

▼ CNN

▼ Import MNIST Images - Deep Learning with PyTorch 14

```

1 import torch
2 import torch.nn as nn
3 import torch.nn.functional as F
4 from torch.utils.data import DataLoader
5 from torchvision import datasets, transforms
6 from torchvision.utils import make_grid
7
8 import numpy as np
9 import pandas as pd
10 from sklearn.metrics import confusion_matrix
11 import matplotlib.pyplot as plt
12
13 %matplotlib inline

1 # Convert MNIST Image Files into a Tensor of 4-Dimensions (# of images, Height, Width, Color Channel)
2 transform = transforms.ToTensor()

1 # Train Data
2 train_data = datasets.MNIST(root='/cnn_data', train=True, download=True, transform=transform)

Downloading http://yann.lecun.com/exdb/mnist/train-images-idx3-ubyte.gz
Downloading http://yann.lecun.com/exdb/mnist/train-images-idx3-ubyte.gz to /cnn_data/MNIST/raw/train-imag
100%|██████████| 9912422/9912422 [00:00<00:00, 83015443.19it/s]
Extracting /cnn_data/MNIST/raw/train-images-idx3-ubyte.gz to /cnn_data/MNIST/raw

Downloading http://yann.lecun.com/exdb/mnist/train-labels-idx1-ubyte.gz
Downloading http://yann.lecun.com/exdb/mnist/train-labels-idx1-ubyte.gz to /cnn_data/MNIST/raw/train-labe
100%|██████████| 28881/28881 [00:00<00:00, 35091452.44it/s]Extracting /cnn_data/MNIST/raw/train-labels-ic

Downloading http://yann.lecun.com/exdb/mnist/t10k-images-idx3-ubyte.gz

Downloading http://yann.lecun.com/exdb/mnist/t10k-images-idx3-ubyte.gz to /cnn_data/MNIST/raw/t10k-images
100%|██████████| 1648877/1648877 [00:00<00:00, 19775792.85it/s]Extracting /cnn_data/MNIST/raw/t10k-images

Downloading http://yann.lecun.com/exdb/mnist/t10k-labels-idx1-ubyte.gz

Downloading http://yann.lecun.com/exdb/mnist/t10k-labels-idx1-ubyte.gz to /cnn_data/MNIST/raw/t10k-labels
100%|██████████| 4542/4542 [00:00<00:00, 3386760.67it/s]
Extracting /cnn_data/MNIST/raw/t10k-labels-idx1-ubyte.gz to /cnn_data/MNIST/raw

1 # test Data
2 test_data = datasets.MNIST(root='/cnn_data', train=False, download=True, transform=transform)

1 train_data

Dataset MNIST
  Number of datapoints: 60000
  Root location: /cnn_data
  Split: Train
  StandardTransform
  Transform: ToTensor()

1 test_data

Dataset MNIST
  Number of datapoints: 10000
  Root location: /cnn_data
  Split: Test
  _
  _
  _

