JacobsonDSC550Week11

February 23, 2025

In this exercise, you will build a convolutional neural network (CNN) to classify handwritten digits from the MNIST dataset. The steps to build this CNN classifier with PyTorch are provided in section 22.1 of Machine Learning with Python Cookbook 2nd Edition. The steps to do this with TensorFlow are outlined in section 20.15 of the Machine Learning with Python Cookbook 1st Edition or at various web links, e.g., https://www.kaggle.com/code/amyjang/tensorflow-mnist-cnn-tutorial. Keep in mind that your code may need to be modified depending on your library versions of PyTorch, Keras, and Tensorflow.

Perform the following steps for this exercise.

1. Load the MNIST data set.

[28]:
```
import torch
import torchvision

# Prevent randomness to ensure retestability
random_seed = 1
torch.backends.cudnn.enabled = False
torch.manual_seed(random_seed)

# Mean and Standard Error were obtained from online sources
mnist_mean = 0.1307
mnist_stderr = 0.3081

# Set up a transformation for when we download the data
transform = torchvision.transforms.Compose([

    # Convert images to tensors
    torchvision.transforms.ToTensor(),

    # Normalize with mean and std
    torchvision.transforms.Normalize((mnist_mean,), (mnist_stderr,))
])

# Obtain the MNIST train & test datasets using our transformation
train_dataset = torchvision.datasets.MNIST(
    root='./data',
    train=True,
    download=True,
```

```
    transform=transform
)
test_dataset = torchvision.datasets.MNIST(
    root='./data',
    train=False,
    download=True,
    transform=transform
)

# Build the data loaders.
batch_size = 64
train_loader = torch.utils.data.DataLoader(
    train_dataset,
    batch_size=batch_size,
    shuffle=True
)
test_loader = torch.utils.data.DataLoader(
    test_dataset,
    batch_size=batch_size,
    shuffle=True
)
```

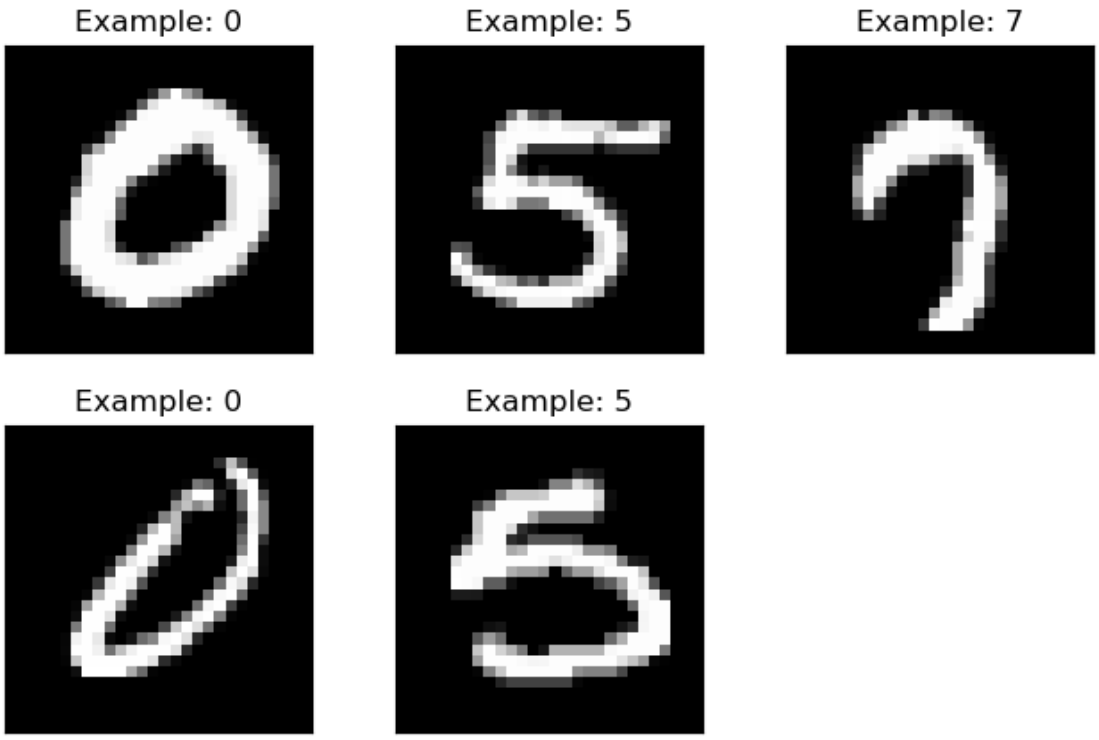
2. Display the first five images in the training data set (see section 8.1 in the Machine Learning with Python Cookbook). Compare these to the first five training labels.

