CSCI677-HW6

Nisha Tiwari | USC ID- 7888495181 | email- nishatiw@usc.edu

- Results
  - Configuration
  - Best performing classes
  - Worst performing classes
  - Detailed Analysis
- Untargeted FGSM
  - Quantitative Results
  - Visualization results
- UnTargeted I-FGSM
  - Quantitative Results
  - Visualization results
- Targeted FGSM
  - Visualization results
  - Quantitative Result
- Targeted I-FGSM
  - Visualization Result
  - Quantitative Results
- Code Description
  - Main Logic
  - Code to run the various experiments
  - Results visualization
  - Model Definition

Results

Configuration

All the results and findings are based on the following configurations
Epsilon values: 1e-3, 1e-2 and 1e-1
Iteration for I-FGSM: 10

Best performing classes

(avg for all eps values)

Settings	T1	T2
M1	Dog	Cat
M2	Cat	Cat

Worst performing classes

(avg for all eps values)

Settings	T1	T2
M1	Ship	Airplane
M2	Airplane	Ship

Detailed Analysis

1. As we increase the epsilon value, the attacker performs well.
2. The classes which do not perform well in the baseline model are the most susceptible to perturbation.
3. As expected, I-FGSM performs better than FGSM for both T1 and T2 methods.
4. For T2, using I-FGSM increases the performance of the attacker by almost ~3 times.
5. For all the configs, either the dog or the cat class is the best performing class. This signifies that the baseline model's incapability of differentiating well from a similar class is utilized by the attacker.
6. The out-performance of I-FGSM over FGSM is better demonstrated in T2 than T1.

Untargeted FGSM

Data in tables 1.1 and 2.1 are filled by first selecting 200 images for each class randomly and then applying perturbations only on the examples correctly classified by the baseline model. For Tables 1.1 and 2.1, columns with column heading eps=**(%) signify the percentage of successful attacks for a class. The percentage is computed over true positives in the baseline model. The number of true positives for every 200 examples is mentioned along with the class. Top Scorer column presents the class classified (with the number of instances) most number of times.

Table 1.2 and 2.2 shows the visualization results. For each class for lowest epsilon (1e-3), various image outputs are put together for randomly selected images.

Quantitative Results

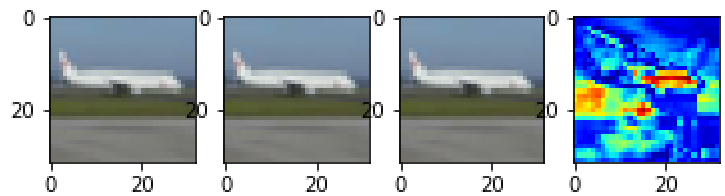
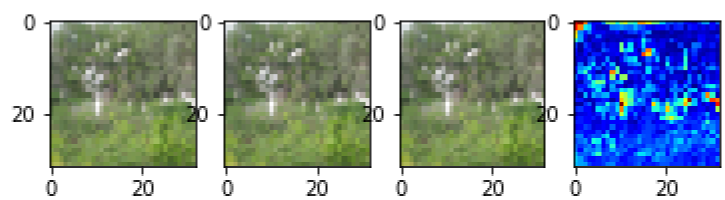
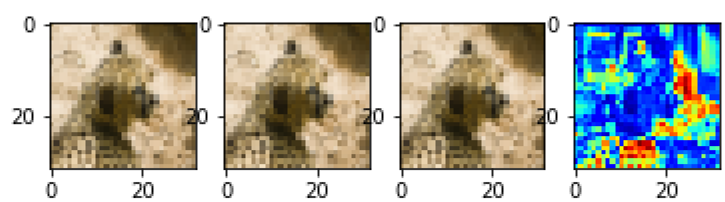
Table 1.1

