CS6910 : Assignment 1 (Code)

Sarthak Vora - EE19B140 & Akshat Joshi - EE19B136

March 17, 2022

Github Repo Link

1 Task 1

1.1 train.py

Python script for training the model

```python
import torch
import torch.nn as nn
from torch.utils.data import DataLoader
import torch.optim as optim
from dataset import function_dataset
from model import function_approximation

data_dir = 'dataset/train.csv'
device = 'cuda' if torch.cuda.is_available() else 'cpu'
batch_size = 1
learning_rate = 2e-6
epochs = 350
momentum = 0.9

train_dataset = function_dataset(data_dir)
train_loader = DataLoader(dataset=train_dataset, batch_size=batch_size, shuffle=True)

losses = []

model = function_approximation().to(device=device)

criterion = nn.MSELoss()
optimizer = optim.SGD(model.parameters(), lr=learning_rate, momentum=momentum)


for epoch in range(epochs):
    total_loss = 0
    cnt = 0
    for batch_idx, (data, target) in enumerate(train_loader):
        data = data.to(device=device)
        target = target.to(device=device)

        out = model(data.float())
        loss = criterion(out, target.float())
        cnt+=1
        total_loss += loss.item()

        optimizer.zero_grad()
        loss.backward()

        optimizer.step()
    losses.append(total_loss/cnt)
    print(f'Epochs:{epoch+1}, Loss:{total_loss/cnt}')
    if(epoch+1 == 1 or epoch+1==2 or epoch+1==10 or epoch+1==50 or epoch+1==epochs):
        torch.save(model.state_dict(), "epoch"+str(epoch+1)+".pt")

with open("losses.txt","w") as f:
```

```
    for l in losses:
        f.write(str(l)+"\n")
torch.save(model.state_dict(), 'model_weights.pth')
```

1.2 eval.py

Python script for generating metrics

```
import torch
import torch.nn as nn
from torch.utils.data import DataLoader
from model import function_approximation
from dataset import function_dataset

device = 'cuda' if torch.cuda.is_available() else 'cpu'

model = function_approximation().to(device=device)
model.load_state_dict(torch.load('model_weights.pth'))

train_dataset = function_dataset("dataset/train.csv")
train_loader = DataLoader(train_dataset)

test_dataset = function_dataset("dataset/test.csv")
test_loader = DataLoader(test_dataset)

valid_dataset = function_dataset("dataset/validation.csv")
valid_loader = DataLoader(valid_dataset)


criterion = nn.MSELoss()


def accuracy(loader, model):



    avg_loss = 0
    cnt=0

    with torch.no_grad():
        for data, target in loader:
            model.eval()
            data = data.to(device=device)
            target = target.to(device=device)

            out = model(data.float())

            loss = criterion(out,target.float())
            avg_loss += loss
            cnt+=1
        avg_loss = avg_loss/cnt
        print(f"Average loss is: {avg_loss:.2f}")
        print("-----------------------------------------------------------------------------------")


print("Train Set metrics:")
accuracy(train_loader, model)

print("Test Set metrics:")
accuracy(test_loader, model)

print("Validation Set metrics:")
accuracy(valid_loader, model)
```

1.3 plot.py

Python script for generating plots

```
import torch
```

```
import torch.nn as nn
from torch.utils.data import DataLoader
from dataset import function_dataset
from model import function_approximation
import numpy as np
import matplotlib.pyplot as plt
from matplotlib import cm
import pandas as pd

df = pd.read_csv('dataset/func_app1.csv')

dir = "plots/"

#device = 'cuda' if torch.cuda.is_available() else 'cpu'
device = 'cpu'



train_dataset = function_dataset("dataset/train.csv")
train_loader = DataLoader(train_dataset)

test_dataset = function_dataset("dataset/test.csv")
test_loader = DataLoader(test_dataset)

valid_dataset = function_dataset("dataset/validation.csv")
valid_loader = DataLoader(valid_dataset)


def gen_plots(model,epoch):
    x1 = np.arange(0,6,0.25,dtype="float32")
    x2 = np.arange(0,6,0.25,dtype="float32")
    x1,x2 = np.meshgrid(x1,x2)

    y = np.zeros(x1.shape)

    for i in range(x1.shape[0]):
        for j in range(x1.shape[1]):

            output = model(torch.tensor([x1[i][j],x2[i][j]]))
            y[i][j]= output

    f = plt.figure()
    ax = plt.axes(projection='3d')
    surf = ax.plot_surface(x1, x2, y, cmap = cm.jet, linewidth=0, antialiased=False)
    f.colorbar(surf, shrink=0.5, aspect=10)
    ax.set_title(f'Approximated Function after {epoch} Epochs')
    ax.set_xlabel('x1')
    ax.set_ylabel('x2')
    plt.savefig(dir+'epoch'+f'{epoch}'+"_approximated.png")
    plt.show()


# Plot of loss variation with epoch
losses = []
with open("losses.txt","r") as f:
    lines = f.readlines()
    for l in lines:
        losses.append(float(l.strip()))

epochs = np.arange(1,len(losses)+1,1)
losses = np.array(losses)
f = plt.figure()
plt.title("Loss v/s Epoch")
plt.plot(epochs,losses)
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.savefig(dir+"loss.png")
plt.show()
plt.close(f)


