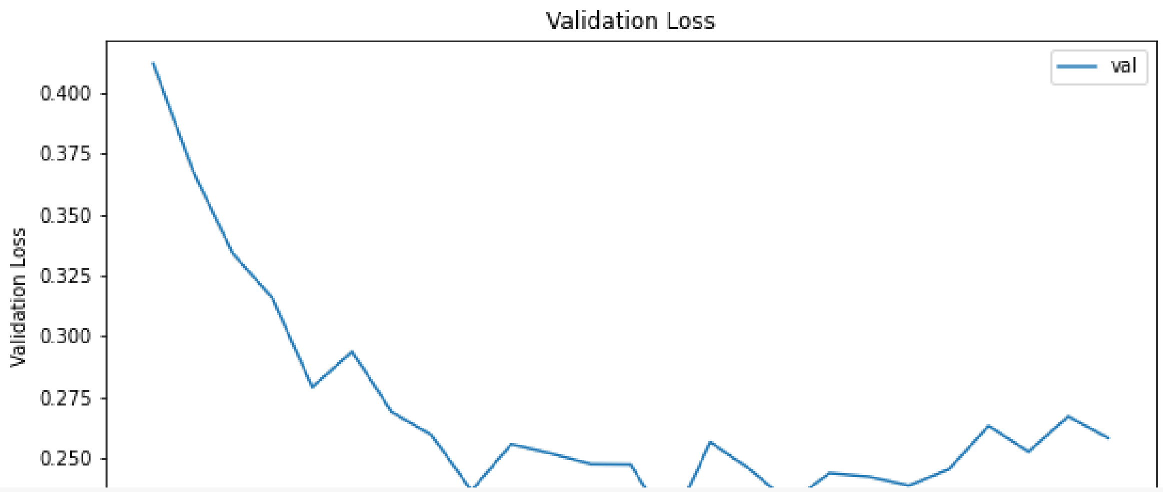
```
def run():

    train(model, trainloader)
    print("Evaluate on validation set...")
    evaluate(model, valloader)
    print("Evaluate on test set")
    evaluate(model, testloader)

    # Print Model Architecture
    print(model)

    save_path = "Lenet5-FashionMNISTtrialv1.pt"

    # Save model
    torch.save(model, save_path)
```

```
# Uncomment the comment below to retrain the model and create a new model file
# (Change torch save path)
run()
```

```
100%|██████████| 157/157 [00:01<00:00, 155.73it/s]
Val Epoch 15 loss:0.256417419310588
Train Epoch 15 loss:0.13453369210128818
100%|██████████| 782/782 [00:06<00:00, 120.32it/s]
100%|██████████| 157/157 [00:01<00:00, 156.97it/s]
Val Epoch 16 loss:0.24524141520641413
Train Epoch 16 loss:0.1240274702554164
100%|██████████| 782/782 [00:06<00:00, 116.11it/s]
100%|██████████| 157/157 [00:01<00:00, 153.98it/s]
Val Epoch 17 loss:0.2313012615273333
Train Epoch 17 loss:0.11579520696097666
Train Epoch 17 loss:0.1137[illegible]
100%|██████████| 782/782 [00:06<00:00, 119.53it/s]
100%|██████████| 157/157 [00:00<00:00, 157.67it/s]
Val Epoch 18 loss:0.2436873173447931
Train Epoch 18 loss:0.10517046021421433
100%|██████████| 782/782 [00:06<00:00, 120.93it/s]

100%|██████████| 157/157 [00:01<00:00, 156.85it/s]
Val Epoch 19 loss:0.24213088000086463
Train Epoch 19 loss:0.09862132198737024
100%|██████████| 782/782 [00:06<00:00, 119.55it/s]
100%|██████████| 157/157 [00:00<00:00, 158.04it/s]
Val Epoch 20 loss:0.23851143673157235
Train Epoch 20 loss:0.09147130147508724
100%|██████████| 782/782 [00:06<00:00, 120.16it/s]
100%|██████████| 157/157 [00:01<00:00, 156.09it/s]
Val Epoch 21 loss:0.24525239392165926
Train Epoch 21 loss:0.08401766061768545
100%|██████████| 782/782 [00:06<00:00, 120.12it/s]
100%|██████████| 157/157 [00:01<00:00, 155.83it/s]
Val Epoch 22 loss:0.2630361841789856
Train Epoch 22 loss:0.07710762176891346
100%|██████████| 782/782 [00:06<00:00, 120.37it/s]
100%|██████████| 157/157 [00:01<00:00, 155.73it/s]
Val Epoch 23 loss:0.2524482058064573
Train Epoch 23 loss:0.07140273056314576
100%|██████████| 782/782 [00:06<00:00, 118.78it/s]
100%|██████████| 157/157 [00:01<00:00, 156.36it/s]
Val Epoch 24 loss:0.26692588655811966
Train Epoch 24 loss:0.06578282328665523
100%|██████████| 782/782 [00:06<00:00, 119.11it/s]
100%|██████████| 157/157 [00:01<00:00, 155.31it/s]
Val Epoch 25 loss:0.25811262737224056
Train Epoch 25 loss:0.062239889574744515
Done!
Evaluate on validation set...
100%|██████████| 157/157 [00:00<00:00, 160.23it/s]
Evaluation accuracy: 0.9165
Evaluate on test set
100%|██████████| 157/157 [00:00<00:00, 158.76it/s]Evaluation accuracy: 0.9124
Network(
  (conv1): Conv2d(1, 20, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2))
  (pool): MaxPool2d(kernel_size=(2, 2), stride=(2, 2), padding=0, dilation=1, ceil_mode=False)
  (conv2): Conv2d(20, 50, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2))
  (fc1): Linear(in_features=2450, out_features=500, bias=True)
  (fc2): Linear(in_features=500, out_features=10, bias=True)
)
```

```
model_path = "Lenet5-FashionMNISTtrialv1.pt"

# Load model for inference
attack_model = torch.load(model_path)
attack_model.eval()

# Plot train loss and val acc
plot_loss(val_loss, training_loss)
```