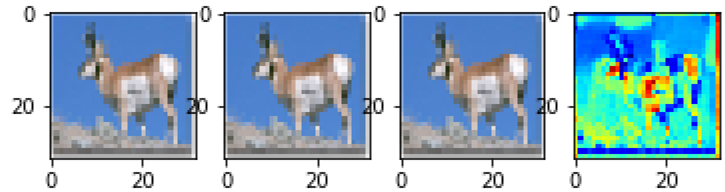
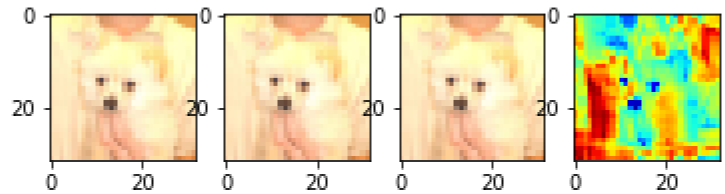
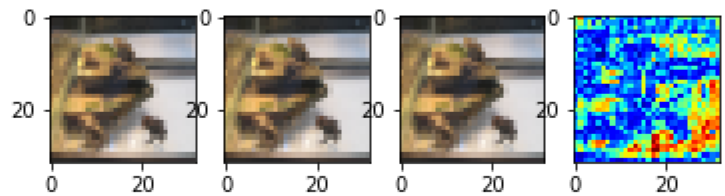
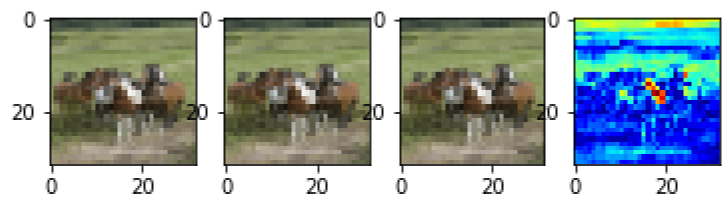
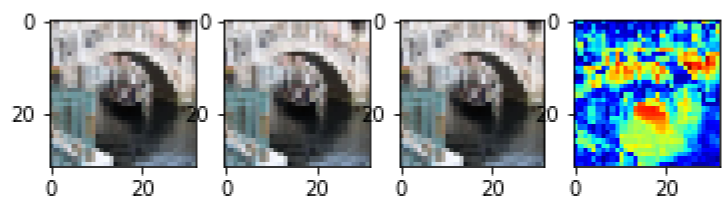
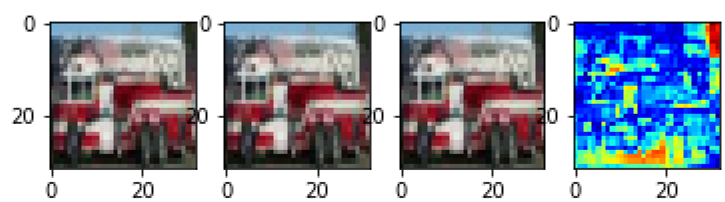
Class (tp/tp+fp)	eps= 1e-3(%)	Top scorer Class (# instances)	eps= 1e-2(%)	Top scorer class(# instances)	eps= 1e-1(%)	Top scorer class(# instances)
Airplane (153/200)	3.92	ship(4)	20.26	ship(10)	95.10	bird (47)
Automobile (143/200)	3.49	truck (5)	33.10	truck (31)	94.44	truck (78)
Bird (119/200)	5.04	Airplane (3)	49.62	frog(18)	98.33	airplane(23)
Cat (100/200)	9.0	Frog (4)	60.39	dog (23)	100.0	dog (38)
Deer (131/200)	6.10	dog (3)	49.15	cat (15)	100.0	dog (40)
Dog (90/200)	8.88	cat (2)	63.04	cat (20)	100.0	cat (34)
Frog (138/200)	5.79	deer (3)	26.89	cat (18)	97.84	bird (54)
Horse (129/200)	1.55	cat(1)	29.77	deer(15)	97.76	deer(58)

Ship (148/200)	0.67	airplane(1)	32.16	bird(11)	94.83	airplane(48)
Truck (148/200)	3.78	horse (2)	25.71	automobile(21)	97.88	automobile(86)

Visualization results

Table1.2

Class	**image (original)** **image (after attack)** **Perturbations (directly)** **Perturbations (heat map)**
Airplane	
Automobile	
Bird	
Cat	