```

```
# Scatter plot of desired output v/s approximated output
model = function_approximation().to(device=device)
model.load_state_dict(torch.load('model_weights.pth'))

desired =[]
approximated = []


for batch_idx, (data, target) in enumerate(train_loader):
        data = data.to(device=device)
        target = target.to(device=device)

        out = model(data.float())

        approximated.append(out.item())
        desired.append(target.item())
desired = np.array(desired)
approximated = np.array(approximated)


f = plt.figure()
plt.title("Desired v/s Approximated Scatter Plot")
plt.scatter(desired, approximated, c='b', linewidths=1)
plt.plot(desired, desired, 'r')
plt.xlabel('Desired Function')
plt.ylabel('Approximated Function')
plt.savefig(dir+"scatter.png")
plt.show()
plt.close(f)


f = plt.figure()
ax = plt.axes(projection='3d')
surf = ax.plot_trisurf(df.iloc[:,0], df.iloc[:,1], df.iloc[:,2], cmap=cm.jet, linewidth=0,
    antialiased=False)
f.colorbar(surf, shrink=0.5, aspect=10)
ax.set_title(f'Desired Function')
ax.set_xlabel('x1')
ax.set_ylabel('x2')
ax.set_zlabel('Desired Function')
plt.savefig(dir+"desired.png")
plt.show()


for epoch in [1,2,10,50,350]:
    model = function_approximation().to(device)
    model.load_state_dict(torch.load(f"epoch{str(epoch)}.pt",map_location=device))
    gen_plots(model,epoch)
```

1.4 model.py

Python script describing PyTorch model

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


class function_approximation(nn.Module):
    def __init__(self):
        super(function_approximation, self).__init__()

        self.linear1 = nn.Linear(in_features=2, out_features=8, bias=True)
        self.linear2 = nn.Linear(in_features=8, out_features=4)
        self.linear3 = nn.Linear(in_features=4, out_features=1)
        self.tanh    = nn.Tanh()
        self.softmax = nn.Softmax(dim=0)


    def forward(self, x):
```

```
        x = self.tanh(self.linear1(x))
        x = self.tanh(self.linear2(x))
        x = self.linear3(x)

        return x

def test():
    model = function_approximation()
    input = torch.Tensor([4.321097794848372, 4.769609253163742])
    out = model(input)

    print(input)
    print(model)
    print(out)

#test()
```

1.5 dataset.py

Python script giving access to the Dataset

```
from torch.utils.data import Dataset
import pandas as pd
import numpy as np
import torch


class function_dataset(Dataset):
    def __init__(self, data_dir):
        self.data_dir = data_dir
        data = np.array(pd.read_csv(self.data_dir))
        self.input_features = data[:,0:2]
        self.target = data[:,2:]
        self.len = data.shape[0]

    def __len__(self):
        return self.len

    def __getitem__(self, index):
        features_index = torch.from_numpy(self.input_features[index])
        target_index = torch.from_numpy(self.target[index])

        return (features_index, target_index)

def test():
    train_dataset = function_dataset(data_dir='dataset/func_app1.csv')
    train_data, train_label = train_dataset[5]

    # print(train_data)
    # print(train_label)
    # print(train_data.shape)
    # print(train_label.shape)

    print(train_dataset[:][0][:,0].shape)

#test()
```

2 Task 2a

2.1 train.py

Python script for training the model

```
import torch
from dataset import Task2aDataset
from model import FFNN
import torch.optim as optim
```