[23]:
```
# Obtain our first batch.
examples = enumerate(test_loader)
batch_idx, (example_data, example_target) = next(examples)
print("Shape of examples: ",example_data.shape)

import matplotlib.pyplot as plt

fig = plt.figure()
for i in range(5): # First five examples
  plt.subplot(2,3,i+1)
  plt.tight_layout()
  plt.imshow(example_data[i][0], cmap='gray', interpolation='none')
  plt.title("Example: {}".format(example_target[i]))
  plt.xticks([])
  plt.yticks([])
```

```
Shape of examples:  torch.Size([64, 1, 28, 28])
```



3. Build and train a Keras CNN classifier on the MNIST training set.

[73]:
```
import warnings
warnings.filterwarnings('ignore')

import torch.nn as nn
import torch.nn.functional as nnf
import torch.optim as optim
import openvino.torch

# Adapted from textbook
class Net(nn.Module):
  def __init__(self):
    super(Net, self).__init__()
    self.conv1 = nn.Conv2d(1,32, kernel_size=3, padding=1)
    self.conv2 = nn.Conv2d(32,64, kernel_size=3, padding=1)
    self.drop1 = nn.Dropout2d(0.25)
    self.drop2 = nn.Dropout2d(0.5)
    self.line1 = nn.Linear(64*14*14,128)
    self.line2 = nn.Linear(128,10)

  def forward(self, x):
    x = nn.functional.relu(self.conv1(x))
```

```
    x = nn.functional.relu(self.conv2(x))
    x = nnf.max_pool2d(self.drop1(x),2)
    x = torch.flatten(x, 1)
    x = nnf.relu(self.line1(self.drop2(x)))
    x = self.line2(x)
    return nnf.log_softmax(x, dim=1) # activate

# Select the device ot run the model on
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

# Create our model with our device
model = Net().to(device)
optimizer = optim.Adam(model.parameters())

# I had challenges compiling this model on Windows due to a
# pytorch bug: https://github.com/pytorch/pytorch/issues/135954
# I was able to modify the backends and get it to compile using
# 'openvino' and then skip compilation on subsequent runs using
# 'cudagraphs' (since it is running on CPU)
# torch._dynamo.list_backends()
model = torch.compile(model, backend='cudagraphs')

# Train the model
model.train()
for idx, (data,target) in enumerate(train_loader):
  data = data.to(device)
  target = target.to(device)
  optimizer.zero_grad()
  output = model(data)
  loss = nnf.nll_loss(output,target)
  loss.backward()
  optimizer.step()
```

```
skipping cudagraphs due to skipping cudagraphs due to cpu device. Found from :
   File "C:\Users\15027\AppData\Local\Temp\ipykernel_12488\1734809149.py", line
21, in forward
    x = nn.functional.relu(self.conv1(x))

skipping cudagraphs due to skipping cudagraphs due to cpu device. Found from :
   File "C:\Users\15027\AppData\Local\Temp\ipykernel_12488\1734809149.py", line
21, in forward
    x = nn.functional.relu(self.conv1(x))
```

4. Report the test accuracy of your model.

[76]:
```
# Test the model
model.eval()
```

```
test_loss = 0
correct = 0
with torch.no_grad():
  for data, target in test_loader:
    data,target = data.to(device), target.to(device)
    output = model(data)
    # get the index of the max log-probability
    test_loss += nn.functional.nll_loss(
      output, target, reduction='sum'
    ).item() # sum up batch loss
    pred = output.argmax(dim=1,keepdim=True)
    correct += pred.eq(target.view_as(pred)).sum().item()

test_total = len(test_loader.dataset)
print("On the first pass, there were only 1,085/10,000 correct, with a test␣
 ↪loss of 2.35")
print("After iterating (below) there are correct: ",correct," of ",test_total,".
 ↪ Test Loss = ",(test_loss/test_total))
```

```
On the first pass, there were only 1,085/10,000 correct, with a test loss of
2.35
After iterating (below) there are correct:  9847  of  10000 . Test Loss =
0.04368605763204396
```