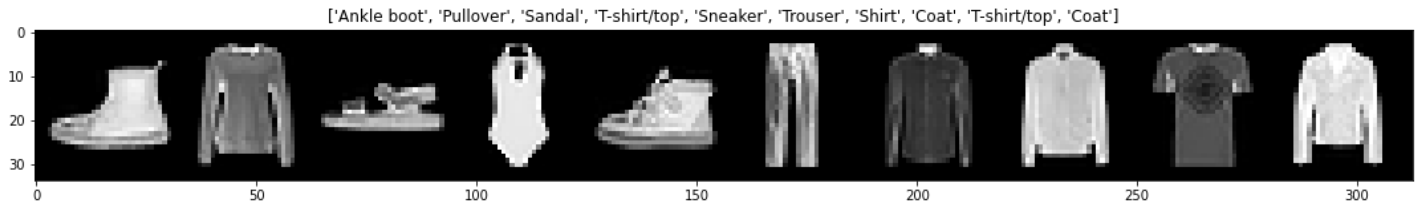




```
# Attack the Model

#=====White Box FGSM and PGD Attack=====#

test(attack_model, device, testloader, epsilon)

# get some random test images and display them with image grid
dataiter = iter(testloader)
images, labels = dataiter.next()
images = images[0:10]
labels = labels[0:10]
img_grid_normal = torchvision.utils.make_grid(images.cpu().data, nrow=10, normalize="False", pac
imshow(img_grid_normal,[FASHION_test.classes[i] for i in labels],1)

# Get adverserial Images
i=pgd_adv_examples[0][0:10]
l=pgd_label[0][0:10]
l=torch.argmax(l,dim=1)
original_l = original_label[0][0:10]

print(l)

print(labels)
```