```
import torch.nn  as nn

criterion = nn.BCELoss()

device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
batch_size = 1
epochs = 360

trainDataset = Task2aDataset("dataset/train_t25.csv")
trainLoader  = torch.utils.data.DataLoader(trainDataset, batch_size=batch_size,
                                           shuffle=True)
model = FFNN()
model.to(device)

optimizer = optim.SGD(model.parameters(),lr=0.0001,momentum=0.7)



print(device)

for epoch in range(epochs):
    print("Current Epoch: ",epoch)
    total_loss = 0
    cnt = 0
    for i,data in enumerate(trainLoader,0):
        cnt+=1
        optimizer.zero_grad()
        inputs, labels = data
        inputs = inputs.to(device)
        labels = labels.to(device)
        # Model
        hl1outputs = model(inputs,path='hidden1').to(device)
        hl2outputs = model(hl1outputs,path='hidden2').to(device)
        outputs = model(hl2outputs,path='output')


        labels = labels.unsqueeze(1)
        loss = criterion(outputs,labels)
        total_loss += loss.item()







        loss.backward()
        optimizer.step()
    if(epoch+1 == 1 or epoch+1==2 or epoch+1==10 or epoch+1==50 or epoch+1==epochs):
        torch.save(model.state_dict(), "epoch"+str(epoch+1)+".pt")
    print("Epoch Average Loss: ", total_loss/(cnt))

torch.save(model.state_dict(), "weights.pt")
```

2.2 eval.py

Python script for generating metrics

```
import torch
import torch.nn as nn

from model import FFNN
from dataset import Task2aDataset
device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")

model = FFNN()
model.load_state_dict(torch.load("weights.pt",map_location=device))


trainSet = Task2aDataset("dataset/train_t25.csv")

trainLoader = torch.utils.data.DataLoader(trainSet, batch_size=1,
```

```
                                            shuffle=True)

devSet = Task2aDataset("dataset/dev_t25.csv")

devLoader = torch.utils.data.DataLoader(devSet, batch_size=1,
                                            shuffle=True)

criterion = nn.BCELoss()
def get_metrics(loader):
    TP = 0
    TN = 0
    FP = 0
    FN = 0

    total_loss = 0
    cnt = 0
    for i,data in enumerate(loader):
        cnt+=1
        inputs,labels = data
        outputs = model(inputs.to(device))
        labels = labels.unsqueeze(1)
        loss = criterion(outputs,labels)

        total_loss+=loss.item()
        if(outputs[0][0] > 0.5):
                if(labels[0][0] == 1.0):
                    TP+=1
                else:
                    FP+=1
        else:
            if(labels[0][0] == 0.0):
                TN+=1
            else:
                FN+=1
    accuracy = (TP+TN)/(FP+FN+TP+TN)
    precision=TP/(TP+FP)
    recall=TP/(TP+FN)


    print("Average Loss: ",total_loss/cnt)
    print("TP:",TP)
    print("TN:",TN)
    print("FP:",FP)
    print("FN:",FN)

    print("Accuracy: ",accuracy)
    print("Precision: ", precision)
    print("Recall: ", recall)
    print("F1 score:",2/(1/recall + 1/precision))


print("==Train Set Metrics==")
get_metrics(trainLoader)



print("==Dev Set Metrics==")
get_metrics(devLoader)
```

2.3 plot.py

Python script for generating plots

```
import matplotlib.pyplot as plt
import torch.nn as nn
import torch
import numpy as np
from model import FFNN
from matplotlib import cm

```

```
from dataset import Task2aDataset

dir = "plots/"
epochs = [1,2,10,50,360]
device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")

def gen_plots(model,epoch):
    name = dir + "epoch" + str(epoch)
    x1 = np.arange(-20,20,0.25,dtype="float32")
    x2 = np.arange(-20,20,0.25,dtype="float32")
    x1,x2 = np.meshgrid(x1,x2)



    hl11 = np.zeros(x1.shape)
    hl12 = np.zeros(x1.shape)
    hl13 = np.zeros(x1.shape)
    hl14 = np.zeros(x1.shape)
    hl21 = np.zeros(x1.shape)
    hl22 = np.zeros(x1.shape)
    y = np.zeros(x1.shape)


    for i in range(x1.shape[0]):
        for j in range(x1.shape[1]):

            hl1 = model(torch.tensor([x1[i][j],x2[i][j]]),path="hidden1")
            hl2 = model(hl1,path="hidden2")
            output = model(hl2,path="output")

