

```

1  # -*- coding: utf-8 -*-
2  """CNN.ipynb
3
4  Automatically generated by Colaboratory.
5
6  Original file is located at
7      https://colab.research.google.com/drive/1YiJluZWoyHPQNAVdLCPD7eLf5TLBQHTn
8
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 # Importation and import dataset
20 """
21
22 from google.colab import drive
23 drive.mount('/content/gdrive' )
24
25 # Commented out IPython magic to ensure Python compatibility.
26 # %cd '/content/gdrive/My Drive/ECSE_551_F_2020/Mini_Project_03/'
27 # !ls './data/'
28
29 import pickle
30 import matplotlib.pyplot as plt
31 import numpy as np
32 from torchvision import transforms
33 from torch.utils.data import Dataset
34 from torch.utils.data import DataLoader
35 from keras.layers import Flatten, Dense, Dropout, Activation, Conv2D, MaxPool2D
36 , BatchNormalization
37 from PIL import Image
38 import torch
39
40 # Read a pickle file and display its samples
41 # Note that image data are stored as unit8 so each element is an integer value
42 # between 0 and 255
43 data = pickle.load(open('./Train.pkl', 'rb'), encoding='bytes')
44 targets = np.genfromtxt('./TrainLabels.csv', delimiter=',', skip_header=1)[: ,1
45 :]
46 plt.imshow(data[1234,:,:], cmap='gray', vmin=0, vmax=256)
47 print(data.shape, targets.shape)
48
49 # get train size
50 train_size = data.shape[0]
51
52 # get test size
53 test_data = pickle.load(open('./Test.pkl', 'rb'), encoding='bytes')
54 print(test_data.shape)
55 test_size = test_data.shape[0]
56
57 """# Dataset class
58 *Dataset* class and the *DataLoader* class in pytorch help us to feed our own
59 training data into the network. Dataset class is used to provide an interface
60 for accessing all the training or testing samples in your dataset. For your
61 convinance, we provide you with a custom Dataset that reads the provided data
62 including images (.pkl file) and labels (.csv file).

```

```

57 # DataLoader class
58 Although we can access all the training data using the Dataset class, for
    neural networks, we would need batching, shuffling, multiprocessing data loading
    , etc. DataLoader class helps us to do this. The DataLoader class accepts a
    dataset and other parameters such as batch_size.
59 """
60
61 # Transforms are common image transformations. They can be chained together
    using Compose.
62 # Here we normalize images img=(img-0.5)/0.5
63 img_transform = transforms.Compose([
64     transforms.ToTensor(),
65     transforms.Normalize((0.5,), (0.5,))
66 ])
67
68 # img_file: the pickle file containing the images
69 # label_file: the .csv file containing the labels
70 # transform: We use it for normalizing images (see above)
71 # idx: This is a binary vector that is useful for creating training and
    validation set.
72 # It return only samples where idx is True
73
74 class MyDataset(Dataset):
75     def __init__(self, img_file, label_file, transform=None, idx = None):
76         self.data = pickle.load(open( img_file, 'rb' ), encoding='bytes')
77         self.targets = np.genfromtxt(label_file, delimiter=',', skip_header=1
    )[:,1:]
78         if idx is not None:
79             self.targets = self.targets[idx]
80             self.data = self.data[idx]
81             self.transform = transform
82             self.targets -= 5
83
84     def __len__(self):
85         return len(self.targets)
86
87     def __getitem__(self, index):
88         img, target = self.data[index], int(self.targets[index])
89         img = Image.fromarray(img.astype('uint8'), mode='L')
90
91         if self.transform is not None:
92             img = self.transform(img)
93
94         return img, target
95
96 # Read image data and their label into a Dataset class
97 dataset = MyDataset('./Train.pkl', './TrainLabels.csv', transform=
    img_transform, idx=None)
98
99 batch_size = 32 #feel free to change it
100 dataloader = DataLoader(dataset, batch_size=batch_size, shuffle=True)
101
102 # Read a batch of data and their labels and display them
103 # Note that since data are transformed, they are between [-1,1]
104 imgs, labels = (next(iter(dataloader)))
105 imgs = np.squeeze(imgs)
106 plt.imshow(imgs[5].cpu().numpy(), cmap='gray', vmin=-1, vmax=1) #.transpose()
107
108 """"# CNN""""
109
110 import torch.nn as nn
111 import torch.nn.functional as F
112

```

```

113 import torch.optim as optim
114
115 # cnn
116 # This cnn is based on the structure of resnet18
117 class Net(nn.Module):
118     def __init__(self):
119         super(Net, self).__init__()
120         self.conv1 = nn.Conv2d(1, 32, kernel_size=(7,7), stride=(2, 2),
padding=(3, 3), bias=False)
121         self.bn1 = nn.BatchNorm2d(32, eps=1e-5, momentum=0.1, affine=True,
track_running_stats=True)
122         self.relu = nn.ReLU(inplace=True)
123         self.maxpool = nn.MaxPool2d(kernel_size=3, stride=2, padding=1,
dilation=1, ceil_mode=False)
124
125         # Layer 1
126         self.layer1block1conv1 = nn.Conv2d(32, 32, kernel_size=(3, 3), stride
=(1, 1), padding=(1, 1), bias=False)
127         self.layer1block1bn1 = nn.BatchNorm2d(32, eps=1e-05, momentum=0.1,
affine=True, track_running_stats=True)
128         self.layer1block1relu = nn.ReLU(inplace=True)
129         self.layer1block1conv2 = nn.Conv2d(32, 32, kernel_size=(3, 3), stride
=(1, 1), padding=(1, 1), bias=False)
130         self.layer1block1bn2 = nn.BatchNorm2d(32, eps=1e-05, momentum=0.1,
affine=True, track_running_stats=True)
131
132         self.layer1block2conv1 = nn.Conv2d(32, 32, kernel_size=(3, 3), stride
=(1, 1), padding=(1, 1), bias=False)
133         self.layer1block2bn1 = nn.BatchNorm2d(32, eps=1e-05, momentum=0.1,
affine=True, track_running_stats=True)
134         self.layer1block2relu = nn.ReLU(inplace=True)
135         self.layer1block2conv2 = nn.Conv2d(32, 32, kernel_size=(3, 3), stride
=(1, 1), padding=(1, 1), bias=False)
136         self.layer1block2bn2 = nn.BatchNorm2d(32, eps=1e-05, momentum=0.1,
affine=True, track_running_stats=True)
137
138         # Layer 2
139         self.layer2block1conv1 = nn.Conv2d(32, 64, kernel_size=(3, 3), stride
=(2, 2), padding=(1, 1), bias=False)
140         self.layer2block1bn1 = nn.BatchNorm2d(64, eps=1e-05, momentum=0.1,
affine=True, track_running_stats=True)
141         self.layer2block1relu = nn.ReLU(inplace=True)
142         self.layer2block1conv2 = nn.Conv2d(64, 64, kernel_size=(3, 3), stride
=(1, 1), padding=(1, 1), bias=False)
143         self.layer2block1bn2 = nn.BatchNorm2d(64, eps=1e-05, momentum=0.1,
affine=True, track_running_stats=True)
144         self.layer2block1conv3 = nn.Conv2d(64, 64, kernel_size=(1, 1), stride
=(2, 2), bias=False)
145         self.layer2block1bn3 = nn.BatchNorm2d(64, eps=1e-05, momentum=0.1,
affine=True, track_running_stats=True)
146
147         self.layer2block2conv1 = nn.Conv2d(64, 64, kernel_size=(3, 3), stride
=(1, 1), padding=(1, 1), bias=False)
148         self.layer2block2bn1 = nn.BatchNorm2d(64, eps=1e-05, momentum=0.1,
affine=True, track_running_stats=True)
149         self.layer2block2relu = nn.ReLU(inplace=True)
150         self.layer2block2conv2 = nn.Conv2d(64, 64, kernel_size=(3, 3), stride
=(1, 1), padding=(1, 1), bias=False)
151         self.layer2block2bn2 = nn.BatchNorm2d(64, eps=1e-05, momentum=0.1,
affine=True, track_running_stats=True)
152
153         # Layer 3
154         self.layer3block1conv1 = nn.Conv2d(64, 128, kernel_size=(3, 3), stride

```

```

154 =(2, 2), padding=(1, 1), bias=False)
155     self.layer3block1bn1 = nn.BatchNorm2d(128, eps=1e-05, momentum=0.1,
    affine=True, track_running_stats=True)
156     self.layer3block1relu = nn.ReLU(inplace=True)
157     self.layer3block1conv2 = nn.Conv2d(128, 128, kernel_size=(3, 3),
    stride=(1, 1), padding=(1, 1), bias=False)
158     self.layer3block1bn2 = nn.BatchNorm2d(128, eps=1e-05, momentum=0.1,
    affine=True, track_running_stats=True)
159     self.layer3block1conv3 = nn.Conv2d(128, 128, kernel_size=(1, 1),
    stride=(2, 2), bias=False)
160     self.layer3block1bn3 = nn.BatchNorm2d(128, eps=1e-05, momentum=0.1,
    affine=True, track_running_stats=True)
161
162     self.layer3block2conv1 = nn.Conv2d(128, 128, kernel_size=(3, 3),
    stride=(1, 1), padding=(1, 1), bias=False)
163     self.layer3block2bn1 = nn.BatchNorm2d(128, eps=1e-05, momentum=0.1,
    affine=True, track_running_stats=True)
164     self.layer3block2relu = nn.ReLU(inplace=True)
165     self.layer3block2conv2 = nn.Conv2d(128, 128, kernel_size=(3, 3),
    stride=(1, 1), padding=(1, 1), bias=False)
166     self.layer3block2bn2 = nn.BatchNorm2d(128, eps=1e-05, momentum=0.1,
    affine=True, track_running_stats=True)
167
168     # Layer 4
169     self.layer4block1conv1 = nn.Conv2d(128, 256, kernel_size=(3, 3),
    stride=(2, 2), padding=(1, 1), bias=False)
170     self.layer4block1bn1 = nn.BatchNorm2d(256, eps=1e-05, momentum=0.1,
    affine=True, track_running_stats=True)
171     self.layer4block1relu = nn.ReLU(inplace=True)
172     self.layer4block1conv2 = nn.Conv2d(256, 256, kernel_size=(3, 3),
    stride=(1, 1), padding=(1, 1), bias=False)
173     self.layer4block1bn2 = nn.BatchNorm2d(256, eps=1e-05, momentum=0.1,
    affine=True, track_running_stats=True)
174     self.layer4block1conv3 = nn.Conv2d(256, 256, kernel_size=(1, 1),
    stride=(2, 2), bias=False)
175     self.layer4block1bn3 = nn.BatchNorm2d(256, eps=1e-05, momentum=0.1,
    affine=True, track_running_stats=True)
176
177     self.layer4block2conv1 = nn.Conv2d(256, 256, kernel_size=(3, 3),
    stride=(1, 1), padding=(1, 1), bias=False)
178     self.layer4block2bn1 = nn.BatchNorm2d(256, eps=1e-05, momentum=0.1,
    affine=True, track_running_stats=True)
179     self.layer4block2relu = nn.ReLU(inplace=True)
180     self.layer4block2conv2 = nn.Conv2d(256, 256, kernel_size=(3, 3),
    stride=(1, 1), padding=(1, 1), bias=False)
181     self.layer4block2bn2 = nn.BatchNorm2d(256, eps=1e-05, momentum=0.1,
    affine=True, track_running_stats=True)
182
183     self.avgpool = nn.AdaptiveAvgPool2d(output_size=(1,1))
184     self.fc = nn.Linear(in_features=256, out_features=9, bias=True)
185
186     def forward(self, x):
187         x = self.conv1(x)
188         x = self.bn1(x)
189         x = self.relu(x)
190         x = self.maxpool(x)
191
192         # Layer 1
193         x = self.layer1block1conv1(x)
194         x = self.layer1block1bn1(x)
195         x = self.layer1block1relu(x)
196         x = self.layer1block1conv2(x)
197         x = self.layer1block1bn2(x)

```

```

198
199     x = self.layer1block2conv1(x)
200     x = self.layer1block2bn1(x)
201     x = self.layer1block2relu(x)
202     x = self.layer1block2conv2(x)
203     x = self.layer1block2bn2(x)
204
205     # Layer 2
206     x = self.layer2block1conv1(x)
207     x = self.layer2block1bn1(x)
208     x = self.layer2block1relu(x)
209     x = self.layer2block1conv2(x)
210     x = self.layer2block1bn2(x)
211     x = self.layer2block1conv3(x)
212     x = self.layer2block1bn3(x)
213
214     x = self.layer2block2conv1(x)
215     x = self.layer2block2bn1(x)
216     x = self.layer2block2relu(x)
217     x = self.layer2block2conv2(x)
218     x = self.layer2block2bn2(x)
219
220     # Layer 3
221     x = self.layer3block1conv1(x)
222     x = self.layer3block1bn1(x)
223     x = self.layer3block1relu(x)
224     x = self.layer3block1conv2(x)
225     x = self.layer3block1bn2(x)
226     x = self.layer3block1conv3(x)
227     x = self.layer3block1bn3(x)
228
229     x = self.layer3block2conv1(x)
230     x = self.layer3block2bn1(x)
231     x = self.layer3block2relu(x)
232     x = self.layer3block2conv2(x)
233     x = self.layer3block2bn2(x)
234
235     # Layer 4
236     x = self.layer4block1conv1(x)
237     x = self.layer4block1bn1(x)
238     x = self.layer4block1relu(x)
239     x = self.layer4block1conv2(x)
240     x = self.layer4block1bn2(x)
241     x = self.layer4block1conv3(x)
242     x = self.layer4block1bn3(x)
243
244     x = self.layer4block2conv1(x)
245     x = self.layer4block2bn1(x)
246     x = self.layer4block2relu(x)
247     x = self.layer4block2conv2(x)
248     x = self.layer4block2bn2(x)
249
250     x = self.avgpool(x)
251     x = torch.flatten(x, 1)
252     x = self.fc(x)
253
254     return x
255
256 train_index = np.arange(50000)
257 test_index = np.arange(50000, 60000)
258 batch_size = 32 #feel free to change it
259
260 # Read image data and their label into a Dataset class

```

```

261 train_set = MyDataset('./Train.pkl', './TrainLabels.csv', transform=
    img_transform, idx=train_index)
262 train_loader = DataLoader(train_set, batch_size=batch_size, shuffle=True,
    num_workers=2)
263 test_set = MyDataset('./Train.pkl', './TrainLabels.csv', transform=
    img_transform, idx=test_index)
264 test_loader = DataLoader(test_set, batch_size=batch_size, shuffle=True,
    num_workers=2)
265
266 net = Net()
267 # if there is a available cuda device, use GPU, else, use CPU
268 device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
269 net = net.to(device)
270
271 # set criterion to cross entropy loss
272 criterion = nn.CrossEntropyLoss()
273 # set learning rate to 0.001
274 optimizer = optim.SGD(net.parameters(), lr=0.01, momentum=0.5)
275
276 running_loss = 0.0
277 num_epochs = 32
278 for epoch in range(num_epochs):
279     for i, data in enumerate(train_loader, 0):
280         img, label = data
281         img = img.to(device)
282         label = label.to(device)
283
284         # zero the parameter gradients
285         optimizer.zero_grad()
286
287         # forward + backward + optimize
288         outputs = net(img)
289         loss = criterion(outputs, label)
290         loss.backward()
291         optimizer.step()
292
293         # print statistics
294         running_loss += loss.item()
295         if i % 320 == 319: # print every 320 mini-batches
296             print('[%d, %5d] loss: %.3f' %
297                 (epoch + 1, i + 1, running_loss / 320))
298             running_loss = 0.0
299
300             torch.save(net.state_dict(), './model.pth')
301             torch.save(optimizer.state_dict(), './optimizer.pth')
302
303 print('Finished Training')
304
305 correct = 0
306 total = 0
307
308 # calculate accuracy
309 with torch.no_grad():
310     for data in test_loader:
311         images, labels = data
312         images = images.to(device)
313         labels = labels.to(device)
314         outputs = net(images)
315         # get the index of the max output
316         _, predicted = torch.max(outputs.data, 1)
317         total += labels.size(0)
318         correct += (predicted == labels).sum().item()
319

```

```
320 print('Accuracy of the network: %d %%' % (  
321     100 * correct/ total))
```