

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

```
1 import torch
2 import torch.nn as nn
3 from torch.utils.data import DataLoader
4 import torch.optim as optim
5 from dataset import function_dataset
6 from model import function_approximation
7
8 data_dir = 'dataset/train.csv'
9 device = 'cuda' if torch.cuda.is_available() else 'cpu'
10 batch_size = 1
11 learning_rate = 2e-6
12 epochs = 350
13 momentum = 0.9
14
15 train_dataset = function_dataset(data_dir)
16 train_loader = DataLoader(dataset=train_dataset, batch_size=batch_size, shuffle=True)
17
18 losses = []
19
20 model = function_approximation().to(device=device)
21
22 criterion = nn.MSELoss()
23 optimizer = optim.SGD(model.parameters(), lr=learning_rate, momentum=momentum)
24
25
26 for epoch in range(epochs):
27     total_loss = 0
28     cnt = 0
29     for batch_idx, (data, target) in enumerate(train_loader):
30         data = data.to(device=device)
31         target = target.to(device=device)
32
33         out = model(data.float())
34         loss = criterion(out, target.float())
35         cnt+=1
36         total_loss += loss.item()
37
38         optimizer.zero_grad()
39         loss.backward()
40
41         optimizer.step()
42     losses.append(total_loss/cnt)
43     print(f'Epochs:{epoch+1}, Loss:{total_loss/cnt}')
44     if(epoch+1 == 1 or epoch+1==2 or epoch+1==10 or epoch+1==50 or epoch+1==epochs):
45         torch.save(model.state_dict(), "epoch"+str(epoch+1)+".pt")
46
47 with open("losses.txt", "w") as f:
```

```

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

```

1.2 eval.py

Python script for generating metrics

```

1 import torch
2 import torch.nn as nn
3 from torch.utils.data import DataLoader
4 from model import function_approximation
5 from dataset import function_dataset
6
7 device = 'cuda' if torch.cuda.is_available() else 'cpu'
8
9 model = function_approximation().to(device=device)
10 model.load_state_dict(torch.load('model_weights.pth'))
11
12 train_dataset = function_dataset("dataset/train.csv")
13 train_loader = DataLoader(train_dataset)
14
15 test_dataset = function_dataset("dataset/test.csv")
16 test_loader = DataLoader(test_dataset)
17
18 valid_dataset = function_dataset("dataset/validation.csv")
19 valid_loader = DataLoader(valid_dataset)
20
21
22 criterion = nn.MSELoss()
23
24
25 def accuracy(loader, model):
26
27
28     avg_loss = 0
29     cnt=0
30
31     with torch.no_grad():
32         for data, target in loader:
33             model.eval()
34             data = data.to(device=device)
35             target = target.to(device=device)
36
37             out = model(data.float())
38
39             loss = criterion(out,target.float())
40             avg_loss += loss
41             cnt+=1
42     avg_loss = avg_loss/cnt
43     print(f"Average loss is: {avg_loss:.2f}")
44     print("-----")
45
46
47 print("Train Set metrics:")
48 accuracy(train_loader, model)
49
50 print("Test Set metrics:")
51 accuracy(test_loader, model)
52
53 print("Validation Set metrics:")
54 accuracy(valid_loader, model)
55