```

```
StandardTransform
Transform: ToTensor()
```

```
1 pwd
```

```
    '/content'
```

```
1 ls
```

```
    sample_data/
```

```
1 cd ../
```

```
    /
```

```
1 ls -al
```

```
total 116
drwxr-xr-x  1 root root 4096 Oct 17 21:53 ./
drwxr-xr-x  1 root root 4096 Oct 17 21:53 ../
lrwxrwxrwx  1 root root    7 Jun  5 14:02 bin -> usr/bin/
drwxr-xr-x  2 root root 4096 Apr 18  2022 boot/
drwxr-xr-x  3 root root 4096 Oct 17 21:53 cnn_data/
drwxr-xr-x  1 root root 4096 Oct 16 13:23 content/
-rw-r--r--  1 root root 4332 Jun 21 00:40 cuda-keyring_1.0-1_all.deb
drwxr-xr-x  1 root root 4096 Oct 16 13:52 datalab/
drwxr-xr-x  6 root root 460 Oct 17 21:51 dev/
-rwxr-xr-x  1 root root    0 Oct 17 21:51 .dockerenv*
drwxr-xr-x  1 root root 4096 Oct 17 21:51 etc/
drwxr-xr-x  2 root root 4096 Apr 18  2022 home/
lrwxrwxrwx  1 root root    7 Jun  5 14:02 lib -> usr/lib/
lrwxrwxrwx  1 root root    9 Jun  5 14:02 lib32 -> usr/lib32/
lrwxrwxrwx  1 root root    9 Jun  5 14:02 lib64 -> usr/lib64/
lrwxrwxrwx  1 root root   10 Jun  5 14:02 libx32 -> usr/libx32/
drwxr-xr-x  2 root root 4096 Jun  5 14:02 media/
drwxr-xr-x  2 root root 4096 Jun  5 14:02 mnt/
-rw-r--r--  1 root root 17294 Jun 21 00:39 NGC-DL-CONTAINER-LICENSE
drwxr-xr-x  1 root root 4096 Oct 17 21:51 opt/
dr-xr-xr-x 205 root root    0 Oct 17 21:51 proc/
drwxr-xr-x 15 root root 4096 Oct 16 13:20 python-apt/
drwx----- 1 root root 4096 Oct 16 13:53 root/
drwxr-xr-x  1 root root 4096 Oct 16 13:15 run/
lrwxrwxrwx  1 root root    8 Jun  5 14:02/sbin -> usr/sbin/
drwxr-xr-x  2 root root 4096 Jun  5 14:02 srv/
dr-xr-xr-x 13 root root    0 Oct 17 21:51 sys/
drwxrwxrwt  1 root root 4096 Oct 17 21:53 tmp/
drwxr-xr-x  1 root root 4096 Oct 16 13:39 tools/
drwxr-xr-x  1 root root 4096 Oct 17 21:51 usr/
drwxr-xr-x  1 root root 4096 Oct 16 13:52 var/
```

```
1 cd cnn_data
```

```
    /cnn_data
```

```
1 ls -l
```

```
total 4
drwxr-xr-x 3 root root 4096 Oct 17 21:53 MNIST/
```

```
1 cd /
```

```
    /
```

```
1 ls -l
```

```
total 108
lrwxrwxrwx  1 root root    7 Jun  5 14:02 bin -> usr/bin/
```

```
total 16
drwxr-xr-x 1 root root 4096 Oct 16 13:23 ./
drwxr-xr-x 1 root root 4096 Oct 17 21:53 ../
drwxr-xr-x 4 root root 4096 Oct 16 13:23 .config/
drwxr-xr-x 1 root root 4096 Oct 16 13:23 sample\_data/
```

```
tensor([[[[0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
          0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
          0.0000, 0.0000, 0.0000, 0.0000],
         [0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
          0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
          0.0000, 0.0000, 0.0000, 0.0000],
         [0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
          0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