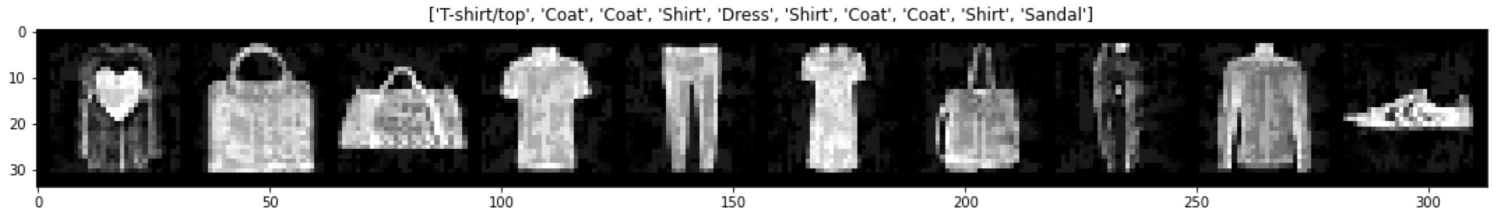
```
# create grid of adverserial images
img_grid = torchvision.utils.make_grid(i.cpu().data, nrow=10, normalize="False", padding=3)
```

```
100%|██████████| 157/157 [00:04<00:00, 33.18it/s]
Test accuracy on Attacked Test Data is: 0.1661 and Attack Success Rate (accuracy) is: 0.8339
Attack Success Rate (accuracy) for PGD on Test Data is: 0.9457
tensor([0, 4, 4, 6, 3, 6, 4, 4, 6, 5], device='cuda:0')
tensor([9, 2, 5, 0, 7, 1, 6, 4, 0, 4])
```


['Ankle boot', 'Pullover', 'Sandal', 'T-shirt/top', 'Sneaker', 'Trouser', 'Shirt', 'Coat', 'T-shirt/top', 'Coat']

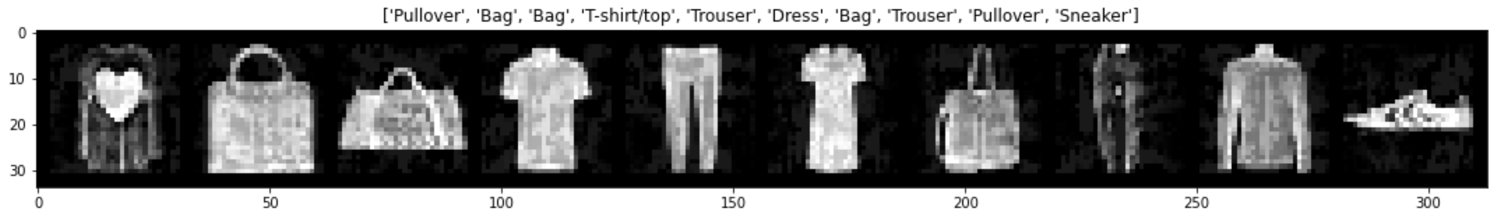
```
# Display Adversarial Images
imshow(img_grid,[FASHION_test.classes[i] for i in l],2)
```

```
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB data ([0..1] for flo
```


['T-shirt/top', 'Coat', 'Coat', 'Shirt', 'Dress', 'Shirt', 'Coat', 'Coat', 'Shirt', 'Sandal']

```
# Display Adverserial Images with True Labels
imshow(img_grid,[FASHION_test.classes[i] for i in original_l],3)
```

```
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB data ([0..1] for flo
```


['Pullover', 'Bag', 'Bag', 'T-shirt/top', 'Trouser', 'Dress', 'Bag', 'Trouser', 'Pullover', 'Sneaker']

```
evaluate(attack_model, testloader)
```

```
100%|██████████| 157/157 [00:00<00:00, 159.78it/s]Evaluation accuracy: 0.9124

0.9124
```