```

1.3 plot.py

Python script for generating plots

```

1 import torch

```

```

2 import torch.nn as nn
3 from torch.utils.data import DataLoader
4 from dataset import function_dataset
5 from model import function_approximation
6 import numpy as np
7 import matplotlib.pyplot as plt
8 from matplotlib import cm
9 import pandas as pd
10
11 df = pd.read_csv('dataset/func_app1.csv')
12
13 dir = "plots/"
14
15 #device = 'cuda' if torch.cuda.is_available() else 'cpu'
16 device = 'cpu'
17
18
19
20 train_dataset = function_dataset("dataset/train.csv")
21 train_loader = DataLoader(train_dataset)
22
23 test_dataset = function_dataset("dataset/test.csv")
24 test_loader = DataLoader(test_dataset)
25
26 valid_dataset = function_dataset("dataset/validation.csv")
27 valid_loader = DataLoader(valid_dataset)
28
29
30 def gen_plots(model, epoch):
31     x1 = np.arange(0,6,0.25, dtype="float32")
32     x2 = np.arange(0,6,0.25, dtype="float32")
33     x1,x2 = np.meshgrid(x1,x2)
34
35     y = np.zeros(x1.shape)
36
37     for i in range(x1.shape[0]):
38         for j in range(x1.shape[1]):
39
40             output = model(torch.tensor([x1[i][j],x2[i][j]]))
41             y[i][j]= output
42
43     f = plt.figure()
44     ax = plt.axes(projection='3d')
45     surf = ax.plot_surface(x1, x2, y, cmap = cm.jet, linewidth=0, antialiased=False)
46     f.colorbar(surf, shrink=0.5, aspect=10)
47     ax.set_title(f'Approximated Function after {epoch} Epochs')
48     ax.set_xlabel('x1')
49     ax.set_ylabel('x2')
50     plt.savefig(dir+'epoch'+f'{epoch}'+ "_approximated.png")
51     plt.show()
52
53
54 # Plot of loss variation with epoch
55 losses = []
56 with open("losses.txt", "r") as f:
57     lines = f.readlines()
58     for l in lines:
59         losses.append(float(l.strip()))
60
61 epochs = np.arange(1, len(losses)+1, 1)
62 losses = np.array(losses)
63 f = plt.figure()
64 plt.title("Loss v/s Epoch")
65 plt.plot(epochs, losses)
66 plt.xlabel('Epochs')
67 plt.ylabel('Loss')
68 plt.savefig(dir+"loss.png")
69 plt.show()
70 plt.close(f)
71
72

```

```

73 # Scatter plot of desired output v/s approximated output
74 model = function_approximation().to(device=device)
75 model.load_state_dict(torch.load('model_weights.pth'))
76
77 desired = []
78 approximated = []
79
80
81 for batch_idx, (data, target) in enumerate(train_loader):
82     data = data.to(device=device)
83     target = target.to(device=device)
84
85     out = model(data.float())
86
87     approximated.append(out.item())
88     desired.append(target.item())
89 desired = np.array(desired)
90 approximated = np.array(approximated)
91
92
93 f = plt.figure()
94 plt.title("Desired v/s Approximated Scatter Plot")
95 plt.scatter(desired, approximated, c='b', linewidths=1)
96 plt.plot(desired, desired, 'r')
97 plt.xlabel('Desired Function')
98 plt.ylabel('Approximated Function')
99 plt.savefig(dir+"scatter.png")
100 plt.show()
101 plt.close(f)
102
103
104 f = plt.figure()
105 ax = plt.axes(projection='3d')
106 surf = ax.plot_trisurf(df.iloc[:,0], df.iloc[:,1], df.iloc[:,2], cmap=cm.jet, linewidth=0,
107                        antialiased=False)
108 f.colorbar(surf, shrink=0.5, aspect=10)
109 ax.set_title(f'Desired Function')
110 ax.set_xlabel('x1')
111 ax.set_ylabel('x2')
112 ax.set_zlabel('Desired Function')
113 plt.savefig(dir+"desired.png")
114 plt.show()
115
116 for epoch in [1,2,10,50,350]:
117     model = function_approximation().to(device)
118     model.load_state_dict(torch.load(f"epoch{str(epoch)}.pt",map_location=device))
119     gen_plots(model,epoch)

```

1.4 model.py

Python script describing PyTorch model

```

1 import torch
2 import torch.nn as nn
3 import torch.nn.functional as F
4
5
6 class function_approximation(nn.Module):
7     def __init__(self):
8         super(function_approximation, self).__init__()
9
10         self.linear1 = nn.Linear(in_features=2, out_features=8, bias=True)
11         self.linear2 = nn.Linear(in_features=8, out_features=4)
12         self.linear3 = nn.Linear(in_features=4, out_features=1)
13         self.tanh = nn.Tanh()
14         self.softmax = nn.Softmax(dim=0)
15
16
17     def forward(self, x):