          0.0000, 0.0000, 0.0000, 0.0000],
         [0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
          0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
          0.0000, 0.0000, 0.0000, 0.0000]]]])
```

```

0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
0.0000, 0.0000, 0.0000, 0.0000],
[0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
0.0000, 0.0000, 0.0000, 0.0000],
[0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
0.0000, 0.0000, 0.0000, 0.0000],
[0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
0.0000, 0.0000, 0.0000, 0.0000],
[0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
0.0000, 0.0000, 0.0000, 0.0000],
[0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
0.0000, 0.0000, 0.0000, 0.0000, 0.0118, 0.0706, 0.0706, 0.0706,
0.4941, 0.5333, 0.6863, 0.1020, 0.6510, 1.0000, 0.9686, 0.4980,
0.0000, 0.0000, 0.0000, 0.0000],
[0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
0.1176, 0.1412, 0.3686, 0.6039, 0.6667, 0.9922, 0.9922, 0.9922,
0.9922, 0.9922, 0.8824, 0.6745, 0.9922, 0.9490, 0.7647, 0.2510,
0.0000, 0.0000, 0.0000, 0.0000],
[0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.1922,
0.9333, 0.9922, 0.9922, 0.9922, 0.9922, 0.9922, 0.9922, 0.9922,
0.9922, 0.9843, 0.3647, 0.3216, 0.3216, 0.2196, 0.1529, 0.0000,
0.0000, 0.0000, 0.0000, 0.0000],
[0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0706,
0.8588, 0.9922, 0.9922, 0.9922, 0.9922, 0.9922, 0.7765, 0.7137,
0.9686, 0.9451, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
0.0000, 0.0000, 0.0000, 0.0000],
[0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
0.3137, 0.6118, 0.4196, 0.9922, 0.9922, 0.8039, 0.0431, 0.0000,
0.1686, 0.6039, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
0.0000, 0.0000, 0.0000, 0.0000],
[0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
0.0000, 0.0549, 0.0039, 0.6039, 0.9922, 0.3529, 0.0000, 0.0000,
0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
0.0000, 0.0000, 0.0000, 0.0000],
[0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
0.0000, 0.0000, 0.0000, 0.5451, 0.9922, 0.7451, 0.0078, 0.0000,
0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
0.0000, 0.0000, 0.0000, 0.0000],
[0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
0.0000, 0.0000, 0.0000, 0.0431, 0.7451, 0.9922, 0.2745, 0.0000,
0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
0.0000, 0.0000, 0.0000, 0.0000],
[0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
0.0000, 0.0000, 0.0000, 0.0000, 0.1373, 0.9451, 0.8824, 0.6275,
0.4235, 0.0039, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
0.0000, 0.0000, 0.0000, 0.0000],
[0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.3176, 0.9412, 0.9922,