Deer	
Dog	
Frog	
Horse	
Ship	
Truck	

UnTargeted I-FGSM

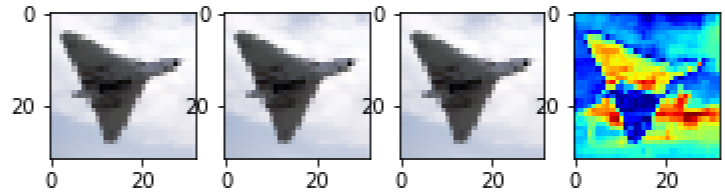
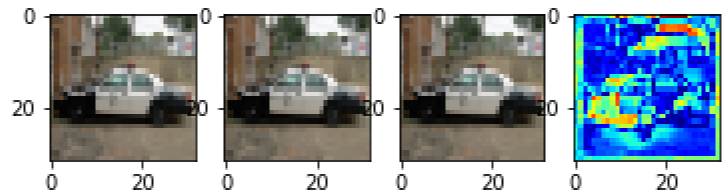
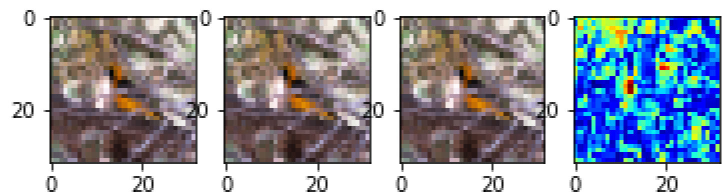
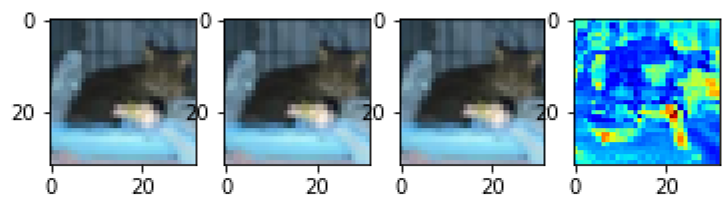
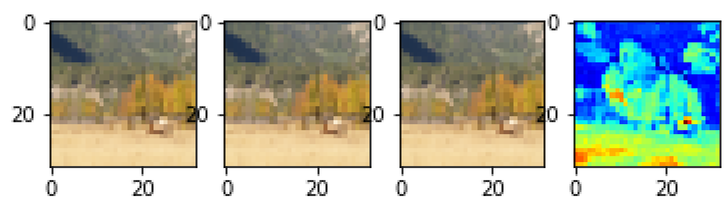
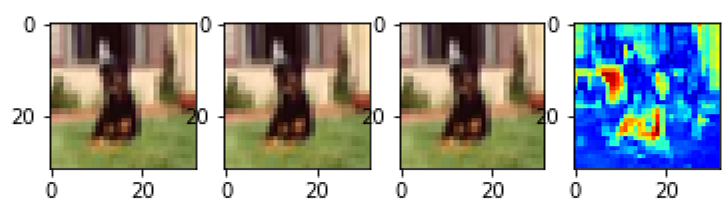
Quantitative Results

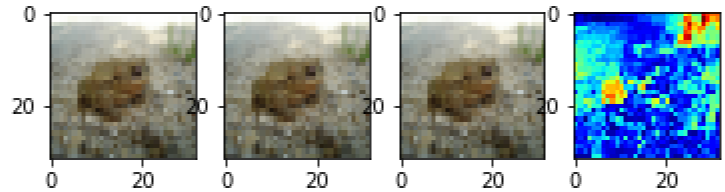
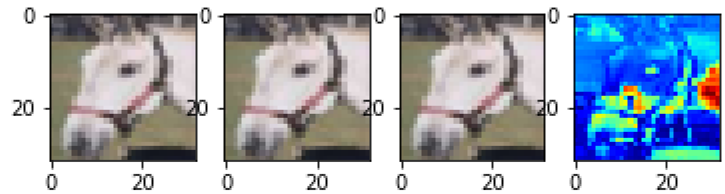
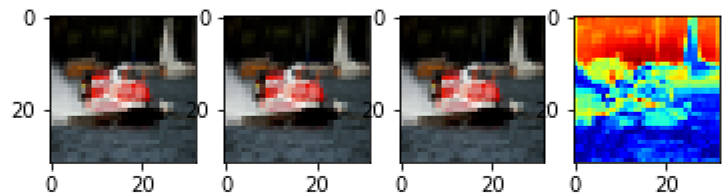
Table 2.1