```

```

18         x = self.tanh(self.linear1(x))
19         x = self.tanh(self.linear2(x))
20         x = self.linear3(x)
21
22         return x
23
24
25 def test():
26     model = function_approximation()
27     input = torch.Tensor([4.321097794848372, 4.769609253163742])
28     out = model(input)
29
30     print(input)
31     print(model)
32     print(out)
33
34 #test()

```

1.5 dataset.py

Python script giving access to the Dataset

```

1 from torch.utils.data import Dataset
2 import pandas as pd
3 import numpy as np
4 import torch
5
6
7 class function_dataset(Dataset):
8     def __init__(self, data_dir):
9         self.data_dir = data_dir
10        data = np.array(pd.read_csv(self.data_dir))
11        self.input_features = data[:,0:2]
12        self.target = data[:,2:]
13        self.len = data.shape[0]
14
15    def __len__(self):
16        return self.len
17
18    def __getitem__(self, index):
19        features_index = torch.from_numpy(self.input_features[index])
20        target_index = torch.from_numpy(self.target[index])
21
22        return (features_index, target_index)
23
24 def test():
25     train_dataset = function_dataset(data_dir='dataset/func_app1.csv')
26     train_data, train_label = train_dataset[5]
27
28     # print(train_data)
29     # print(train_label)
30     # print(train_data.shape)
31     # print(train_label.shape)
32
33     print(train_dataset[:][0][:,0].shape)
34
35 #test()

```

2 Task 2a

2.1 train.py

Python script for training the model

```

1 import torch
2 from dataset import Task2aDataset
3 from model import FFNN
4 import torch.optim as optim

```

```

5 import torch.nn as nn
6
7 criterion = nn.BCELoss()
8
9 device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
10 batch_size = 1
11 epochs = 360
12
13 trainDataset = Task2aDataset("dataset/train_t25.csv")
14 trainLoader = torch.utils.data.DataLoader(trainDataset, batch_size=batch_size,
15                                           shuffle=True)
16 model = FFNN()
17 model.to(device)
18
19 optimizer = optim.SGD(model.parameters(), lr=0.0001, momentum=0.7)
20
21
22
23 print(device)
24
25 for epoch in range(epochs):
26     print("Current Epoch: ", epoch)
27     total_loss = 0
28     cnt = 0
29     for i, data in enumerate(trainLoader, 0):
30         cnt += 1
31         optimizer.zero_grad()
32         inputs, labels = data
33         inputs = inputs.to(device)
34         labels = labels.to(device)
35         # Model
36         h1outputs = model(inputs, path='hidden1').to(device)
37         h12outputs = model(h1outputs, path='hidden2').to(device)
38         outputs = model(h12outputs, path='output')
39
40
41         labels = labels.unsqueeze(1)
42         loss = criterion(outputs, labels)
43         total_loss += loss.item()
44
45
46
47
48
49         loss.backward()
50         optimizer.step()
51     if (epoch+1 == 1 or epoch+1==2 or epoch+1==10 or epoch+1==50 or epoch+1==epochs):
52         torch.save(model.state_dict(), "epoch"+str(epoch+1)+".pt")
53     print("Epoch Average Loss: ", total_loss/(cnt))
54
55 torch.save(model.state_dict(), "weights.pt")

```

2.2 eval.py

Python script for generating metrics

```

1 import torch
2 import torch.nn as nn
3
4 from model import FFNN
5 from dataset import Task2aDataset
6 device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
7
8 model = FFNN()
9 model.load_state_dict(torch.load("weights.pt", map_location=device))
10
11
12 trainSet = Task2aDataset("dataset/train_t25.csv")
13
14 trainLoader = torch.utils.data.DataLoader(trainSet, batch_size=1,