[83]:
```
# Here is another training/test loop that allows us to
# train the model further.
# https://nextjournal.com/gkoehler/pytorch-mnist

n_epochs = 3
log_interval = 10

train_losses = []
train_counter = []
test_losses = []
test_counter = [i*len(train_loader.dataset) for i in range(n_epochs + 1)]

def train(epoch):
  model.train()
  for batch_idx, (data, target) in enumerate(train_loader):
    optimizer.zero_grad()
    output = model(data)
    loss = nnf.nll_loss(output, target)
    loss.backward()
    optimizer.step()
    if batch_idx % log_interval == 0:
      print('Train Epoch: {} [{}/{} ({:.0f}%)]\tLoss: {:.6f}'.format(
        epoch, batch_idx * len(data), len(train_loader.dataset),
```

```
        100. * batch_idx / len(train_loader), loss.item()))
      train_losses.append(loss.item())
      train_counter.append(
        (batch_idx*64) + ((epoch-1)*len(train_loader.dataset)))
      torch.save(model.state_dict(), 'wk11_model.pth')
      torch.save(optimizer.state_dict(), 'wk11_optimizer.pth')

def test():
  model.eval()
  test_loss = 0
  correct = 0
  with torch.no_grad():
    for data, target in test_loader:
      output = model(data)
      test_loss += nnf.nll_loss(output, target, size_average=False).item()
      pred = output.data.max(1, keepdim=True)[1]
      print(pred,target)
      from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
      cm = confusion_matrix(pred, target)
      ConfusionMatrixDisplay(cm).plot()

      break
      correct += pred.eq(target.data.view_as(pred)).sum()
  test_loss /= len(test_loader.dataset)
  test_losses.append(test_loss)
  print('\nTest set: Avg. loss: {:.4f}, Accuracy: {}/{} ({:.0f}%)\n'.format(
    test_loss, correct, len(test_loader.dataset),
    100. * correct / len(test_loader.dataset)))
```