Class (tp/tp+fp)	eps= 1e-3(%)	Top scorer Class (# instances)	eps= 1e-2(%)	Top scorer class(# instances)	eps= 1e-1(%)	Top scorer class(# instances)
Airplane (152/200)	2.63	bird (2)	25.51	ship (16)	98.61	bird (54)
Automobile (136/200)	3.67	truck (2)	23.25	truck (17)	100.0	truck (96)
Bird (136/200)	5.14	frog (3)	43.79	deer (16)	100.0	dog (26)
Cat (107/200)	12.14	deer (4)	63.15	dog (19)	100.0	dog (27)
Deer (128/200)	6.25	cat (3)	55.2	cat (22)	100.0	cat (29)
Dog (93/200)	8.60	bird (3)	59.52	cat (22)	100.0	cat (28)
Frog (151/200)	1.32	dog (1)	32.51	cat (20)	100.0	bird (51)
Horse (139/200)	3.59	deer (2)	33.08	dog (16)	100.0	deer (58)
Ship (139/200)	4.31	airplane(3)	35.06	airplane (23)	100.0	airplane (46)
Truck (128/200)	2.34	automobile (2)	33.33	automobile(20)	100.0	automobile(77)

Visualization results

Table 2.2

Class	Image (original) Image (after attack) Perturbations (directly) Perturbations (heat map)
Airplane	
Automobile	
Bird	
Cat	
Deer	
Dog	

Frog	
Horse	
Ship	
Truck	

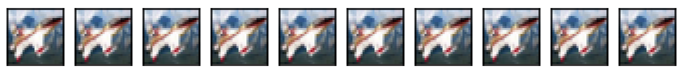
Targeted FGSM

Table 3.1 and 4.1 shows the perturbed images in a matrix, one for each combination of source and target categories. The images in the display are randomly selected. The epsilon value used for the results here is 0.03.

Table 3.2 and 4.2 shows quantitative results on 10 randomly selected images from each class and using the other nine classes as targets (one at a time). The diagonal entry in the table is not used for any inference or calculations.The entries in green color shows the cases where the attacker performed well. Red color entries shows the opposite. To summarize the results well, I have created column from-class(%) and row to-class(%). From-class(%) gives a percentage of successful attacks for a given class. To-class(%), on the other hand, contains percentage value for how many times a class is successfully selected as a target.

Visualization results

Table 3.1

class	0 1 2 3 4 5 6 7 8 9
0	
1	
2	
3	
4	
5	
6	
7	
8	
9	

Quantitative Result

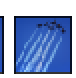
Table 3.2

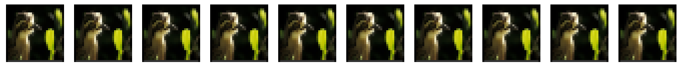
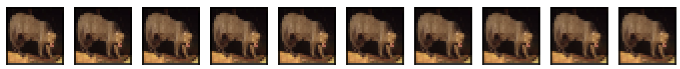
class	Airplane	Automobile	Bird	Cat	Deer	Dog	Frog	Horse	Ship	Truck	from-class (%)
Airplane	10	2	5	2	3	1	2	2	8	1	28.89
Automobile	3	10	5	1	3	3	4	2	6	8	38.89
Bird	4	1	10	6	7	4	7	4	5	0	42.22
Cat	1	3	9	10	10	8	10	5	8	3	**63.33**
Deer	2	2	7	7	10	7	10	6	5	2	53.33
Dog	3	2	6	10	7	10	3	8	2	4	50.00
Frog	2	2	6	6	7	4	10	1	6	3	41.11
Horse	3	3	1	4	7	7	2	10	4	2	36.67
Ship	8	4	5	1	2	1	3	2	10	4	33.33
Truck	6	10	2	3	2	2	1	3	9	10	42.22
to_class(%)	35.56	32.22	51.11	44.44	53.33	41.11	46.67	36.67	**58.89**	30.00	

Targeted I-FGSM

Visualization Result

Table 4.1

class	0	1	2	3	4	5	6	7	8	9
0										

1	
2	
3	
4	
5	
6	
7	
8	
9	

Quantitative Results

Table 4.2

class	**Airplane**	**Automobile**	**Bird**	**Cat**	**Deer**	**Dog**	**Frog**	**Horse**	**Ship**	**Truck**	**from-class(%)**
Airplane	10	10	10	8	9	8	9	9	10	10	92.22
Automobile	9	10	10	9	9	9	7	10	10	10	92.22