```

```

15         shuffle=True)
16
17 devSet = Task2aDataset("dataset/dev_t25.csv")
18
19 devLoader = torch.utils.data.DataLoader(devSet, batch_size=1,
20                                         shuffle=True)
21
22 criterion = nn.BCELoss()
23 def get_metrics(loader):
24     TP = 0
25     TN = 0
26     FP = 0
27     FN = 0
28
29     total_loss = 0
30     cnt = 0
31     for i,data in enumerate(loader):
32         cnt+=1
33         inputs,labels = data
34         outputs = model(inputs.to(device))
35         labels = labels.unsqueeze(1)
36         loss = criterion(outputs,labels)
37
38         total_loss+=loss.item()
39         if(outputs[0][0] > 0.5):
40             if(labels[0][0] == 1.0):
41                 TP+=1
42             else:
43                 FP+=1
44         else:
45             if(labels[0][0] == 0.0):
46                 TN+=1
47             else:
48                 FN+=1
49     accuracy = (TP+TN)/(FP+FN+TP+TN)
50     precision=TP/(TP+FP)
51     recall=TP/(TP+FN)
52
53
54     print("Average Loss: ",total_loss/cnt)
55     print("TP:",TP)
56     print("TN:",TN)
57     print("FP:",FP)
58     print("FN:",FN)
59
60     print("Accuracy: ",accuracy)
61     print("Precision: ", precision)
62     print("Recall: ", recall)
63     print("F1 score:",2/(1/recall + 1/precision))
64
65
66 print("==Train Set Metrics==")
67 get_metrics(trainLoader)
68
69
70
71 print("==Dev Set Metrics==")
72 get_metrics(devLoader)

```

2.3 plot.py

Python script for generating plots

```

1 import matplotlib.pyplot as plt
2 import torch.nn as nn
3 import torch
4 import numpy as np
5 from model import FFNN
6 from matplotlib import cm
7

```

```

8 from dataset import Task2aDataset
9
10 dir = "plots/"
11 epochs = [1,2,10,50,360]
12 device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
13
14 def gen_plots(model, epoch):
15     name = dir + "epoch" + str(epoch)
16     x1 = np.arange(-20,20,0.25, dtype="float32")
17     x2 = np.arange(-20,20,0.25, dtype="float32")
18     x1,x2 = np.meshgrid(x1,x2)
19
20
21
22     hl11 = np.zeros(x1.shape)
23     hl12 = np.zeros(x1.shape)
24     hl13 = np.zeros(x1.shape)
25     hl14 = np.zeros(x1.shape)
26     hl21 = np.zeros(x1.shape)
27     hl22 = np.zeros(x1.shape)
28     y = np.zeros(x1.shape)
29
30
31     for i in range(x1.shape[0]):
32         for j in range(x1.shape[1]):
33
34             hl1 = model(torch.tensor([x1[i][j],x2[i][j]]),path="hidden1")
35             hl2 = model(hl1,path="hidden2")
36             output = model(hl2,path="output")
37
38             #print(hl1)
39
40             hl11[i][j] = hl1[0]
41             hl12[i][j]= hl1[1]
42             hl13[i][j]= hl1[2]
43             hl14[i][j]= hl1[3]
44
45             hl21[i][j]= hl2[0]
46             hl22[i][j]= hl2[1]
47
48             y[i][j]= output[0]
49
50     f = plt.figure()
51     fig, ax = plt.subplots(subplot_kw={"projection": "3d"})
52     surf = ax.plot_surface(x1, x2, hl11, cmap=cm.coolwarm,linewidth=0, antialiased=False)
53     ax.set_title(f"HL 1, Node 1 (epoch: {epoch} )")
54     ax.set_xlabel("x1")
55     ax.set_ylabel("x2")
56     ax.set_zlabel("Value")
57
58     plt.savefig(name+"_hl11.png")
59     plt.close(f)
60
61     f = plt.figure()
62     fig, ax = plt.subplots(subplot_kw={"projection": "3d"})
63     surf = ax.plot_surface(x1, x2, hl12, cmap=cm.coolwarm,linewidth=0, antialiased=False)
64     ax.set_title(f"HL 1, Node 2 (epoch: {epoch} )")
65     ax.set_xlabel("x1")
66     ax.set_ylabel("x2")
67     ax.set_zlabel("Value")
68     plt.savefig(name+"_hl12.png")
69     plt.close(f)
70
71     f = plt.figure()
72     fig, ax = plt.subplots(subplot_kw={"projection": "3d"})
73     surf = ax.plot_surface(x1, x2, hl13, cmap=cm.coolwarm,linewidth=0, antialiased=False)
74     ax.set_title(f"HL 1, Node 3 (epoch: {epoch} )")
75     ax.set_xlabel("x1")
76     ax.set_ylabel("x2")
77     ax.set_zlabel("Value")
78     plt.savefig(name+"_hl13.png")