            #print(hl1)

            hl11[i][j] = hl1[0]
            hl12[i][j]= hl1[1]
            hl13[i][j]= hl1[2]
            hl14[i][j]= hl1[3]

            hl21[i][j]= hl2[0]
            hl22[i][j]= hl2[1]

            y[i][j]= output[0]

    f = plt.figure()
    fig, ax = plt.subplots(subplot_kw={"projection": "3d"})
    surf = ax.plot_surface(x1, x2, hl11, cmap=cm.coolwarm,linewidth=0, antialiased=False)
    ax.set_title(f"HL 1, Node 1 (epoch: {epoch} )")
    ax.set_xlabel("x1")
    ax.set_ylabel("x2")
    ax.set_zlabel("Value")

    plt.savefig(name+"_hl11.png")
    plt.close(f)

    f = plt.figure()
    fig, ax = plt.subplots(subplot_kw={"projection": "3d"})
    surf = ax.plot_surface(x1, x2, hl12, cmap=cm.coolwarm,linewidth=0, antialiased=False)
    ax.set_title(f"HL 1, Node 2 (epoch: {epoch} )")
    ax.set_xlabel("x1")
    ax.set_ylabel("x2")
    ax.set_zlabel("Value")
    plt.savefig(name+"_hl12.png")
    plt.close(f)

    f = plt.figure()
    fig, ax = plt.subplots(subplot_kw={"projection": "3d"})
    surf = ax.plot_surface(x1, x2, hl13, cmap=cm.coolwarm,linewidth=0, antialiased=False)
    ax.set_title(f"HL 1, Node 3 (epoch: {epoch} )")
    ax.set_xlabel("x1")
    ax.set_ylabel("x2")
    ax.set_zlabel("Value")
    plt.savefig(name+"_hl13.png")
```

```
    plt.close(f)

    f = plt.figure()
    fig, ax = plt.subplots(subplot_kw={"projection": "3d"})
    surf = ax.plot_surface(x1, x2, hl14, cmap=cm.coolwarm,linewidth=0, antialiased=False)
    ax.set_title(f"HL 1, Node 4 (epoch: {epoch} )")
    ax.set_xlabel("x1")
    ax.set_ylabel("x2")
    ax.set_zlabel("Value")
    plt.savefig(name+"_hl14.png")
    plt.close(f)

    f = plt.figure()
    fig, ax = plt.subplots(subplot_kw={"projection": "3d"})
    surf = ax.plot_surface(x1, x2, hl21, cmap=cm.coolwarm,linewidth=0, antialiased=False)
    ax.set_title(f"HL 2, Node 1 (epoch: {epoch} )")
    ax.set_xlabel("x1")
    ax.set_ylabel("x2")
    ax.set_zlabel("Value")
    plt.savefig(name+"_hl21.png")
    plt.close(f)

    f = plt.figure()
    fig, ax = plt.subplots(subplot_kw={"projection": "3d"})
    surf = ax.plot_surface(x1, x2, hl22, cmap=cm.coolwarm,linewidth=0, antialiased=False)
    ax.set_title(f"HL 2, Node 2 (epoch: {epoch} )")
    ax.set_xlabel("x1")
    ax.set_ylabel("x2")
    ax.set_zlabel("Value")
    plt.savefig(name+"_hl22.png")
    plt.close(f)

    f = plt.figure()
    fig, ax = plt.subplots(subplot_kw={"projection": "3d"})
    surf = ax.plot_surface(x1, x2, y, cmap=cm.coolwarm,linewidth=0, antialiased=False)
    ax.set_title(f"Output (epoch: {epoch} )")
    ax.set_xlabel("x1")
    ax.set_ylabel("x2")
    ax.set_zlabel("Value")
    plt.savefig(name+"_output.png")
    plt.close(f)


def create_decision_plot(name):
    model = FFNN()
    model.load_state_dict(torch.load(f"weights.pt",map_location=device))

    x1 = np.arange(-20,20,0.25,dtype="float32")
    x2 = np.arange(-20,20,0.25,dtype="float32")
    x1,x2 = np.meshgrid(x1,x2)
    y = np.zeros(x1.shape)
    for i in range(x1.shape[0]):
        for j in range(x1.shape[1]):

            output = model(torch.tensor([x1[i][j],x2[i][j]]))


            if(output[0] > 0.5):
                y[i][j] = 1.0
            else:
                y[i][j] = 0.0
    f = plt.figure()
    plt.title("Decision Plot")
    plt.xlabel("x1")
    plt.ylabel("x2")
    plt.contourf(x1,x2,y)
    plt.colorbar()
    plt.savefig(name+"decision.png")
    plt.close(f)


```