Bird	9	7	10	10	10	10	9	10	10	7	91.11
Cat	10	9	10	10	10	10	10	10	10	9	**97.78**
Deer	8	6	10	10	10	10	10	10	10	9	92.22
Dog	8	9	10	10	10	10	10	10	10	9	95.56
Frog	9	8	9	10	9	8	10	8	9	9	87.78
Horse	8	8	10	8	10	9	8	10	8	7	84.44
Ship	10	9	9	7	8	9	7	6	10	10	83.33
Truck	10	10	10	10	10	9	7	10	10	10	95.56
To-class(%)	90.00	84.44	**97.78**	91.11	94.44	91.11	85.56	92.22	**96.67**	88.89	

Code Description

Main Logic

Following are the changes done over the class provided with the starter code. The main updates are in perturbed_untargeted, perturbed_tageted, and generate_experiment methods.

```
"""
Adversary Attack example code
"""
# from torchvision import models as tvm
import torch
from torch.nn import functional as F
from pathlib import Path
from PIL import Image
import numpy as np
import torchvision.transforms as transforms
from torch.autograd import Variable


class AdversialAttacker(object):

    def __init__(self, method='FGSM'):
        assert method in ['FGSM', 'I-FGSM']
        self.method = method
        self.criterion = torch.nn.CrossEntropyLoss()
        print("created adversial attacker in method '%s'" % (method))

    def get_pred_label(self, mdl, inp, ret_out_scores=False, ret_out_pred=True):
        # use current model to get predicted label
```

```
    train = mdl.training
    mdl.eval()
    with torch.no_grad():
        out = F.softmax(mdl(inp), dim=1)
    out_score, out_pred = out.max(dim=1)
    if ret_out_scores and not ret_out_pred:
        return out
    if ret_out_pred and not ret_out_scores:
        return out_pred
    #mdl.train(train)
    return out_pred, out

def perturb_untargeted(self, mdl, inp, targ_label=None, eps=0.3):
    # perform attacking perturbation in the untargeted setting
    # note: feel free the change the function arguments for your implementation
    out_pred, out_score=self.get_pred_label(mdl,inp,True,True)
    x_adv = Variable(inp.data, requires_grad=True)
    if self.method == 'FGSM':

       cost = -self.criterion(h_adv, out_pred)

        mdl.zero_grad()
        if x_adv.grad is not None:
            x_adv.grad.data.fill_(0)
        cost.backward()

        x_adv.grad.sign_()
        x_adv = x_adv - eps * x_adv.grad

        h = mdl(inp)
        h_adv = mdl(x_adv)

        return x_adv, h_adv, h

    elif self.method == 'I-FGSM':
        iteration=10
        alpha=eps/iteration
        out_pred, out_score=self.get_pred_label(mdl,inp,True,True)
        x_adv = Variable(inp.data, requires_grad=True)
        for i in range(iteration):
            h_adv = mdl(x_adv)
            cost = -self.criterion(h_adv, out_pred)
            mdl.zero_grad()
            if x_adv.grad is not None:
                x_adv.grad.data.fill_(0)
            cost.backward()

            x_adv.grad.sign_()
            x_adv = x_adv - alpha * x_adv.grad
            x_adv = torch.where(x_adv > inp + eps, inp + eps, x_adv)
```

```python
            x_adv = torch.where(x_adv < inp - eps, inp - eps, x_adv)
            x_adv = Variable(x_adv.data, requires_grad=True)
        h = mdl(inp)
        h_adv = mdl(x_adv)
        return x_adv, h_adv, h

def perturb_targeted(self, mdl, inp, targ_label, eps=0.3):
    # perform attacking perturbation in the targeted setting
    # note: feel free the change the function arguments for your implementation