```



```

79 plt.close(f)
80
81 f = plt.figure()
82 fig, ax = plt.subplots(subplot_kw={"projection": "3d"})
83 surf = ax.plot_surface(x1, x2, hl14, cmap=cm.coolwarm,linewidth=0, antialiased=False)
84 ax.set_title(f"HL 1, Node 4 (epoch: {epoch} )")
85 ax.set_xlabel("x1")
86 ax.set_ylabel("x2")
87 ax.set_zlabel("Value")
88 plt.savefig(name+"_hl14.png")
89 plt.close(f)
90
91 f = plt.figure()
92 fig, ax = plt.subplots(subplot_kw={"projection": "3d"})
93 surf = ax.plot_surface(x1, x2, hl21, cmap=cm.coolwarm,linewidth=0, antialiased=False)
94 ax.set_title(f"HL 2, Node 1 (epoch: {epoch} )")
95 ax.set_xlabel("x1")
96 ax.set_ylabel("x2")
97 ax.set_zlabel("Value")
98 plt.savefig(name+"_hl21.png")
99 plt.close(f)
100
101 f = plt.figure()
102 fig, ax = plt.subplots(subplot_kw={"projection": "3d"})
103 surf = ax.plot_surface(x1, x2, hl22, cmap=cm.coolwarm,linewidth=0, antialiased=False)
104 ax.set_title(f"HL 2, Node 2 (epoch: {epoch} )")
105 ax.set_xlabel("x1")
106 ax.set_ylabel("x2")
107 ax.set_zlabel("Value")
108 plt.savefig(name+"_hl22.png")
109 plt.close(f)
110
111 f = plt.figure()
112 fig, ax = plt.subplots(subplot_kw={"projection": "3d"})
113 surf = ax.plot_surface(x1, x2, y, cmap=cm.coolwarm,linewidth=0, antialiased=False)
114 ax.set_title(f"Output (epoch: {epoch} )")
115 ax.set_xlabel("x1")
116 ax.set_ylabel("x2")
117 ax.set_zlabel("Value")
118 plt.savefig(name+"_output.png")
119 plt.close(f)
120
121
122 def create_decision_plot(name):
123     model = FFNN()
124     model.load_state_dict(torch.load(f"weights.pt",map_location=device))
125
126     x1 = np.arange(-20,20,0.25, dtype="float32")
127     x2 = np.arange(-20,20,0.25, dtype="float32")
128     x1,x2 = np.meshgrid(x1,x2)
129     y = np.zeros(x1.shape)
130     for i in range(x1.shape[0]):
131         for j in range(x1.shape[1]):
132
133             output = model(torch.tensor([x1[i][j],x2[i][j]]))
134
135
136             if(output[0] > 0.5):
137                 y[i][j] = 1.0
138             else:
139                 y[i][j] = 0.0
140     f = plt.figure()
141     plt.title("Decision Plot")
142     plt.xlabel("x1")
143     plt.ylabel("x2")
144     plt.contourf(x1,x2,y)
145     plt.colorbar()
146     plt.savefig(name+"decision.png")
147     plt.close(f)
148
149