```

```
1 X_train.shape
```

```
torch.Size([1, 28, 28])
```

```
1 x = X_train.view(1,1, 28, 28)
```

```
1 # Perform the first convolution
```

```
2 x = F.relu(conv1(x)) # Rectified Linear Unit for the activation function
```

```
1 x
```

```

tensor([[[[0.0787, 0.0787, 0.0787, ..., 0.0787, 0.0787, 0.0787],
[0.0787, 0.0787, 0.0787, ..., 0.0787, 0.0787, 0.0787],
[0.0787, 0.0787, 0.0787, ..., 0.0787, 0.0787, 0.0787],
...,
[0.0787, 0.0787, 0.1688, ..., 0.0787, 0.0787, 0.0787],
[0.0787, 0.0787, 0.0787, ..., 0.0787, 0.0787, 0.0787],

```

```

[[0.0787, 0.0787, 0.0787, ..., 0.0787, 0.0787, 0.0787],
 [0.0787, 0.0787, 0.0787, ..., 0.0787, 0.0787, 0.0787]],

[[0.1850, 0.1850, 0.1850, ..., 0.1850, 0.1850, 0.1850],
 [0.1850, 0.1850, 0.1850, ..., 0.1850, 0.1850, 0.1850],
 [0.1850, 0.1850, 0.1850, ..., 0.1850, 0.1850, 0.1850],
 ...,
 [0.1850, 0.1850, 0.1209, ..., 0.1850, 0.1850, 0.1850],
 [0.1850, 0.1850, 0.2727, ..., 0.1850, 0.1850, 0.1850],
 [0.1850, 0.1850, 0.1850, ..., 0.1850, 0.1850, 0.1850]],

[[0.0170, 0.0170, 0.0170, ..., 0.0170, 0.0170, 0.0170],
 [0.0170, 0.0170, 0.0170, ..., 0.0170, 0.0170, 0.0170],
 [0.0170, 0.0170, 0.0170, ..., 0.0170, 0.0170, 0.0170],
 ...,
 [0.0170, 0.0170, 0.1280, ..., 0.0170, 0.0170, 0.0170],
 [0.0170, 0.0170, 0.1844, ..., 0.0170, 0.0170, 0.0170],
 [0.0170, 0.0170, 0.0170, ..., 0.0170, 0.0170, 0.0170]],

[[0.1734, 0.1734, 0.1734, ..., 0.1734, 0.1734, 0.1734],
 [0.1734, 0.1734, 0.1734, ..., 0.1734, 0.1734, 0.1734],
 [0.1734, 0.1734, 0.1734, ..., 0.1734, 0.1734, 0.1734],
 ...,
 [0.1734, 0.1734, 0.0000, ..., 0.1734, 0.1734, 0.1734],
 [0.1734, 0.1734, 0.0454, ..., 0.1734, 0.1734, 0.1734],
 [0.1734, 0.1734, 0.1734, ..., 0.1734, 0.1734, 0.1734]],

[[0.0000, 0.0000, 0.0000, ..., 0.0000, 0.0000, 0.0000],
 [0.0000, 0.0000, 0.0000, ..., 0.0000, 0.0000, 0.0000],
 [0.0000, 0.0000, 0.0000, ..., 0.0000, 0.0000, 0.0000],
 ...,
 [0.0000, 0.0000, 0.0000, ..., 0.0000, 0.0000, 0.0000],
 [0.0000, 0.0000, 0.0000, ..., 0.0000, 0.0000, 0.0000],
 [0.0000, 0.0000, 0.0000, ..., 0.0000, 0.0000, 0.0000]],

[[0.0000, 0.0000, 0.0000, ..., 0.0000, 0.0000, 0.0000],
 [0.0000, 0.0000, 0.0000, ..., 0.0000, 0.0000, 0.0000],
 [0.0000, 0.0000, 0.0000, ..., 0.0000, 0.0000, 0.0000],
 ...,
 [0.0000, 0.0000, 0.0000, ..., 0.0000, 0.0000, 0.0000],
 [0.0000, 0.0000, 0.0000, ..., 0.0000, 0.0000, 0.0000],
 [0.0000, 0.0000, 0.0000, ..., 0.0000, 0.0000, 0.0000]]],
grad_fn=<ReluBackward0>)