[67]:
```
test()
for epoch in range(1, n_epochs + 1):
  train(epoch)
  test()
```

```
Test set: Avg. loss: 1.8167, Accuracy: 4944/10000 (49%)

Train Epoch: 1 [0/60000 (0%)]   Loss: 1.787303
Train Epoch: 1 [640/60000 (1%)] Loss: 0.789982
Train Epoch: 1 [1280/60000 (2%)]        Loss: 0.233046
Train Epoch: 1 [1920/60000 (3%)]        Loss: 0.528525
Train Epoch: 1 [2560/60000 (4%)]        Loss: 0.332181
Train Epoch: 1 [3200/60000 (5%)]        Loss: 0.505139
Train Epoch: 1 [3840/60000 (6%)]        Loss: 0.346914
Train Epoch: 1 [4480/60000 (7%)]        Loss: 0.462374
Train Epoch: 1 [5120/60000 (9%)]        Loss: 0.279321
Train Epoch: 1 [5760/60000 (10%)]       Loss: 0.162381
Train Epoch: 1 [6400/60000 (11%)]       Loss: 0.189181
```

```
Train Epoch: 1 [7040/60000 (12%)]       Loss: 0.310861
Train Epoch: 1 [7680/60000 (13%)]       Loss: 0.185528
Train Epoch: 1 [8320/60000 (14%)]       Loss: 0.415371
Train Epoch: 1 [8960/60000 (15%)]       Loss: 0.107939
Train Epoch: 1 [9600/60000 (16%)]       Loss: 0.334561
Train Epoch: 1 [10240/60000 (17%)]      Loss: 0.152601
Train Epoch: 1 [10880/60000 (18%)]      Loss: 0.057292
Train Epoch: 1 [11520/60000 (19%)]      Loss: 0.192610
Train Epoch: 1 [12160/60000 (20%)]      Loss: 0.190667
Train Epoch: 1 [12800/60000 (21%)]      Loss: 0.302302
Train Epoch: 1 [13440/60000 (22%)]      Loss: 0.219826
Train Epoch: 1 [14080/60000 (23%)]      Loss: 0.053769
Train Epoch: 1 [14720/60000 (25%)]      Loss: 0.242418
Train Epoch: 1 [15360/60000 (26%)]      Loss: 0.104456
Train Epoch: 1 [16000/60000 (27%)]      Loss: 0.142808
Train Epoch: 1 [16640/60000 (28%)]      Loss: 0.258058
Train Epoch: 1 [17280/60000 (29%)]      Loss: 0.037646
Train Epoch: 1 [17920/60000 (30%)]      Loss: 0.095907
Train Epoch: 1 [18560/60000 (31%)]      Loss: 0.113949
Train Epoch: 1 [19200/60000 (32%)]      Loss: 0.110632
Train Epoch: 1 [19840/60000 (33%)]      Loss: 0.197251
Train Epoch: 1 [20480/60000 (34%)]      Loss: 0.229219
Train Epoch: 1 [21120/60000 (35%)]      Loss: 0.180948
Train Epoch: 1 [21760/60000 (36%)]      Loss: 0.155479
Train Epoch: 1 [22400/60000 (37%)]      Loss: 0.139622
Train Epoch: 1 [23040/60000 (38%)]      Loss: 0.094913
Train Epoch: 1 [23680/60000 (39%)]      Loss: 0.036055
Train Epoch: 1 [24320/60000 (41%)]      Loss: 0.162184
Train Epoch: 1 [24960/60000 (42%)]      Loss: 0.106902
Train Epoch: 1 [25600/60000 (43%)]      Loss: 0.149839
Train Epoch: 1 [26240/60000 (44%)]      Loss: 0.046192
Train Epoch: 1 [26880/60000 (45%)]      Loss: 0.091164
Train Epoch: 1 [27520/60000 (46%)]      Loss: 0.114896
Train Epoch: 1 [28160/60000 (47%)]      Loss: 0.070225
Train Epoch: 1 [28800/60000 (48%)]      Loss: 0.026746
Train Epoch: 1 [29440/60000 (49%)]      Loss: 0.132509
Train Epoch: 1 [30080/60000 (50%)]      Loss: 0.134058
Train Epoch: 1 [30720/60000 (51%)]      Loss: 0.101152
Train Epoch: 1 [31360/60000 (52%)]      Loss: 0.083241
Train Epoch: 1 [32000/60000 (53%)]      Loss: 0.014204
Train Epoch: 1 [32640/60000 (54%)]      Loss: 0.167025
Train Epoch: 1 [33280/60000 (55%)]      Loss: 0.066986
Train Epoch: 1 [33920/60000 (57%)]      Loss: 0.174111
Train Epoch: 1 [34560/60000 (58%)]      Loss: 0.107913
Train Epoch: 1 [35200/60000 (59%)]      Loss: 0.039556
Train Epoch: 1 [35840/60000 (60%)]      Loss: 0.086835
Train Epoch: 1 [36480/60000 (61%)]      Loss: 0.020871
Train Epoch: 1 [37120/60000 (62%)]      Loss: 0.032063
```

```
Train Epoch: 1 [37760/60000 (63%)]      Loss: 0.065531
Train Epoch: 1 [38400/60000 (64%)]      Loss: 0.224747
Train Epoch: 1 [39040/60000 (65%)]      Loss: 0.050120
Train Epoch: 1 [39680/60000 (66%)]      Loss: 0.068773
Train Epoch: 1 [40320/60000 (67%)]      Loss: 0.159216
Train Epoch: 1 [40960/60000 (68%)]      Loss: 0.086746
Train Epoch: 1 [41600/60000 (69%)]      Loss: 0.063236
Train Epoch: 1 [42240/60000 (70%)]      Loss: 0.122778
Train Epoch: 1 [42880/60000 (71%)]      Loss: 0.096187
Train Epoch: 1 [43520/60000 (72%)]      Loss: 0.059142