```

```

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

```

```

17         self.linedata = tuple(map(float,line.strip().split(",")))
18         self.inputs.append(torch.tensor(self.linedata[1:3]))
19         self.outputs.append(torch.tensor(self.linedata[3]))
20
21
22     def __len__(self):
23         return len(self.outputs)
24     def __getitem__(self,idx):
25         return self.inputs[idx],self.outputs[idx]
26
27 def test():
28     print("Testing dataset")
29     train_dataset = Task2aDataset(data_source="dataset/train_t25.csv")
30     train_dataloader = DataLoader(train_dataset, batch_size=2, shuffle=False)
31     #train_features, train_labels = next(iter(train_dataloader))
32     for i,data in enumerate(train_dataloader,0):
33         if(i==0):
34             inputs,labels = data
35             model = FFNN()
36             print(model(inputs))
37
38
39
40
41 if __name__ == '__main__':
42     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](#)

```

1 # Get outputs for test dataset
2 import torch
3 import torch.nn as nn
4
5 from model import FFNN
6
7 device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
8
9 model = FFNN()
10 model.load_state_dict(torch.load("weights.pt",map_location=device))
11
12 lines = []
13 with open("dataset/test_t25.csv","r") as f:
14     lines = f.readlines()
15 outputs = []
16 for line in lines[1:]:
17     c = torch.tensor(list(map(float,line.strip().split(",")))[1:])
18     print(c)
19     outputs.append(model(c).item())
20
21 with open("test_output.csv","w") as f:
22     f.write(lines[0].strip()+"label\n")
23     for i,output in enumerate(outputs):
24         if(output>0.5):
25             output = 1.0
26         else:
27             output = 0.0
28         f.write(f"{lines[i+1].strip()},{output}\n")

```

3 Task 2b

3.1 train.py

Python script for training the model

```

1 import torch
2 import argparse
3 import torch.nn as nn
4 import torch.optim as optim
5 from torch.utils.data import DataLoader
6 from dataset import train_dataset, val_dataset
7 from model import Image_Classification, weights_init
8 import matplotlib.pyplot as plt
9
10
11 parser = argparse.ArgumentParser()
12 parser.add_argument("weight_update", help="Enter the weight update rule", type=str)
13 args = parser.parse_args()
14
15
16 #device = 'cuda' if torch.cuda.is_available() else 'cpu'
17 device = 'cpu'
18 batch_size = 1
19 learning_rate = 2e-5
20 epochs = 150
21 weight_update = args.weight_update
22
23
24 train_loader = DataLoader(dataset=train_dataset, batch_size=batch_size, shuffle=True)
25 val_loader = DataLoader(dataset=val_dataset, batch_size=batch_size, shuffle=True)
26
27
28
29 model = Image_Classification().to(device=device)
30 model.apply(weights_init)
31
32 criterion = nn.CrossEntropyLoss()
33
34 if weight_update=='generalized_delta':
35     optimizer = optim.SGD(model.parameters(), lr=learning_rate, momentum=0.9)
36
37 if weight_update=='delta':
38     optimizer = optim.SGD(model.parameters(), lr=learning_rate)
39
40 if weight_update=='adam':
41     optimizer = optim.Adam(model.parameters(), lr=learning_rate)
42
43
44 avg_train_losses = list()
45 avg_val_losses = list()
46 for epoch in range(epochs):
47     count = 0
48     avg_train_loss = 0
49
50     model.train()
51     for batch_idx, (image, label) in enumerate(train_loader):
52         count += 1
53
54         image = image.to(device=device)
55         label = label.to(device=device)
56
57         out = model(image)
58         loss = criterion(out, label.long())
59
60         avg_train_loss += loss.item()
61
62         optimizer.zero_grad()
63         loss.backward()
64
65         optimizer.step()
66
67     avg_train_loss = avg_train_loss/count
68     avg_train_losses.append(avg_train_loss)
69
70
71     count = 0

```