```

```

1 # 1 is the single image, 6 is the filters asked for, 26x26
2 x.shape

```

```
torch.Size([1, 6, 26, 26])
```

```

1 # Pass through the pooling layer
2 x = F.max_pool2d(x, 2, 2) # Kernel of 2 and stride of 2

```

```
1 x.shape # 26 / 2 = 13
```

```
torch.Size([1, 6, 13, 13])
```

```

1 # Do the second convolutional layer
2 x = F.relu(conv2(x))

```

```
1 x.shape # Again, no padding was specified so 2 pixels were lost around the outside of the image
```

```
torch.Size([1, 16, 11, 11])
```

```

1 # Pooling layer
2 x = F.max_pool2d(x, 2, 2)

```

```
1 x.shape # 11 / 2 = 5.5 but it is rounded down because no data can be invented to round up
```

```
torch.Size([1, 16, 5, 5])
```

```
1 (((28-2) / 2) -2) / 2

5.5
```

Convolutional Neural Network Model - Deep Learning with PyTorch 16

```
1 # Model Class
2 class ConvolutionalNetwork(nn.Module):
3     def __init__(self) -> None:
4         super().__init__()
5         self.conv1 = nn.Conv2d(1, 6, 3, 1)
6         self.conv2 = nn.Conv2d(6, 16, 3, 1)
7
8         # Fully Connected Layers
9         self.fc1 = nn.Linear(5*5*16, 120)
10        self.fc2 = nn.Linear(120, 84)
11        self.fc3 = nn.Linear(84, 10)
12
13    def forward(self, X):
14        X = F.relu(self.conv1(X))
15        X = F.max_pool2d(X, 2, 2) # 2x2 kernel and stride = 2
16        # Second pass
17        X = F.relu(self.conv2(X))
18        X = F.max_pool2d(X, 2, 2) # 2x2 kernel and stride = 2
19
20        # Re-View the data to flatten it out
21        X = X.view(-1, 16*5*5) # Negative one so the batch size can be varied
22
23        # Fully Connected Layers
24        X = F.relu(self.fc1(X))
25        X = F.relu(self.fc2(X))
26        X = self.fc3(X)
27
28        return F.log_softmax(X, dim=1)
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000
```

Train and Test CNN Model - Deep Learning with PyTorch 17

```
1 import time
2 start_time = time.time()
3
4 # Create Variables to track things
5 epochs = 5
6 train_losses = []
7 test_losses = []
8 train_correct = []
9 test_correct = []
```

```

9 test_correct = 0
10
11
12 # For Loop offor Epochs
13 for i in range(epochs):
14     training_correct = 0
15     testing_correct = 0
16
17     # Train
18     for b, (X_train, y_train) in enumerate(train_loader):
19         b += 1 # Start the batches at 1
20
21         y_pred = model(X_train) # Get the predicted values from the training set (data is 2d, not flattened.)
22         loss = criterion(y_pred, y_train) # How off are we? Compare the predictions to the correct answers in y_train
23
24         predicted = torch.max(y_pred.data, 1)[1] # Add up the number of correct predictions. Indexed off the first point
25         batch_correct = (predicted == y_train).sum() # How many we got correct from this specific batch. True=1, False=0, sum th
26         training_correct += batch_correct # Keep track as we go along in training.
27
28         # Update the parameters
29         optimizer.zero_grad()
30         loss.backward()
31         optimizer.step()
32
33         # Print out some results
34         if b % 600 == 0 :
35             print(f'Epoch: {i} Batch: {b} Loss: {loss.item()}')
36
37
38     train_losses.append(loss)
39     train_correct.append(training_correct)
40
41     # Test
42     with torch.no_grad(): # No gradient so the weights and the bias are not updated with test data
43         for b , (X_test, y_test) in enumerate(test_loader):
44             y_val = model(X_test)
45             predicted = torch.max(y_val.data, 1)[1] # Adding up correct predictions
46             testing_correct += (predicted == y_test).sum() # True=1, False=0, sum all
47
48         loss = criterion(y_val, y_test)
49         test_losses.append(loss)
50         test_correct.append(testing_correct)
51
52 current_time = time.time()
53 total = current_time - start_time
54 print(f'Training time: {total/60} minutes!')
55 total