Train Epoch: 1 [44160/60000 (74%)]      Loss: 0.022261
Train Epoch: 1 [44800/60000 (75%)]      Loss: 0.246865
Train Epoch: 1 [45440/60000 (76%)]      Loss: 0.052239
Train Epoch: 1 [46080/60000 (77%)]      Loss: 0.118324
Train Epoch: 1 [46720/60000 (78%)]      Loss: 0.020976
Train Epoch: 1 [47360/60000 (79%)]      Loss: 0.069391
Train Epoch: 1 [48000/60000 (80%)]      Loss: 0.044461
Train Epoch: 1 [48640/60000 (81%)]      Loss: 0.217366
Train Epoch: 1 [49280/60000 (82%)]      Loss: 0.059037
Train Epoch: 1 [49920/60000 (83%)]      Loss: 0.057663
Train Epoch: 1 [50560/60000 (84%)]      Loss: 0.055250
Train Epoch: 1 [51200/60000 (85%)]      Loss: 0.181696
Train Epoch: 1 [51840/60000 (86%)]      Loss: 0.048629
Train Epoch: 1 [52480/60000 (87%)]      Loss: 0.019526
Train Epoch: 1 [53120/60000 (88%)]      Loss: 0.026788
Train Epoch: 1 [53760/60000 (90%)]      Loss: 0.019788
Train Epoch: 1 [54400/60000 (91%)]      Loss: 0.069583
Train Epoch: 1 [55040/60000 (92%)]      Loss: 0.048664
Train Epoch: 1 [55680/60000 (93%)]      Loss: 0.137237
Train Epoch: 1 [56320/60000 (94%)]      Loss: 0.102238
Train Epoch: 1 [56960/60000 (95%)]      Loss: 0.150948
Train Epoch: 1 [57600/60000 (96%)]      Loss: 0.060273
Train Epoch: 1 [58240/60000 (97%)]      Loss: 0.016352
Train Epoch: 1 [58880/60000 (98%)]      Loss: 0.013367
Train Epoch: 1 [59520/60000 (99%)]      Loss: 0.173313

Test set: Avg. loss: 0.0503, Accuracy: 9835/10000 (98%)

Train Epoch: 2 [0/60000 (0%)]   Loss: 0.173719
Train Epoch: 2 [640/60000 (1%)] Loss: 0.018145
Train Epoch: 2 [1280/60000 (2%)]        Loss: 0.083318
Train Epoch: 2 [1920/60000 (3%)]        Loss: 0.048203
Train Epoch: 2 [2560/60000 (4%)]        Loss: 0.111381
Train Epoch: 2 [3200/60000 (5%)]        Loss: 0.031121
Train Epoch: 2 [3840/60000 (6%)]        Loss: 0.024273
Train Epoch: 2 [4480/60000 (7%)]        Loss: 0.021723
Train Epoch: 2 [5120/60000 (9%)]        Loss: 0.021337
Train Epoch: 2 [5760/60000 (10%)]       Loss: 0.068354
```

```
Train Epoch: 2 [6400/60000 (11%)]       Loss: 0.069738
Train Epoch: 2 [7040/60000 (12%)]       Loss: 0.063265
Train Epoch: 2 [7680/60000 (13%)]       Loss: 0.088607
Train Epoch: 2 [8320/60000 (14%)]       Loss: 0.120847
Train Epoch: 2 [8960/60000 (15%)]       Loss: 0.078278
Train Epoch: 2 [9600/60000 (16%)]       Loss: 0.039220
Train Epoch: 2 [10240/60000 (17%)]      Loss: 0.006279
Train Epoch: 2 [10880/60000 (18%)]      Loss: 0.050535
Train Epoch: 2 [11520/60000 (19%)]      Loss: 0.053520
Train Epoch: 2 [12160/60000 (20%)]      Loss: 0.040785
Train Epoch: 2 [12800/60000 (21%)]      Loss: 0.068720
Train Epoch: 2 [13440/60000 (22%)]      Loss: 0.052005
Train Epoch: 2 [14080/60000 (23%)]      Loss: 0.066251
Train Epoch: 2 [14720/60000 (25%)]      Loss: 0.079930
Train Epoch: 2 [15360/60000 (26%)]      Loss: 0.091619
Train Epoch: 2 [16000/60000 (27%)]      Loss: 0.045227
Train Epoch: 2 [16640/60000 (28%)]      Loss: 0.026442
Train Epoch: 2 [17280/60000 (29%)]      Loss: 0.231730
Train Epoch: 2 [17920/60000 (30%)]      Loss: 0.028831
Train Epoch: 2 [18560/60000 (31%)]      Loss: 0.019594
Train Epoch: 2 [19200/60000 (32%)]      Loss: 0.093678
Train Epoch: 2 [19840/60000 (33%)]      Loss: 0.028969
Train Epoch: 2 [20480/60000 (34%)]      Loss: 0.067434
Train Epoch: 2 [21120/60000 (35%)]      Loss: 0.122657
Train Epoch: 2 [21760/60000 (36%)]      Loss: 0.017714
Train Epoch: 2 [22400/60000 (37%)]      Loss: 0.149684
Train Epoch: 2 [23040/60000 (38%)]      Loss: 0.188424
Train Epoch: 2 [23680/60000 (39%)]      Loss: 0.038570
Train Epoch: 2 [24320/60000 (41%)]      Loss: 0.023438
Train Epoch: 2 [24960/60000 (42%)]      Loss: 0.117611
Train Epoch: 2 [25600/60000 (43%)]      Loss: 0.065100
Train Epoch: 2 [26240/60000 (44%)]      Loss: 0.036758
Train Epoch: 2 [26880/60000 (45%)]      Loss: 0.017783
Train Epoch: 2 [27520/60000 (46%)]      Loss: 0.099963
Train Epoch: 2 [28160/60000 (47%)]      Loss: 0.032316
Train Epoch: 2 [28800/60000 (48%)]      Loss: 0.066287