```

72     avg_val_loss = 0
73     model.eval()
74     for batch_idx, (image, label) in enumerate(val_loader):
75         with torch.no_grad():
76             count += 1
77
78             image = image.to(device=device)
79             label = label.to(device=device)
80
81             out = model(image)
82             loss = criterion(out, label.long())
83
84             avg_val_loss += loss.item()
85
86
87     avg_val_loss = avg_val_loss / count
88     avg_val_losses.append(avg_val_loss)
89     print(f'Epochs:{epoch+1}, Average Train Loss:{avg_train_loss}, Average Validation Loss:{avg_val_loss}')
90
91     if ((epoch+1)%10==0):
92         if weight_update=='generalized_delta':
93             torch.save(model.state_dict(), 'model_weights_generalized_delta.pth')
94
95         if weight_update=='delta':
96             torch.save(model.state_dict(), 'model_weights_delta.pth')
97
98         if weight_update=='adam':
99             torch.save(model.state_dict(), 'model_weights_adam.pth')
100
101
102
103
104 plt.figure()
105 plt.plot(list(range(1, epochs+1)), avg_train_losses, 'b', label="Train Loss")
106 plt.plot(list(range(1, epochs+1)), avg_val_losses, 'r', label="Validation Loss")
107 plt.xlabel('Epochs')
108 plt.ylabel('Average Loss')
109 plt.title("Average Loss v/s Epoch")
110 plt.legend()
111 plt.savefig(f"plots/{weight_update}.png")
112 plt.show()

```

3.2 eval.py

Python script for generating metrics

```

1  import torch
2  from torch.utils.data import DataLoader
3  from model import Image_Classification
4  from dataset import train_dataset, val_dataset#, test_dataset
5  from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
6  import matplotlib.pyplot as plt
7  import argparse
8
9
10 parser = argparse.ArgumentParser()
11 parser.add_argument("weight_update", help="Enter the weight update rule", type=str)
12 args = parser.parse_args()
13
14
15 weight_update = args.weight_update
16 #device = 'cuda' if torch.cuda.is_available() else 'cpu'
17 device = 'cpu'
18
19 train_loader = DataLoader(dataset=train_dataset)
20 val_loader = DataLoader(dataset=val_dataset)
21 #test_loader = DataLoader(dataset=test_dataset)
22
23

```

```

24 model = Image_Classification().to(device=device)
25
26 if weight_update=='generalized_delta':
27     model.load_state_dict(torch.load('model_weights_generalized_delta.pth'))
28
29 if weight_update=='delta':
30     model.load_state_dict(torch.load('model_weights_delta.pth'))
31
32 if weight_update=='adam':
33     model.load_state_dict(torch.load('model_weights_adam.pth'))
34
35
36 def accuracy(loader, model):
37     y_pred = []
38     y = []
39
40     print("-----")
41
42     num_correct_labels = 0
43     num_labels = 0
44
45     with torch.no_grad():
46         for image, label in loader:
47             model.eval()
48             image = image.to(device=device)
49             label = label.to(device=device)
50
51             out = model(image)
52             preds = torch.argmax(out)
53             y_pred.append(preds.item())
54             y.append(int(label.item()))
55             num_correct_labels += (preds==label)
56             num_labels += 1
57
58             print(f"Accuracy of the model is {float(num_correct_labels/num_labels)*100:.2f} %")
59             print("-----")
60             print('')
61
62     model.train()
63
64     return (y_pred, y)
65
66
67
68
69 print("Checking accuracy on train data..")
70 y_pred_train, y_train = accuracy(train_loader, model)
71
72 print("Checking accuracy on validation data..")
73 y_pred_val, y_val = accuracy(val_loader, model)
74
75 # print(len(y_pred_train), len(y_train))
76 # print(y_pred_train, y_train)
77
78 cm_train = confusion_matrix(y_train, y_pred_train)
79 disp1 = ConfusionMatrixDisplay(confusion_matrix=cm_train ,display_labels=['coast', 'forest', '
    opencountry', 'street', 'tallbuilding'])
80 disp1.plot()
81 plt.savefig(f"plots/cm_train_{weight_update}.png")
82 plt.show()
83
84
85 cm_val = confusion_matrix(y_val, y_pred_val)
86 disp2 = ConfusionMatrixDisplay(confusion_matrix=cm_val ,display_labels=['coast', 'forest', '
    opencountry', 'street', 'tallbuilding'])
87 disp2.plot()
88 plt.savefig(f"plots/cm_val_{weight_update}.png")
89 plt.show()
90
91
92 #print("Checking accuracy on test data..")

```

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

3.3 model.py

Python script describing PyTorch model