```

```

Epoch: 0 Batch: 600 Loss: 0.1623624861240387
Epoch: 0 Batch: 1200 Loss: 0.1641543209552765
Epoch: 0 Batch: 1800 Loss: 0.5098981857299805
Epoch: 0 Batch: 2400 Loss: 0.1306418627500534
Epoch: 0 Batch: 3000 Loss: 0.005703817121684551
Epoch: 0 Batch: 3600 Loss: 0.46332210302352905
Epoch: 0 Batch: 4200 Loss: 0.004197881557047367
Epoch: 0 Batch: 4800 Loss: 0.0018000779673457146
Epoch: 0 Batch: 5400 Loss: 0.07375213503837585
Epoch: 0 Batch: 6000 Loss: 0.0003859388525597751
Epoch: 1 Batch: 600 Loss: 0.004290326032787561
Epoch: 1 Batch: 1200 Loss: 0.2521086633205414
Epoch: 1 Batch: 1800 Loss: 0.002427826402708888
Epoch: 1 Batch: 2400 Loss: 0.0021776421926915646
Epoch: 1 Batch: 3000 Loss: 0.02223074808716774
Epoch: 1 Batch: 3600 Loss: 0.6111965179443359
Epoch: 1 Batch: 4200 Loss: 0.016707444563508034
Epoch: 1 Batch: 4800 Loss: 0.0006908098584972322
Epoch: 1 Batch: 5400 Loss: 0.0002799260546453297
Epoch: 1 Batch: 6000 Loss: 0.4848875403404236
Epoch: 2 Batch: 600 Loss: 0.03840283304452896
Epoch: 2 Batch: 1200 Loss: 0.005653898231685162
Epoch: 2 Batch: 1800 Loss: 0.0019390363013371825
Epoch: 2 Batch: 2400 Loss: 0.013562331907451153
Epoch: 2 Batch: 3000 Loss: 0.004443833604454994
Epoch: 2 Batch: 3600 Loss: 0.0005063370917923748
Epoch: 2 Batch: 4200 Loss: 0.04586367309093475
Epoch: 2 Batch: 4800 Loss: 0.000673374444586020

```

```

Epoch: 2 Batch: 4000 Loss: 0.000075174444500059
Epoch: 2 Batch: 5400 Loss: 0.02076614275574684
Epoch: 2 Batch: 6000 Loss: 0.13829907774925232
Epoch: 3 Batch: 600 Loss: 0.0021926886402070522
Epoch: 3 Batch: 1200 Loss: 0.10718250274658203
Epoch: 3 Batch: 1800 Loss: 0.0005934062064625323
Epoch: 3 Batch: 2400 Loss: 0.00016517048061359674
Epoch: 3 Batch: 3000 Loss: 0.0017503199633210897
Epoch: 3 Batch: 3600 Loss: 0.0007489544805139303
Epoch: 3 Batch: 4200 Loss: 0.01164452824741602
Epoch: 3 Batch: 4800 Loss: 7.926556281745434e-05
Epoch: 3 Batch: 5400 Loss: 0.039218269288539886
Epoch: 3 Batch: 6000 Loss: 0.012269356288015842
Epoch: 4 Batch: 600 Loss: 0.019970223307609558
Epoch: 4 Batch: 1200 Loss: 0.031921446323394775
Epoch: 4 Batch: 1800 Loss: 0.2468046396970749
Epoch: 4 Batch: 2400 Loss: 0.00011870403250213712
Epoch: 4 Batch: 3000 Loss: 0.0005112775252200663
Epoch: 4 Batch: 3600 Loss: 0.00011618930147960782
Epoch: 4 Batch: 4200 Loss: 0.00035029969876632094
Epoch: 4 Batch: 4800 Loss: 0.04948687180876732
Epoch: 4 Batch: 5400 Loss: 0.03161567822098732
Epoch: 4 Batch: 6000 Loss: 0.0017111159395426512
Training time: 3.0181969881057737 minutes!
181.09181928634644

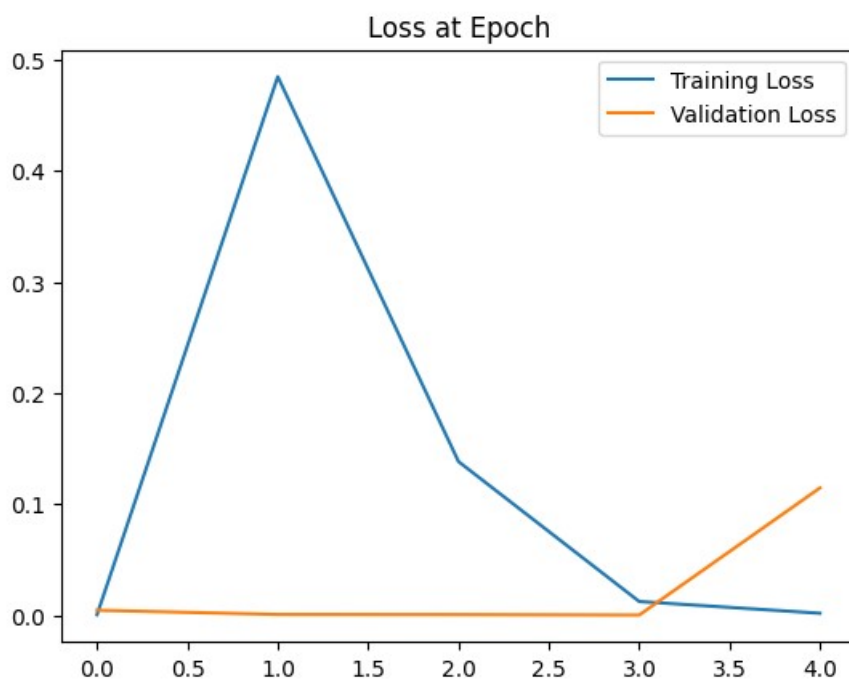
```

```

1 # Graph the loss of Epoch
2 train_losses = [tl.item() for tl in train_losses]
3 plt.plot(train_losses, label='Training Loss')
4 plt.plot(test_losses, label='Validation Loss')
5 plt.title('Loss at Epoch')
6 plt.legend()