```
150
151
152
153 for epoch in epochs:
154
155     model = FFNN()
156     model.load_state_dict(torch.load(f"epoch{str(epoch)}.pt",map_location=device))
157     gen_plots(model,epoch)
158
159
160 create_decision_plot(dir)
```

2.4 model.py

Python script describing PyTorch model

```
1  import torch
2  from torch import nn
3  class FFNN(nn.Module):
4      def __init__(self):
5          super().__init__()
6          self.hidden1 = nn.Sequential(nn.Linear(in_features=2,out_features=4,bias=True),nn.Tanh())
7          self.hidden2 = nn.Sequential( nn.Linear(in_features=4,out_features=2,bias=True),
8              nn.Tanh())
9          self.output = nn.Sequential(nn.Linear(in_features=2,out_features=1,bias=True),
10             nn.Sigmoid())
11
12
13
14     def forward(self,x,path='all'):
15         if(path=='hidden1'):
16             return self.hidden1(x)
17         if(path=='hidden2'):
18             return self.hidden2(x)
19         if(path=='output'):
20             return self.output(x)
21         if(path=='all'):
22             return self.output(self.hidden2(self.hidden1(x)))
23
24
25
26 def test():
27     print("Testing")
28     model = FFNN()
29     print(model(torch.tensor([8.85218218,1.47735284])))
30
31 if __name__ == "__main__":
32     test()
```

2.5 dataset.py

Python script giving access to the Dataset

```
1  import torch
2  from torch.utils.data import Dataset
3  from torch.utils.data import DataLoader
4  from model import FFNN
5
6
7
8  class Task2aDataset(Dataset):
9      def __init__(self,data_source):
10
11         with open(data_source,"r") as f:
12             self.lines = f.readlines()
13         self.inputs = []
14         self.outputs = []
15         for line in self.lines[1:]:
16
```

```
            self.linedata = tuple(map(float,line.strip().split(",")))
            self.inputs.append(torch.tensor(self.linedata[1:3]))
            self.outputs.append(torch.tensor(self.linedata[3]))


    def __len__(self):
        return len(self.outputs)
    def __getitem__(self,idx):
        return self.inputs[idx],self.outputs[idx]

def test():
    print("Testing datset")
    train_dataset = Task2aDataset(data_source="dataset/train_t25.csv")
    train_dataloader = DataLoader(train_dataset, batch_size=2, shuffle=False)
    #train_features, train_labels = next(iter(train_dataloader))
    for i,data in enumerate(train_dataloader,0):
        if(i==0):
            inputs,labels = data
            model = FFNN()
            print(model(inputs))



if __name__ == '__main__':
    test()
```

2.6 test.py

Python script to generate output for test set data in Task2a

The test set output is attached in this link

```
# Get outputs for test dataset
import torch
import torch.nn as nn

from model import FFNN

device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")

model = FFNN()
model.load_state_dict(torch.load("weights.pt",map_location=device))

lines = []
with open("dataset/test_t25.csv","r") as f:
    lines  = f.readlines()
outputs = []
for line in lines[1:]:
    c = torch.tensor(list(map(float,line.strip().split(",")))[1:])
    print(c)
    outputs.append(model(c).item())

with open("test_output.csv","w") as f:
    f.write(lines[0].strip()+",label\n")
    for i,output in enumerate(outputs):
        if(output>0.5):
            output = 1.0
        else:
            output = 0.0
        f.write(f"{lines[i+1].strip()},{output}\n")
```

3 Task 2b

3.1 train.py

Python script for training the model

```
import torch
import argparse
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import DataLoader
from dataset import train_dataset, val_dataset
from model import Image_Classification, weights_init
import matplotlib.pyplot as plt


parser = argparse.ArgumentParser()
parser.add_argument("weight_update", help="Enter the weight update rule", type=str)
args = parser.parse_args()