```
1 import torch
2 import torch.nn as nn
3 import torch.nn.functional as F
4
5 class Image_Classification(nn.Module):
6     def __init__(self, l1=256, l2=8):
7         super(Image_Classification, self).__init__()
8
9         self.linear1 = nn.Linear(in_features=60, out_features=l1)
10        self.linear2 = nn.Linear(in_features=l1, out_features=l2)
11        self.linear3 = nn.Linear(in_features=l2, out_features=5)
12        self.tanh = nn.Tanh()
13        self.softmax = nn.Softmax(dim=0)
14
15
16    def forward(self, x):
17
18        x = self.tanh(self.linear1(x))
19        x = self.tanh(self.linear2(x))
20        #x = self.softmax(self.linear3(x))
21        x = self.linear3(x)
22
23        return x
24
25    def weights_init(m):
26        if isinstance(m, nn.Linear):
27            torch.manual_seed(768)
28            nn.init.xavier_uniform_(m.weight.data)
29            nn.init.zeros_(m.bias.data)
30
31    def test():
32        model = Image_Classification()
33        input = torch.randn([60])
34        out = model(input)
35
36        print(model)
37        print(out)
38        print(torch.argmax(out).float())
39
40    #test()
```

3.4 dataset.py

Python script giving access to the Dataset

```
1 from torch.functional import norm
2 from torch.utils import data
3 from torch.utils.data import Dataset, random_split
4 import numpy as np
5 import torch
6 import pandas as pd
7
8 train_image_dir = 'dataset/image_data_dim60.txt'
9 train_label_dir = 'dataset/image_data_labels.txt'
10
11 class Image_Dataset(Dataset):
12     def __init__(self, image_dir, label_dir):
13         self.image_dir = image_dir
14         self.label_dir = label_dir
15         self.image = np.loadtxt(self.image_dir)
16         self.label = np.loadtxt(self.label_dir)
17
```

```

18     mask = (self.label==0) | (self.label==1) | (self.label==5) | (self.label==6) | (self.
    label==7)
19     self.label, self.image = self.label[mask], self.image[mask]
20     self.label[(self.label==5)], self.label[(self.label==6)], self.label[(self.label==7)] =
    2, 3, 4
21
22     def __len__(self):
23         return len(self.image)
24
25     def __getitem__(self, index):
26         img_index = torch.from_numpy(self.image[index].astype(np.float32))
27         label_index = torch.tensor(self.label[index])
28
29         return (img_index, label_index)
30
31
32 dataset = Image_Dataset(image_dir=train_image_dir, label_dir=train_label_dir)
33
34 # 70-20-10 train-dev-test split
35 # train_size = int(0.7*len(dataset))
36 # val_size = int(0.2*len(dataset))
37 # test_size = len(dataset) - train_size - val_size
38 # train_dataset, val_dataset, test_dataset = random_split(dataset, [train_size, val_size,
    test_size], generator=torch.Generator().manual_seed(42))
39
40 train_size = int(0.8*len(dataset))
41 val_size = len(dataset) - train_size
42 train_dataset, val_dataset = random_split(dataset, [train_size, val_size], generator=torch.
    Generator().manual_seed(42))
43
44 # scaler = StandardScaler()
45 # scaler.fit(train_dataset[:,0])
46
47 # train_dataset[:,0] = torch.tensor(scaler.transform(train_dataset[:,0].item()))
48 # val_dataset[:,0] = torch.tensor(scaler.transform(val_dataset[:,0].item()))
49 # test_dataset[:,0] = torch.tensor(scaler.transform(test_dataset[:,0].item()))
50
51
52
53
54
55 def test():
56     dataset = Image_Dataset(image_dir='dataset/image_data_dim60.txt', label_dir='dataset/
    image_data_labels.txt')
57     train_image, train_label = dataset[567]
58
59     print(dataset[:,0].shape, dataset[:,1].shape)
60     print(pd.DataFrame(dataset[:,1]).value_counts())
61     print(train_image.shape)
62     print(train_label)
63
64
65 #test()

```