    #mdl.train()  # switch model to train mode
    out_pred, out_score=self.get_pred_label(mdl,inp,True,True)
    x_adv = Variable(inp.data, requires_grad=True)
    if self.method == 'FGSM':
        h_adv = mdl(x_adv)
        cost = self.criterion(h_adv, targ_label)

        mdl.zero_grad()
        if x_adv.grad is not None:
            x_adv.grad.data.fill_(0)
        cost.backward()

        x_adv.grad.sign_()
        x_adv = x_adv - eps * x_adv.grad

        h = mdl(inp)
        h_adv = mdl(x_adv)

        return x_adv, h_adv, h

    elif self.method == 'I-FGSM':
        iteration=10
        alpha=eps/iteration
        # TODO
        # you may add arguments like iter, eps_iter, ...
        out_pred, out_score=self.get_pred_label(mdl,inp,True,True)
        x_adv = Variable(inp.data, requires_grad=True)
        for i in range(iteration):
            h_adv = mdl(x_adv)
            cost = self.criterion(h_adv, targ_label)
            mdl.zero_grad()
            if x_adv.grad is not None:
                x_adv.grad.data.fill_(0)
            cost.backward()

            x_adv.grad.sign_()
            x_adv = x_adv - alpha * x_adv.grad
            x_adv = torch.where(x_adv > inp + eps, inp + eps, x_adv)
            x_adv = torch.where(x_adv < inp - eps, inp - eps, x_adv)
            x_adv = Variable(x_adv.data, requires_grad=True)
        h = mdl(inp)
```

```python
            h_adv = mdl(x_adv)
            return x_adv, h_adv, h

class Clamp:
    def __call__(self, inp):
        return torch.clamp(inp, 0., 1.)


def generate_experiment(image_path,method='FGSM'):

    # define your model and load pretrained weights
    # TODO
    # model = ...
    model = Net()
    # trained model path.
    model.load_state_dict(torch.load("/content/drive/My Drive/Colab Notebooks/model64"))

    # cinic class names
    import yaml
    with open('/content/cinic_classnames.yml', 'r') as fp:
        classnames = yaml.safe_load(fp)

    # load image
    # TODO:
    # img_path = Path(....)
    #/content/test/airplane/cifar10-test-9577.png
    input_img = Image.open(image_path)

    # define normalizer and un-normalizer for images
    # cinic
    mean = [0.47889522, 0.47227842, 0.43047404]
    std = [0.24205776, 0.23828046, 0.25874835]

    tf_img = transforms.Compose(
        [
            # transforms.Resize((224, 224)),
            transforms.ToTensor(),
            transforms.Normalize(
                mean=mean,
                std=std
            )
        ]
    )
    un_norm = transforms.Compose(
        [
            transforms.Normalize(
                mean=[-m/s for m, s in zip(mean, std)],
                std=[1/s for s in std]
            ),
            Clamp(),
```

```
            transforms.ToPILImage()
        ]
    )

    # To be used for iterative method
    # to ensure staying within Linf limits
    clip_min = min([-m/s for m, s in zip(mean, std)])
    clip_max = max([(1-m)/s for m, s in zip(mean, std)])

    input_tensor = tf_img(input_img)
    attacker = AdversialAttacker(method=method)

    return {
        'img': input_img,
        'inp': input_tensor.unsqueeze(0),
        'attacker': attacker,
        'mdl': model,
        'clip_min': clip_min,
        'clip_max': clip_max,
        'un_norm': un_norm,
        'classnames': classnames
    }
```

