```
1 # -*- coding: utf-8 -*-
 2 """PyTorch_Net.ipynb
 4 Automatically generated by Colaboratory.
 6 Original file is located at
       https://colab.research.google.com/drive/1VaRJXCTRYPnSrfmQcq6PVJuMZ5xxB250
 9 <center><h1>Mini Project 3 - Convolutional Neural Network</h1>
10 <h4>The PyTorch File.</h4></center>
11
12 <h3>Team Members:</h3>
13 <center>
14 Yi Zhu, 260716006<br>
15 Fei Peng, 260712440<br>
16 Yukai Zhang, 260710915
17 </center>
18 """
19
20 # Commented out IPython magic to ensure Python compatibility.
21 from google.colab import drive
22 drive.mount("/content/drive")
23
24 # %cd '/content/drive/My Drive/ECSE_551_F_2020/Mini_Project_03/'
25
26 import numpy as np
27 import pandas as pd
28 import torch
29 import torchvision
30 import torchvision.transforms as transforms
31 import matplotlib.pyplot as plt
32 import pickle
33
34 from torch.utils.data import Dataset
35 from torch.utils.data import DataLoader
36 from PIL import Image
37
38 import torch.nn as nn
39 import torch.nn.functional as F
40
41 import torch.optim as optim
42
43 import torchvision.models as models
44
45 class MyDataset(Dataset):
       def __init__(self, img_file, label_file, transform=None, idx = None):
46
47
           self.data = pickle.load( open( img_file, 'rb' ), encoding='bytes')
           self.targets = np.genfromtxt(label_file, delimiter=',', skip_header=1
48
   )[:,1:]
49
           if idx is not None:
50
             self.targets = self.targets[idx]
51
             self.data = self.data[idx]
52
           self.transform = transform
53
           self.targets -= 5
54
55
       def __len__(self):
56
           return len(self.targets)
57
58
       def __getitem__(self, index):
59
           img, target = self.data[index], int(self.targets[index])
60
           img = Image.fromarray(img.astype('uint8'), mode='L')
61
           if self.transform is not None:
62
```

```
ima = self.transform(ima)
 64
 65
            return img, target
 66
 67 img_transform = transforms.Compose([
 68
        transforms.ToTensor(),
        transforms.Normalize((0.5,),(0.5,))
 69
 70 ])
 71
 72 train_index = np.arange(50000)
 73 test_index = np.arange(50000, 60000)
 74 batch_size = 32 #feel free to change it
 75
 76 # Read image data and their label into a Dataset class
 77 train_set = MyDataset('./Train.pkl', './TrainLabels.csv', transform=
    img_transform, idx=train_index)
 78 train_loader = DataLoader(train_set, batch_size=batch_size, shuffle=True,
    num_workers=2)
 79 test_set = MyDataset('./Train.pkl', './TrainLabels.csv', transform=
    img_transform, idx=None)
 80 test_loader = DataLoader(test_set, batch_size=batch_size, shuffle=True,
    num_workers=2)
 81
 82 """# Notice: In case the code blocks are not in the right order, please run
    the blocks containing train() and test() functions every time after [model,
    criterion and optimizer] definition and before calling them.
 83
 84 # ResNet18
 85 """
 86
 87 resnet18 = models.resnet18()
 88
 89 \text{ net} = \text{resnet18}
 90 net.conv1 = nn.Conv2d(1, 64, kernel_size=(7, 7), stride=(2, 2), padding=(3, 3)
    ), bias=False)
 91 net.fc = nn.Linear(512, 9)
 92 # if there is a available cuda device, use GPU, else, use CPU
 93 device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
 94 net = net.to(device)
 96 # set criterion to cross entropy loss
 97 criterion = nn.CrossEntropyLoss()
 98
 99 # set learning rate to 0.001
100 optimizer = optim.SGD(net.parameters(), lr=0.001)
101 # optimizer = optim.SGD(net.parameters(), lr=0.01, momentum=0.5)
102
103 \text{ epoch} = 32
104 train(epoch)
105
106 test()
107
108 """# AlexNet"""
109
110 alexnet = models.alexnet()
111
112 \text{ net} = \text{alexnet}
113 net.features[0] = nn.Conv2d(1, 64, kernel_size=(11, 11), stride=(4, 4),
    padding=(2, 2))
114 net.classifier[6] = nn.Linear(in_features=4096, out_features=9, bias=True)
115 # if there is a available cuda device, use GPU, else, use CPU
116 device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
117 net = net.to(device)
```

```
118
119 # set criterion to cross entropy loss
120 criterion = nn.CrossEntropyLoss()
122 # set learning rate to 0.001
123 # optimizer = optim.SGD(net.parameters(), lr=0.001)
124 optimizer = optim.SGD(net.parameters(), lr=0.001, momentum=0.8)
126 \text{ epoch} = 32
127 train(epoch)
128
129 test()
130
131 """# VGG16"""
132
133 vgg16 = models.vgg16()
134
135 \text{ net} = \text{vgg}16
136 net.features[0] = nn.Conv2d(1, 64, kernel_size=(3, 3), stride=(1, 1), padding
    =(1, 1)
137 net.classifier[6] = nn.Linear(in_features=4096, out_features=9, bias=True)
138 # if there is a available cuda device, use GPU, else, use CPU
139 device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
140 net = net.to(device)
141
142 # set criterion to cross entropy loss
143 criterion = nn.CrossEntropyLoss()
144
145 # set learning rate to 0.001
146 optimizer = optim.SGD(net.parameters(), lr=0.001)
147 # optimizer = optim.SGD(net.parameters(), lr=0.001, momentum=0.8)
148
149 \ \text{epoch} = 32
150 train(epoch)
151
152 test()
153
154 """# VGG19"""
155
156 vgg19 = models.vgg19()
157
158 \text{ net} = vqq19
159 net.features[0] = nn.Conv2d(1, 64, \text{kernel\_size}=(3, 3), \text{stride}=(1, 1), padding}
    =(1, 1)
160 net.classifier[6] = nn.Linear(in_features=4096, out_features=9, bias=True)
161 # if there is a available cuda device, use GPU, else, use CPU
162 device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
163 net = net.to(device)
164
165 # set criterion to cross entropy loss
166 criterion = nn.CrossEntropyLoss()
167
168 # set learning rate to 0.001
169 # optimizer = optim.SGD(net.parameters(), lr=0.001)
170 optimizer = optim.SGD(net.parameters(), lr=0.005, momentum=0.5)
172 # with: optimizer = optim.SGD(net.parameters(), lr=0.001)
173 \text{ epoch} = 32
174 train(epoch)
175
176 test()
177
178 # with: optimizer = optim.SGD(net.parameters(), lr=0.005, momentum=0.5)
```

```
179 \ \text{epoch} = 32
180 train(epoch)
181
182 test()
183
184 """# VGG19-bn"""
186 vgg19bn = models.vgg19_bn()
187
188 \text{ net} = \text{vgg19bn}
189 net.features[0] = nn.Conv2d(1, 64, kernel_size=(3, 3), stride=(1, 1), padding
    =(1, 1)
190 net.classifier[6] = nn.Linear(in_features=4096, out_features=9, bias=True)
191 # if there is a available cuda device, use GPU, else, use CPU
192 device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
193 net = net.to(device)
194
195 # set criterion to cross entropy loss
196 criterion = nn.CrossEntropyLoss()
197
198 # set learning rate to 0.001
199 # optimizer = optim.SGD(net.parameters(), lr=0.001)
200 optimizer = optim.SGD(net.parameters(), lr=0.005, momentum=0.5)
201
202 def train(num_epochs=2): # Feel free to change it
203
        net.train()
204
205
        running_loss = 0.0
206
207
        # Here is a piece of code that reads data in batch.
208
        # In each epoch all samples are read in batches using dataloader
209
        for epoch in range(num_epochs):
            for i, data in enumerate(train_loader):
210
211
                 img, label = data
212
213
                 img = img.to(device)
214
                 label = label.to(device)
215
216
                 # zero the parameter gradients
217
                 optimizer.zero_grad()
218
219
                 # forward + backward + optimize
220
                 outputs = net(img)
221
222
                loss = criterion(outputs, label)
                 # loss = F.nll_loss(outputs, label)
223
224
                 loss.backward()
225
                 optimizer.step()
226
                 running_loss += loss.item()
227
228
                 if i % 320 == 319: # print every 320 mini-batches
229
                     print('[%d, %5d] loss: %.3f' %
                         (epoch + 1, i + 1, running_loss / 320))
230
231
                     running_loss = 0.0
232
                     torch.save(net.state_dict(), '/model.pth')
233
234
                     torch.save(optimizer.state_dict(), '/optimizer.pth')
235
        print('Finished Training')
236
237
238 def test():
239
        net.eval()
240
```

```
241
        correct = 0
242
        total = 0
243
244
        # calculate accuracy
245
        with torch.no_grad():
246
            for data in test_loader:
247
                 images, labels = data
248
249
                 images = images.to(device)
250
                 labels = labels.to(device)
251
252
                 outputs = net(images)
253
                 # get the index of the max output
                 _, predicted = torch.max(outputs.data, 1)
254
255
                 total += labels.size(0)
256
                 correct += (predicted == labels).sum().item()
257
258
        print('Accuracy of the network: %d %%' % (
259
            100 * correct / total))
260
261 # with: optimizer = optim.SGD(net.parameters(), lr=0.001)
262 \text{ epoch} = 32
263 train(epoch)
264
265 test()
266
267 # with: optimizer = optim.SGD(net.parameters(), lr=0.005, momentum=0.5)
268 \text{ epoch} = 32
269 train(epoch)
270
271 test()
```