Train Epoch: 2 [29440/60000 (49%)]      Loss: 0.010271
Train Epoch: 2 [30080/60000 (50%)]      Loss: 0.067356
Train Epoch: 2 [30720/60000 (51%)]      Loss: 0.099058
Train Epoch: 2 [31360/60000 (52%)]      Loss: 0.009346
Train Epoch: 2 [32000/60000 (53%)]      Loss: 0.129916
Train Epoch: 2 [32640/60000 (54%)]      Loss: 0.038503
Train Epoch: 2 [33280/60000 (55%)]      Loss: 0.017839
Train Epoch: 2 [33920/60000 (57%)]      Loss: 0.022110
Train Epoch: 2 [34560/60000 (58%)]      Loss: 0.038003
Train Epoch: 2 [35200/60000 (59%)]      Loss: 0.030224
Train Epoch: 2 [35840/60000 (60%)]      Loss: 0.009400
Train Epoch: 2 [36480/60000 (61%)]      Loss: 0.021281
```

```
Train Epoch: 2 [37120/60000 (62%)]      Loss: 0.090700
Train Epoch: 2 [37760/60000 (63%)]      Loss: 0.151693
Train Epoch: 2 [38400/60000 (64%)]      Loss: 0.030718
Train Epoch: 2 [39040/60000 (65%)]      Loss: 0.032571
Train Epoch: 2 [39680/60000 (66%)]      Loss: 0.140652
Train Epoch: 2 [40320/60000 (67%)]      Loss: 0.229193
Train Epoch: 2 [40960/60000 (68%)]      Loss: 0.053012
Train Epoch: 2 [41600/60000 (69%)]      Loss: 0.011724
Train Epoch: 2 [42240/60000 (70%)]      Loss: 0.014052
Train Epoch: 2 [42880/60000 (71%)]      Loss: 0.014493
Train Epoch: 2 [43520/60000 (72%)]      Loss: 0.018799
Train Epoch: 2 [44160/60000 (74%)]      Loss: 0.184925
Train Epoch: 2 [44800/60000 (75%)]      Loss: 0.033101
Train Epoch: 2 [45440/60000 (76%)]      Loss: 0.069316
Train Epoch: 2 [46080/60000 (77%)]      Loss: 0.103894
Train Epoch: 2 [46720/60000 (78%)]      Loss: 0.054936
Train Epoch: 2 [47360/60000 (79%)]      Loss: 0.004651
Train Epoch: 2 [48000/60000 (80%)]      Loss: 0.020839
Train Epoch: 2 [48640/60000 (81%)]      Loss: 0.082374
Train Epoch: 2 [49280/60000 (82%)]      Loss: 0.096077
Train Epoch: 2 [49920/60000 (83%)]      Loss: 0.119520
Train Epoch: 2 [50560/60000 (84%)]      Loss: 0.008397
Train Epoch: 2 [51200/60000 (85%)]      Loss: 0.034475
Train Epoch: 2 [51840/60000 (86%)]      Loss: 0.007249
Train Epoch: 2 [52480/60000 (87%)]      Loss: 0.014861
Train Epoch: 2 [53120/60000 (88%)]      Loss: 0.015303
Train Epoch: 2 [53760/60000 (90%)]      Loss: 0.009265
Train Epoch: 2 [54400/60000 (91%)]      Loss: 0.045750
Train Epoch: 2 [55040/60000 (92%)]      Loss: 0.003490
Train Epoch: 2 [55680/60000 (93%)]      Loss: 0.126124
Train Epoch: 2 [56320/60000 (94%)]      Loss: 0.024775
Train Epoch: 2 [56960/60000 (95%)]      Loss: 0.209336
Train Epoch: 2 [57600/60000 (96%)]      Loss: 0.004077
Train Epoch: 2 [58240/60000 (97%)]      Loss: 0.141577
Train Epoch: 2 [58880/60000 (98%)]      Loss: 0.093095
Train Epoch: 2 [59520/60000 (99%)]      Loss: 0.084388

Test set: Avg. loss: 0.0355, Accuracy: 9869/10000 (99%)

Train Epoch: 3 [0/60000 (0%)]   Loss: 0.021192
Train Epoch: 3 [640/60000 (1%)] Loss: 0.047752
Train Epoch: 3 [1280/60000 (2%)]        Loss: 0.026002
Train Epoch: 3 [1920/60000 (3%)]        Loss: 0.006557
Train Epoch: 3 [2560/60000 (4%)]        Loss: 0.018178
Train Epoch: 3 [3200/60000 (5%)]        Loss: 0.027997
Train Epoch: 3 [3840/60000 (6%)]        Loss: 0.039839
Train Epoch: 3 [4480/60000 (7%)]        Loss: 0.040884
Train Epoch: 3 [5120/60000 (9%)]        Loss: 0.017612
```

```
Train Epoch: 3 [5760/60000 (10%)]       Loss: 0.038158
Train Epoch: 3 [6400/60000 (11%)]       Loss: 0.070720
Train Epoch: 3 [7040/60000 (12%)]       Loss: 0.070562
Train Epoch: 3 [7680/60000 (13%)]       Loss: 0.016437
Train Epoch: 3 [8320/60000 (14%)]       Loss: 0.028133
Train Epoch: 3 [8960/60000 (15%)]       Loss: 0.014373
Train Epoch: 3 [9600/60000 (16%)]       Loss: 0.139232
Train Epoch: 3 [10240/60000 (17%)]      Loss: 0.019528
Train Epoch: 3 [10880/60000 (18%)]      Loss: 0.007628
Train Epoch: 3 [11520/60000 (19%)]      Loss: 0.052592
Train Epoch: 3 [12160/60000 (20%)]      Loss: 0.006315
Train Epoch: 3 [12800/60000 (21%)]      Loss: 0.312131
Train Epoch: 3 [13440/60000 (22%)]      Loss: 0.031131
Train Epoch: 3 [14080/60000 (23%)]      Loss: 0.026771