Code to run the various experiments

Following code snippet was used (by making various changes for every desired result) to generate all the quantitative/visualization results. For each source class to each other target class, it calls the M1 or M2 method for a few randomly selected images and prints them.

```
count=0
import glob
import yaml
import random
with open('/content/cinic_classnames.yml', 'r') as fp:
  classnames = yaml.safe_load(fp)
TEST_DATA_PATH="/content/test/"+classnames[count]+"/"
test = glob.glob(TEST_DATA_PATH+"*.png")
selectedList=random.sample(test, 50)

import PIL
result={}
target=0
list_image=[]
while(target<10):
  prevCorrectlyClassified=0
  print("Current class:: %s", classnames[target])
  #selectedList=random.choices(range(count*10000, (count+1)*10000), k=200)
```

```
  result[classnames[count]]=[]
  for  i in selectedList:
    image=PIL.Image.open(i)
    tensor=transforms.ToTensor()
    #to handle size issues
    if tensor(image).shape!=(3,32,32):
      continue
    x=generate_experiment(i)
    input_img    = x['img']
    input_tensor = x['inp']
    attacker     = x['attacker']
    model        = x['mdl']
    un_norm      = x['un_norm']
    classnames   = x['classnames']
    # run the classifier model
    out_pred, scores = attacker.get_pred_label(model, input_tensor, ret_out_scores=True,
ret_out_pred=True)
    x_adv, h_adv, h = attacker.perturb_targeted(model, input_tensor,
targ_label=torch.LongTensor([target]), eps=1e-1)
    result[classnames[count]].append({
          'input_tensor':input_tensor,
          'img': input_img,
          'x_adv':x_adv,
          'h_adv':h_adv,
          'h':h,
          'out_pred':out_pred,
          'scores':scores
      })
  output={}
  #print(len(result[classnames[count]]))
  targetAttackSuccessful=0
  for i in range(0,len(result[classnames[count]])):
    h=result[classnames[count]][i]['h']
    h_adv=result[classnames[count]][i]['h_adv']
    img=result[classnames[count]][i]['img']
    out_pred=result[classnames[count]][i]['out_pred']
    scores=result[classnames[count]][i]['scores']
    #out_pred=result[classnames[count]][i]['out_pred']
    x_adv=result[classnames[count]][i]['x_adv']
    input_tensor=result[classnames[count]][i]['input_tensor']
    img_adv = un_norm(x_adv.squeeze(0))
    img_diff = diff_img(img_adv, un_norm(input_tensor.squeeze(0)), scale=1) # you can play
with scale to amplify the signals
    img_orig_np = np.array(un_norm(input_tensor.squeeze(0))).astype('float')
    img_adv_np  = np.array(img_adv).astype('float')
    img_diff_np = np.abs( img_adv_np - img_orig_np ).sum(axis=2)
    # run classifier again for the attacked image
    attacked_pred, attacked_score = attacker.get_pred_label(model, x_adv, ret_out_scores=True,
ret_out_pred=True)
```

```
    #only consider the examples which were correctly classified in the base model
    if int(out_pred)==count:
      prevCorrectlyClassified+=1
      if int(attacked_pred) ==target:
        targetAttackSuccessful+=1
      #only need 10 examplease
      if prevCorrectlyClassified==10:
       break
      '''if int(attacked_pred) in output:
        output[int(attacked_pred)]=output[int(attacked_pred)]+1
      else:
        output[int(attacked_pred)]=1'''

      #list_image.append([img,img_adv,img_diff, img_diff_np])
      list_image.append(img_adv)
      #print( "original prediction: %d current prediction: %d (%s)\n" % (
int(out_pred),int(attacked_pred), classnames[int(attacked_pred)] ) )
# code to generate the values for untargeted method's table
  #sorted_output = sorted(output.items(), key=operator.itemgetter(1))
  #print(sum(output.values())*100/prevCorrectlyClassified,sum(output.values()))

#print(classnames[sorted_output[len(sorted_output)-1][0]],"("+str(sorted_output[len(sorted_out
put)-1][1])+")")
  target+=1
  #showImagesHorizontally(list_image[0])
```

Results visualization

The following code helps in visualizing target images for each source horizontally.

```
from matplotlib.pyplot import figure, imshow, axis
from matplotlib.image import imread

def showImagesHorizontally(list_of_files):
    f,ax = pyplot.subplots(1,10)
    number_of_files = len(list_of_files)
    for i in range(number_of_files):
        ax[i].axes.get_xaxis().set_visible(False)
        ax[i].axes.get_yaxis().set_visible(False)
        ax[i].imshow(image)
    pyplot.show()
```

Model Definition

The code below provides the definition of the LeNet model used for assignment 4.

```
import torch.nn as nn
import torch.nn.functional as F


class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(3, 32, 3)
        self.pool1 = nn.MaxPool2d(2, 2)
        self.bn1= nn.BatchNorm2d(num_features=32, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        self.conv2 = nn.Conv2d(32, 64, 4)
        self.pool2 = nn.MaxPool2d(2, 2)
        self.bn2= nn.BatchNorm2d(num_features=64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        self.fc1 = nn.Linear(64*6*6, 600)
        self.fc2 = nn.Linear(600, 200)
        self.fc3 = nn.Linear(200, 10)

    def forward(self, x):
        x = self.pool1(F.relu(self.conv1(x)))
        x=self.bn1(x)
        x = self.pool2(F.relu(self.conv2(x)))
        x=self.bn2(x)
        x = x.view(-1, 64*6*6)
        nn.Dropout(0.5)
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x= F.log_softmax(self.fc3(x))
        return x
```