#device = 'cuda' if torch.cuda.is_available() else 'cpu'
device = 'cpu'
batch_size = 1
learning_rate = 2e-5
epochs = 150
weight_update = args.weight_update


train_loader = DataLoader(dataset=train_dataset, batch_size=batch_size, shuffle=True)
val_loader = DataLoader(dataset=val_dataset, batch_size=batch_size, shuffle=True)



model = Image_Classification().to(device=device)
model.apply(weights_init)

criterion = nn.CrossEntropyLoss()

if weight_update=='generalized_delta':
    optimizer = optim.SGD(model.parameters(), lr=learning_rate, momentum=0.9)

if weight_update=='delta':
    optimizer = optim.SGD(model.parameters(), lr=learning_rate)

if weight_update=='adam':
    optimizer = optim.Adam(model.parameters(), lr=learning_rate)


avg_train_losses = list()
avg_val_losses = list()
for epoch in range(epochs):
    count = 0
    avg_train_loss = 0

    model.train()
    for batch_idx, (image, label) in enumerate(train_loader):
        count += 1

        image = image.to(device=device)
        label = label.to(device=device)

        out = model(image)
        loss = criterion(out, label.long())

        avg_train_loss += loss.item()

        optimizer.zero_grad()
        loss.backward()

        optimizer.step()

    avg_train_loss = avg_train_loss/count
    avg_train_losses.append(avg_train_loss)


    count = 0
```

```
    avg_val_loss = 0
    model.eval()
    for batch_idx, (image, label) in enumerate(val_loader):
        with torch.no_grad():
            count += 1

            image = image.to(device=device)
            label = label.to(device=device)

            out = model(image)
            loss = criterion(out, label.long())

            avg_val_loss += loss.item()


    avg_val_loss = avg_val_loss/count
    avg_val_losses.append(avg_val_loss)
    print(f'Epochs:{epoch+1}, Average Train Loss:{avg_train_loss}, Average Validation Loss:{
    avg_val_loss}')

    if ((epoch+1)%10==0):
        if weight_update=='generalized_delta':
            torch.save(model.state_dict(), 'model_weights_generalized_delta.pth')

        if weight_update=='delta':
            torch.save(model.state_dict(), 'model_weights_delta.pth')

        if weight_update=='adam':
            torch.save(model.state_dict(), 'model_weights_adam.pth')




plt.figure()
plt.plot(list(range(1,epochs+1)), avg_train_losses, 'b', label="Train Loss")
plt.plot(list(range(1,epochs+1)), avg_val_losses, 'r', label="Validation Loss")
plt.xlabel('Epochs')
plt.ylabel('Average Loss')
plt.title("Average Loss v/s Epoch")
plt.legend()
plt.savefig(f"plots/{weight_update}.png")
plt.show()
```

3.2 eval.py

Python script for generating metrics

```
import torch
from torch.utils.data import DataLoader
from model import Image_Classification
from dataset import train_dataset, val_dataset#, test_dataset
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
import matplotlib.pyplot as plt
import argparse


parser = argparse.ArgumentParser()
parser.add_argument("weight_update", help="Enter the weight update rule", type=str)
args = parser.parse_args()


weight_update = args.weight_update
#device = 'cuda' if torch.cuda.is_available() else 'cpu'
device = 'cpu'

train_loader = DataLoader(dataset=train_dataset)
val_loader = DataLoader(dataset=val_dataset)
#test_loader = DataLoader(dataset=test_dataset)


```

```
model = Image_Classification().to(device=device)

if weight_update=='generalized_delta':
    model.load_state_dict(torch.load('model_weights_generalized_delta.pth'))

if weight_update=='delta':
    model.load_state_dict(torch.load('model_weights_delta.pth'))

if weight_update=='adam':
    model.load_state_dict(torch.load('model_weights_adam.pth'))


def accuracy(loader, model):
    y_pred = []
    y = []

    print("-----------------------------------------------------")

    num_correct_labels = 0
    num_labels = 0

    with torch.no_grad():
        for image, label in loader:
            model.eval()
            image = image.to(device=device)
            label = label.to(device=device)

            out = model(image)
            preds = torch.argmax(out)
            y_pred.append(preds.item())
            y.append(int(label.item()))
            num_correct_labels += (preds==label)
            num_labels += 1

        print(f"Accuracy of the model is {float(num_correct_labels/num_labels)*100:.2f} %")
        print("-------------------------------------------------------------------------------")
        print('')

    model.train()

    return (y_pred, y)




print("Checking accuracy on train data..")
y_pred_train, y_train = accuracy(train_loader, model)

print("Checking accuracy on validation data..")
y_pred_val, y_val = accuracy(val_loader, model)

# print(len(y_pred_train), len(y_train))
# print(y_pred_train, y_train)

cm_train = confusion_matrix(y_train, y_pred_train)
disp1 = ConfusionMatrixDisplay(confusion_matrix=cm_train ,display_labels=['coast', 'forest', '
    opencountry', 'street', 'tallbuilding'])
disp1.plot()
plt.savefig(f"plots/cm_train_{weight_update}.png")
plt.show()


cm_val = confusion_matrix(y_val, y_pred_val)
disp2 = ConfusionMatrixDisplay(confusion_matrix=cm_val ,display_labels=['coast', 'forest','
    opencountry', 'street', 'tallbuilding'])
disp2.plot()
plt.savefig(f"plots/cm_val_{weight_update}.png")
plt.show()


#print("Checking accuracy on test data..")
```

```
#accuracy(test_loader, model)
```

3.3 model.py

Python script describing PyTorch model

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

class Image_Classification(nn.Module):
    def __init__(self, l1=256, l2=8):
        super(Image_Classification, self).__init__()

        self.linear1 = nn.Linear(in_features=60, out_features=l1)
        self.linear2 = nn.Linear(in_features=l1, out_features=l2)
        self.linear3 = nn.Linear(in_features=l2, out_features=5)
        self.tanh = nn.Tanh()
        self.softmax = nn.Softmax(dim=0)


    def forward(self, x):

        x = self.tanh(self.linear1(x))
        x = self.tanh(self.linear2(x))
        #x = self.softmax(self.linear3(x))
        x = self.linear3(x)

        return x

def weights_init(m):
    if isinstance(m, nn.Linear):
        torch.manual_seed(768)
        nn.init.xavier_uniform_(m.weight.data)
        nn.init.zeros_(m.bias.data)

def test():
    model = Image_Classification()
    input = torch.randn([60])
    out = model(input)

    print(model)
    print(out)
    print(torch.argmax(out).float())

#test()
```

3.4 dataset.py

Python script giving access to the Dataset

```
from torch.functional import norm
from torch.utils import data
from torch.utils.data import Dataset, random_split
import numpy as np
import torch
import pandas as pd

train_image_dir = 'dataset/image_data_dim60.txt'
train_label_dir = 'dataset/image_data_labels.txt'

class Image_Dataset(Dataset):
    def __init__(self, image_dir, label_dir):
        self.image_dir = image_dir
        self.label_dir = label_dir
        self.image = np.loadtxt(self.image_dir)
        self.label = np.loadtxt(self.label_dir)

```

```
        mask = (self.label==0) | (self.label==1) | (self.label==5) | (self.label==6) | (self.label==7)
        self.label, self.image = self.label[mask], self.image[mask]
        self.label[(self.label==5)], self.label[(self.label==6)], self.label[(self.label==7)] = 2, 3, 4

    def __len__(self):
        return len(self.image)

    def __getitem__(self, index):
        img_index = torch.from_numpy(self.image[index].astype(np.float32))
        label_index = torch.tensor(self.label[index])

        return (img_index, label_index)


dataset = Image_Dataset(image_dir=train_image_dir, label_dir=train_label_dir)

# 70-20-10 train-dev-test split
# train_size = int(0.7*len(dataset))
# val_size = int(0.2*len(dataset))
# test_size = len(dataset) - train_size - val_size
# train_dataset, val_dataset, test_dataset = random_split(dataset, [train_size, val_size, test_size], generator=torch.Generator().manual_seed(42))

train_size = int(0.8*len(dataset))
val_size =  len(dataset) - train_size
train_dataset, val_dataset = random_split(dataset, [train_size, val_size], generator=torch.Generator().manual_seed(42))

# scaler = StandardScaler()
# scaler.fit(train_dataset[:][0])

# train_dataset[:][0] = torch.tensor(scaler.transform(train_dataset[:][0].item()))
# val_dataset[:][0] = torch.tensor(scaler.transform(val_dataset[:][0].item()))
# test_dataset[:][0] = torch.tensor(scaler.transform(test_dataset[:][0].item()))




def test():
    dataset = Image_Dataset(image_dir='dataset/image_data_dim60.txt', label_dir='dataset/image_data_labels.txt')
    train_image, train_label = dataset[567]

    print(dataset[:][0].shape, dataset[:][1].shape)
    print(pd.DataFrame(dataset[:][1]).value_counts())
    print(train_image.shape)
    print(train_label)


#test()
```