Train Epoch: 3 [14720/60000 (25%)]      Loss: 0.027203
Train Epoch: 3 [15360/60000 (26%)]      Loss: 0.140746
Train Epoch: 3 [16000/60000 (27%)]      Loss: 0.057976
Train Epoch: 3 [16640/60000 (28%)]      Loss: 0.042931
Train Epoch: 3 [17280/60000 (29%)]      Loss: 0.025247
Train Epoch: 3 [17920/60000 (30%)]      Loss: 0.020268
Train Epoch: 3 [18560/60000 (31%)]      Loss: 0.022457
Train Epoch: 3 [19200/60000 (32%)]      Loss: 0.056908
Train Epoch: 3 [19840/60000 (33%)]      Loss: 0.001451
Train Epoch: 3 [20480/60000 (34%)]      Loss: 0.038613
Train Epoch: 3 [21120/60000 (35%)]      Loss: 0.075999
Train Epoch: 3 [21760/60000 (36%)]      Loss: 0.030143
Train Epoch: 3 [22400/60000 (37%)]      Loss: 0.134324
Train Epoch: 3 [23040/60000 (38%)]      Loss: 0.037764
Train Epoch: 3 [23680/60000 (39%)]      Loss: 0.110380
Train Epoch: 3 [24320/60000 (41%)]      Loss: 0.046698
Train Epoch: 3 [24960/60000 (42%)]      Loss: 0.054705
Train Epoch: 3 [25600/60000 (43%)]      Loss: 0.061501
Train Epoch: 3 [26240/60000 (44%)]      Loss: 0.074163
Train Epoch: 3 [26880/60000 (45%)]      Loss: 0.015737
Train Epoch: 3 [27520/60000 (46%)]      Loss: 0.015110
Train Epoch: 3 [28160/60000 (47%)]      Loss: 0.002446
Train Epoch: 3 [28800/60000 (48%)]      Loss: 0.016042
Train Epoch: 3 [29440/60000 (49%)]      Loss: 0.161428
Train Epoch: 3 [30080/60000 (50%)]      Loss: 0.085536
Train Epoch: 3 [30720/60000 (51%)]      Loss: 0.012127
Train Epoch: 3 [31360/60000 (52%)]      Loss: 0.232441
Train Epoch: 3 [32000/60000 (53%)]      Loss: 0.058338
Train Epoch: 3 [32640/60000 (54%)]      Loss: 0.004249
Train Epoch: 3 [33280/60000 (55%)]      Loss: 0.069005
Train Epoch: 3 [33920/60000 (57%)]      Loss: 0.042441
Train Epoch: 3 [34560/60000 (58%)]      Loss: 0.007050
Train Epoch: 3 [35200/60000 (59%)]      Loss: 0.003061
Train Epoch: 3 [35840/60000 (60%)]      Loss: 0.004169
```

```
Train Epoch: 3 [36480/60000 (61%)]      Loss: 0.023669
Train Epoch: 3 [37120/60000 (62%)]      Loss: 0.007486
Train Epoch: 3 [37760/60000 (63%)]      Loss: 0.076680
Train Epoch: 3 [38400/60000 (64%)]      Loss: 0.075477
Train Epoch: 3 [39040/60000 (65%)]      Loss: 0.085464
Train Epoch: 3 [39680/60000 (66%)]      Loss: 0.043327
Train Epoch: 3 [40320/60000 (67%)]      Loss: 0.051415
Train Epoch: 3 [40960/60000 (68%)]      Loss: 0.024234
Train Epoch: 3 [41600/60000 (69%)]      Loss: 0.060456
Train Epoch: 3 [42240/60000 (70%)]      Loss: 0.131241
Train Epoch: 3 [42880/60000 (71%)]      Loss: 0.081116
Train Epoch: 3 [43520/60000 (72%)]      Loss: 0.032721
Train Epoch: 3 [44160/60000 (74%)]      Loss: 0.056822
Train Epoch: 3 [44800/60000 (75%)]      Loss: 0.013124
Train Epoch: 3 [45440/60000 (76%)]      Loss: 0.043620
Train Epoch: 3 [46080/60000 (77%)]      Loss: 0.082773
Train Epoch: 3 [46720/60000 (78%)]      Loss: 0.005880
Train Epoch: 3 [47360/60000 (79%)]      Loss: 0.118412
Train Epoch: 3 [48000/60000 (80%)]      Loss: 0.037641
Train Epoch: 3 [48640/60000 (81%)]      Loss: 0.008767
Train Epoch: 3 [49280/60000 (82%)]      Loss: 0.025132
Train Epoch: 3 [49920/60000 (83%)]      Loss: 0.060461
Train Epoch: 3 [50560/60000 (84%)]      Loss: 0.078745
Train Epoch: 3 [51200/60000 (85%)]      Loss: 0.099730
Train Epoch: 3 [51840/60000 (86%)]      Loss: 0.008808
Train Epoch: 3 [52480/60000 (87%)]      Loss: 0.117262
Train Epoch: 3 [53120/60000 (88%)]      Loss: 0.038653
Train Epoch: 3 [53760/60000 (90%)]      Loss: 0.151319
Train Epoch: 3 [54400/60000 (91%)]      Loss: 0.014580
Train Epoch: 3 [55040/60000 (92%)]      Loss: 0.011940
Train Epoch: 3 [55680/60000 (93%)]      Loss: 0.014991
Train Epoch: 3 [56320/60000 (94%)]      Loss: 0.005064
Train Epoch: 3 [56960/60000 (95%)]      Loss: 0.088861
Train Epoch: 3 [57600/60000 (96%)]      Loss: 0.136915
Train Epoch: 3 [58240/60000 (97%)]      Loss: 0.015551
Train Epoch: 3 [58880/60000 (98%)]      Loss: 0.022551
Train Epoch: 3 [59520/60000 (99%)]      Loss: 0.089453

Test set: Avg. loss: 0.0356, Accuracy: 9892/10000 (99%)
```