```

<matplotlib.legend.Legend at 0x7d8341f4a080>



```

1
2 import time
3 start_time = time.time()
4
5 # Create Variables To Tracks Things
6 epochs = 5
7 train_losses = []
8 test_losses = []
9 train_correct = []
10 test_correct = []
11
12 # For Loop of Epochs
13 for i in range(epochs):

```



```

14  trn_corr = 0
15  tst_corr = 0
16
17
18  # Train
19  for b,(X_train, y_train) in enumerate(train_loader):
20      b+=1 # start our batches at 1
21      y_pred = model(X_train) # get predicted values from the training set. Not flattened 2D
22      loss = criterion(y_pred, y_train) # how off are we? Compare the predictions to correct answers in y_train
23
24      predicted = torch.max(y_pred.data, 1)[1] # add up the number of correct predictions. Indexed off the first point
25      batch_corr = (predicted == y_train).sum() # how many we got correct from this batch. True = 1, False=0, sum those up
26      trn_corr += batch_corr # keep track as we go along in training.
27
28      # Update our parameters
29      optimizer.zero_grad()
30      loss.backward()
31      optimizer.step()
32
33
34      # Print out some results
35      if b%600 == 0:
36          print(f'Epoch: {i}  Batch: {b}  Loss: {loss.item()}')
37
38  train_losses.append(loss)
39  train_correct.append(trn_corr)
40
41
42  # Test
43  with torch.no_grad(): #No gradient so we don't update our weights and biases with test data
44      for b,(X_test, y_test) in enumerate(test_loader):
45          y_val = model(X_test)
46          predicted = torch.max(y_val.data, 1)[1] # Adding up correct predictions
47          tst_corr += (predicted == y_test).sum() # T=1 F=0 and sum away
48
49
50  loss = criterion(y_val, y_test)
51  test_losses.append(loss)
52  test_correct.append(tst_corr)
53
54
55
56  current_time = time.time()
57  total = current_time - start_time
58  print(f'Training Took: {total/60} minutes!')
```

```

Epoch: 0  Batch: 600  Loss: 1.5258686971719726e-06
Epoch: 0  Batch: 1200  Loss: 0.168989360332489
Epoch: 0  Batch: 1800  Loss: 2.086142558255233e-06
Epoch: 0  Batch: 2400  Loss: 2.9802276912960224e-07
Epoch: 0  Batch: 3000  Loss: 9.488784598943312e-06
Epoch: 0  Batch: 3600  Loss: 0.014862915500998497
Epoch: 0  Batch: 4200  Loss: 0.00016575635527260602
Epoch: 0  Batch: 4800  Loss: 0.00015801463450770825
Epoch: 0  Batch: 5400  Loss: 0.00019357565906830132
Epoch: 0  Batch: 6000  Loss: 3.496990757412277e-05
Epoch: 1  Batch: 600  Loss: 6.198691153258551e-06
Epoch: 1  Batch: 1200  Loss: 1.1455599633336533e-05
Epoch: 1  Batch: 1800  Loss: 0.001703915884718299
Epoch: 1  Batch: 2400  Loss: 1.4305105366929638e-07
Epoch: 1  Batch: 3000  Loss: 1.275532326872053e-06
Epoch: 1  Batch: 3600  Loss: 0.003019407857209444
Epoch: 1  Batch: 4200  Loss: 2.8133265459473478e-06
Epoch: 1  Batch: 4800  Loss: 4.0172722037823405e-06
Epoch: 1  Batch: 5400  Loss: 7.653100510651711e-06
Epoch: 1  Batch: 6000  Loss: 0.0010450903791934252
Epoch: 2  Batch: 600  Loss: 8.793851884547621e-05
Epoch: 2  Batch: 1200  Loss: 6.437282991100801e-07
Epoch: 2  Batch: 1800  Loss: 4.410739222748816e-07
Epoch: 2  Batch: 2400  Loss: 1.4066652056499152e-06
Epoch: 2  Batch: 3000  Loss: 0.0011308621615171432
Epoch: 2  Batch: 3600  Loss: 3.187211768818088e-05
Epoch: 2  Batch: 4200  Loss: 4.9947784646064974e-06
Epoch: 2  Batch: 4800  Loss: 0.00017195694090332836
Epoch: 2  Batch: 5400  Loss: 0.18030156195163727
```

```

Epoch: 2 Batch: 6000 Loss: 2.675892028491944e-05
Epoch: 3 Batch: 600 Loss: 0.0
Epoch: 3 Batch: 1200 Loss: 2.2172694116306957e-06
Epoch: 3 Batch: 1800 Loss: 1.192092824453539e-08
Epoch: 3 Batch: 2400 Loss: 1.6689291726379452e-07
Epoch: 3 Batch: 3000 Loss: 1.7642827288000262e-06
Epoch: 3 Batch: 3600 Loss: 0.00017890651361085474
Epoch: 3 Batch: 4200 Loss: 0.00012734861229546368
Epoch: 3 Batch: 4800 Loss: 1.7881379221762472e-07
Epoch: 3 Batch: 5400 Loss: 3.3139699553430546e-06
Epoch: 3 Batch: 6000 Loss: 0.0
Epoch: 4 Batch: 600 Loss: 0.00020975605002604425
Epoch: 4 Batch: 1200 Loss: 1.715281541692093e-05
Epoch: 4 Batch: 1800 Loss: 5.614644578599837e-06
Epoch: 4 Batch: 2400 Loss: 1.561633325763978e-06
Epoch: 4 Batch: 3000 Loss: 1.9835362763842568e-05
Epoch: 4 Batch: 3600 Loss: 2.360334747208981e-06
Epoch: 4 Batch: 4200 Loss: 2.2159549189382233e-05
Epoch: 4 Batch: 4800 Loss: 7.152555525635762e-08
Epoch: 4 Batch: 5400 Loss: 0.017368320375680923
Epoch: 4 Batch: 6000 Loss: 0.142767995595932
Training Took: 3.449174189567566 minutes!

```

Double-cliquez (ou appuyez sur Entrée) pour modifier

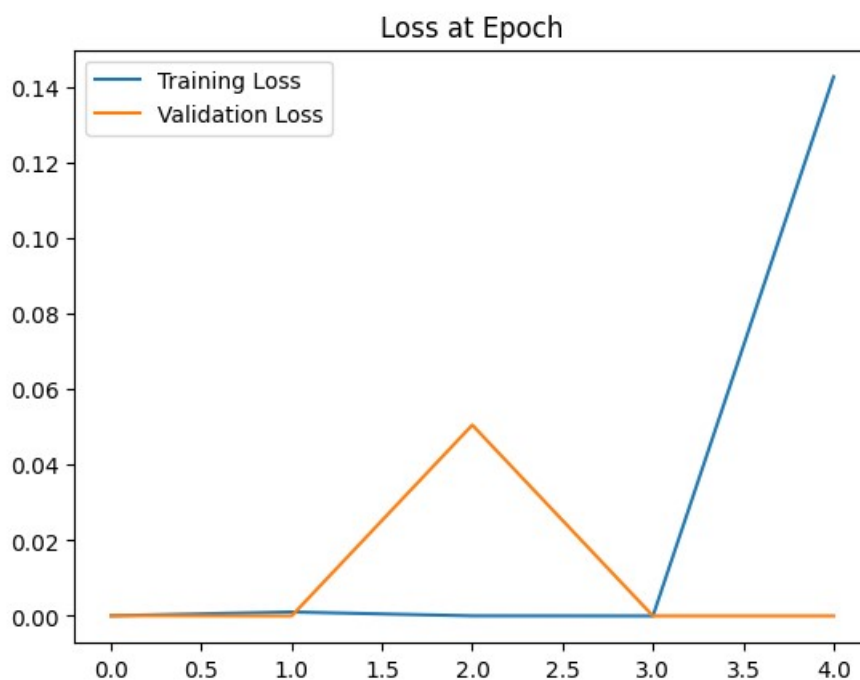
Graph CNN Results - Deep Learning with PyTorch 18

```

1 # Graph the loss of Epoch
2 train_losses = [tl.item() for tl in train_losses]
3 plt.plot(train_losses, label='Training Loss')
4 plt.plot(test_losses, label='Validation Loss')
5 plt.title('Loss at Epoch')
6 plt.legend()

```

<matplotlib.legend.Legend at 0x7d833f02dea0>

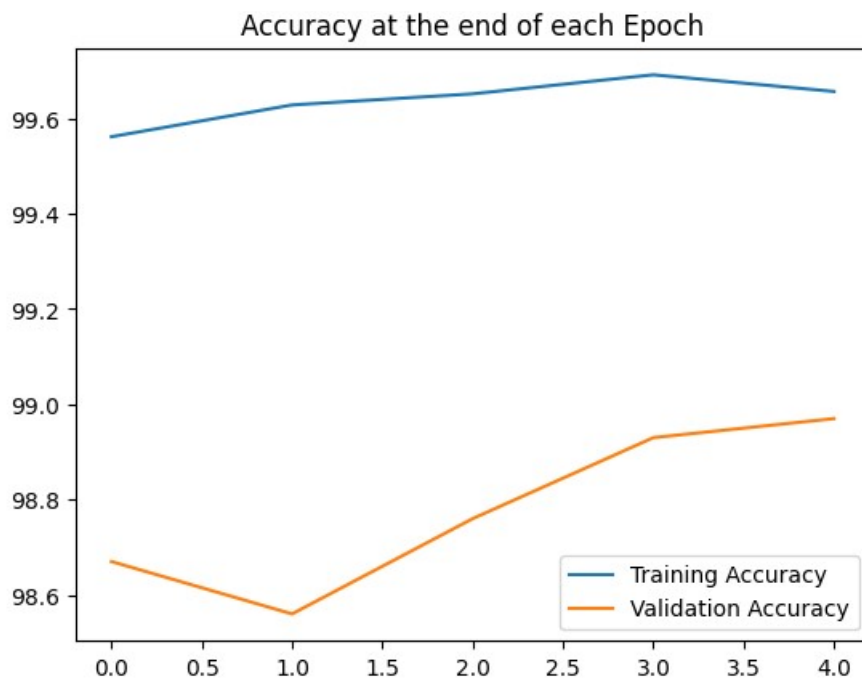


```

1 # Graph the accuracy at the end of each Epoch
2 plt.plot([t/600 for t in train_correct], label='Training Accuracy')
3 plt.plot([t/100 for t in test_correct], label='Validation Accuracy')
4 plt.title('Accuracy at the end of each Epoch')
5 plt.legend()

```

<matplotlib.legend.Legend at 0x7d833eefb070>



```
1 test_load_everything = DataLoader(test_data, batch_size=10_000, shuffle=False)
```

```
1 with torch.no_grad():
2     correct = 0
3
4     for X_test, y_test in test_load_everything:
5         y_val = model(X_test)
6         predicted = torch.max(y_val.data, 1)[1]
7         correct += (predicted == y_test).sum()
8
```

```
1 # Did for correct (out of 10000)
2 correct.item()
```

9897

```
1 # Did for correct (percentage of correct)
2 correct.item() / len(test_data) * 100
```

98.97

Double-cliquez (ou appuyez sur Entrée) pour modifier