Discussion The accuracy started out fairly low (10.9%). By iterating through the training data several times we were able to get the accuracy up to 9892/10000 (99%).

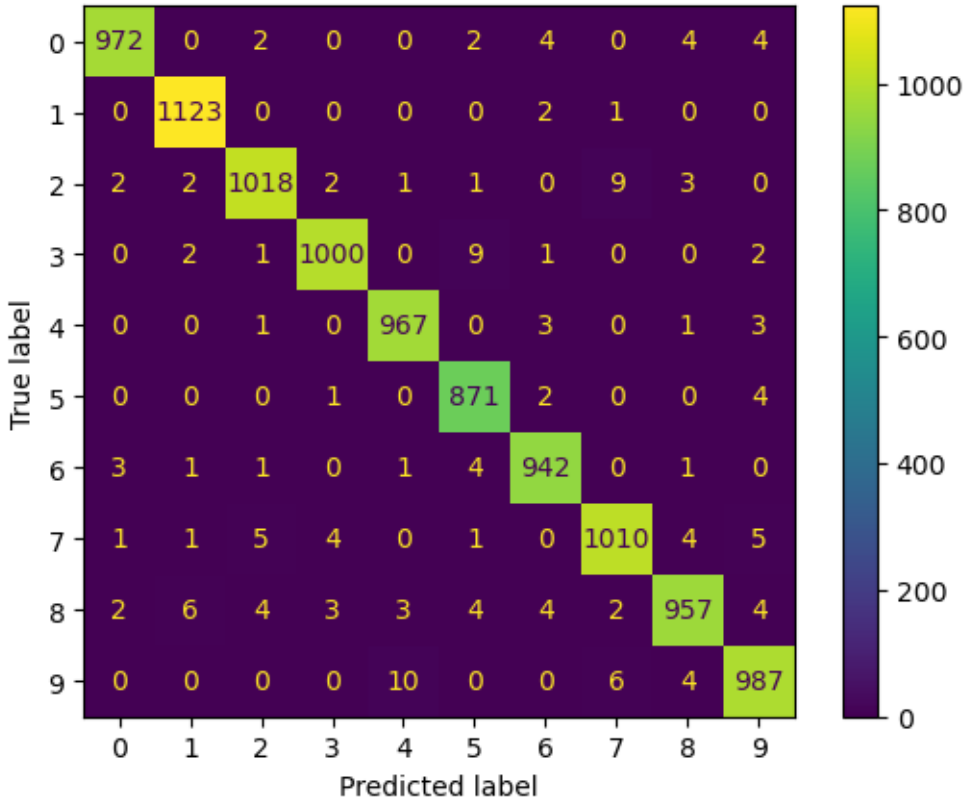
5. Display a confusion matrix on the test set classifications.

[87]:
```
preds = None
targets = None
```

```
model.eval()
with torch.no_grad():
    for data, target in test_loader:
        output = model(data)
        pred = output.data.max(1, keepdim=True)[1]
        if preds is None:
            preds = pred
        else:
            preds = torch.cat((preds,pred))
        if targets is None:
            targets = target
        else:
            targets = torch.cat((targets,target))

from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
cm = confusion_matrix(preds, targets)
ConfusionMatrixDisplay(cm).plot()
```

[87]: <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x238343b5510>



6. Summarize your results.

CNN, and neural networks in general, are very complicated but tools like PyTorch make them relatively simple. Our model used rectified linear units (RELU), convolutional layers, and a dropout layer for regularization. I saw several arrangements of these layers in examples online, but I used the arrangement from our textbook. Obviously, arranging these layers differently would create a different CNN that would behave differently. On the first pass through the training data, the CNN only learned a little and got 11% accuracy. However, four times through the training data caused it to increase the accuracy to 99%. In the confusion matrix, you can see that "4" and "9" are mistaken for each other the most, and "3" and "5" also have a higher error rate. Overall, the model did an excelent job of classifying these images.

[ ]: