Nama : Hurin Salimah
Nim : 1103200021

Input

```
# Check for GPU
!nvidia-smi
```

Output

```
Thu Jan  4 06:38:31 2024
+---------------------------------------------------------------------------------------+
| NVIDIA-SMI 535.104.05             Driver Version: 535.104.05   CUDA Version: 12.2     |
|-----------------------------------------+----------------------+----------------------+
| GPU  Name                 Persistence-M | Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp   Perf          Pwr:Usage/Cap |         Memory-Usage | GPU-Util  Compute M. |
|                                         |                      |               MIG M. |
|=========================================+======================+======================|
|   0  Tesla T4                       Off | 00000000:00:04.0 Off |                    0 |
| N/A   48C    P8               9W /  70W |      0MiB / 15360MiB |      0%      Default |
|                                         |                      |                  N/A |
+-----------------------------------------+----------------------+----------------------+

+---------------------------------------------------------------------------------------+
| Processes:                                                                            |
|  GPU   GI   CI        PID   Type   Process name                            GPU Memory |
|        ID   ID                                                             Usage      |
|=======================================================================================|
|  No running processes found                                                           |
+---------------------------------------------------------------------------------------+
```

Input

```
# Import torch
import torch

# Exercises require PyTorch > 1.10.0
print(torch.__version__)

# Setup device agnostic code
device = "cuda" if torch.cuda.is_available() else "cpu"
```

```
device
```
Output
```
2.1.0+cu121
'cuda'
```

1. What are 3 areas in industry where computer vision is currently being used?

2. Search "what is overfitting in machine learning" and write down a sentence about what you find.

3. Search "ways to prevent overfitting in machine learning", write down 3 of the things you find and a sentence about each.

4. Spend 20-minutes reading and clicking through the CNN Explainer website.

5. Load the torchvision.datasets.MNIST() train and test datasets.

Input
```
# mengimpor pustaka 'torchvision'
import torchvision
# memuat dataset yang telah disediakan oleh PyTorch
from torchvision import datasets
# melakukan transformasi pada data sebelum dimuat ke dalam model
from torchvision import transforms
```

Input
```
# mengunduh data MNIST
# 'root="."' menunjukkan lokasi dimana dataset akan diunduh
train_data = datasets.MNIST(root=".",
                            train=True,
                            download=True,
# mengubah gambar menjadi tensor
                            transform=transforms.ToTensor()) # do we
want to transform the data as we download it?

# Get the MNIST test dataset
test_data = datasets.MNIST(root=".",
                           train=False,
                           download=True,
                           transform=transforms.ToTensor())
```
Output

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 ./MNIST/raw/train-images-idx3-ubyte.gz
100%|██████████| 9912422/9912422 [00:00<00:00, 97105253.16it/s]
Extracting ./MNIST/raw/train-images-idx3-ubyte.gz to ./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 ./MNIST/raw/train-labels-idx1-ubyte.gz
100%|██████████| 28881/28881 [00:00<00:00, 110223561.26it/s]
Extracting ./MNIST/raw/train-labels-idx1-ubyte.gz to ./MNIST/raw

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 ./MNIST/raw/t10k-images-idx3-ubyte.gz
100%|██████████| 1648877/1648877 [00:00<00:00, 44845194.74it/s]
Extracting ./MNIST/raw/t10k-images-idx3-ubyte.gz to ./MNIST/raw

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 ./MNIST/raw/t10k-labels-idx1-ubyte.gz
100%|██████████| 4542/4542 [00:00<00:00, 21747178.96it/s]
Extracting ./MNIST/raw/t10k-labels-idx1-ubyte.gz to ./MNIST/raw

Input

```
# melatih model untuk evaluasi model pada dataset yang tidak terlihat
train_data, test_data
```

Output

```
(Dataset MNIST
    Number of datapoints: 60000
    Root location: .
    Split: Train
    StandardTransform
 Transform: ToTensor(),
 Dataset MNIST
    Number of datapoints: 10000
    Root location: .
    Split: Test
    StandardTransform
 Transform: ToTensor())
```

Input

```
# memberikan jumlah total sampel yng tersedia dalam dataset pelatihan
dan dataset pengujian, secara berturut-turut
len(train_data), len(test_data)
```

Output

(60000, 10000)

Input

```
# mengakses tensor yang mewakili gambar
img = train_data[0][0]
# mengakses label kelas yang sesuai
label = train_data[0][1]
print(f"Image:\n {img}")
print(f"Label:\n {label}")
```

Output

```
Image:
 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, 0.0118, 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, 0.3176, 0.9412,
0.9922,
          0.9922, 0.4667, 0.0980, 0.0000, 0.0000, 0.0000, 0.0000,
0.0000,
          0.0000, 0.0000, 0.0000, 0.0000],
         [0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
0.0000,
          0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.1765,
0.7294,
          0.9922, 0.9922, 0.5882, 0.1059, 0.0000, 0.0000, 0.0000,
0.0000,
          0.0000, 0.0000, 0.0000, 0.0000],
         [0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
0.0000,
          0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
0.0627,
          0.3647, 0.9882, 0.9922, 0.7333, 0.0000, 0.0000, 0.0000,
0.0000,
          0.0000, 0.0000, 0.0000, 0.0000],
         [0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
0.0000,
          0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
0.0000,
```

```
          0.0000, 0.9765, 0.9922, 0.9765, 0.2510, 0.0000, 0.0000,
0.0000,
          0.0000, 0.0000, 0.0000, 0.0000],
         [0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
0.0000,
          0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.1804,
0.5098,
          0.7176, 0.9922, 0.9922, 0.8118, 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, 0.1529, 0.5804, 0.8980,
0.9922,
          0.9922, 0.9922, 0.9804, 0.7137, 0.0000, 0.0000, 0.0000,
0.0000,
          0.0000, 0.0000, 0.0000, 0.0000],
         [0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
0.0000,
          0.0000, 0.0000, 0.0941, 0.4471, 0.8667, 0.9922, 0.9922,
0.9922,
          0.9922, 0.7882, 0.3059, 0.0000, 0.0000, 0.0000, 0.0000,
0.0000,
          0.0000, 0.0000, 0.0000, 0.0000],
         [0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
0.0000,
          0.0902, 0.2588, 0.8353, 0.9922, 0.9922, 0.9922, 0.9922,
0.7765,
          0.3176, 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, 0.0706,
0.6706,
          0.8588, 0.9922, 0.9922, 0.9922, 0.9922, 0.7647, 0.3137,
0.0353,
          0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
0.0000,
          0.0000, 0.0000, 0.0000, 0.0000],
         [0.0000, 0.0000, 0.0000, 0.0000, 0.2157, 0.6745, 0.8863,
0.9922,
          0.9922, 0.9922, 0.9922, 0.9569, 0.5216, 0.0431, 0.0000,
0.0000,
          0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
0.0000,
          0.0000, 0.0000, 0.0000, 0.0000],
         [0.0000, 0.0000, 0.0000, 0.0000, 0.5333, 0.9922, 0.9922,
0.9922,
          0.8314, 0.5294, 0.5176, 0.0627, 0.0000, 0.0000, 0.0000,
0.0000,
          0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
0.0000,
          0.0000, 0.0000, 0.0000, 0.0000],
         [0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
0.0000,
          0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
0.0000,
          0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
0.0000,
```

```
           0.0000, 0.0000, 0.0000, 0.0000],
          [0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
0.0000,
           0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
0.0000,
           0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
0.0000,
           0.0000, 0.0000, 0.0000, 0.0000],
          [0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
0.0000,
           0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
0.0000,
           0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
0.0000,
           0.0000, 0.0000, 0.0000, 0.0000]]])
Label:
 5
```

Input

```
# menunjukkan bentuk dari tensor gambar
# 'color_channels' adalah jumlah saluran warna
# 'height' adalah tinggi dari gambar dalam piksel
# 'width' adalah lebar dari gambar dalam piksel
print(f"Image shape: {img.shape} -> [color_channels, height, width]
(CHW)")
# menunjukkan label kelas untuk gambar
print(f"Label: {label} -> no shape, due to being integer")
```

Output

```
Image shape: torch.Size([1, 28, 28]) -> [color_channels, height, width]
(CHW)
Label: 5 -> no shape, due to being integer
```

Input

```
# Get the class names from the dataset
# berisi daftar nama kelas secara eksplisit
class_names = train_data.classes
class_names
```

Output

['0 - zero',
 '1 - one',
 '2 - two',
 '3 - three',
 '4 - four',
 '5 - five',
 '6 - six',
 '7 - seven',
 '8 - eight',
 '9 - nine']

6. Visualize at least 5 different samples of the MNIST training dataset.

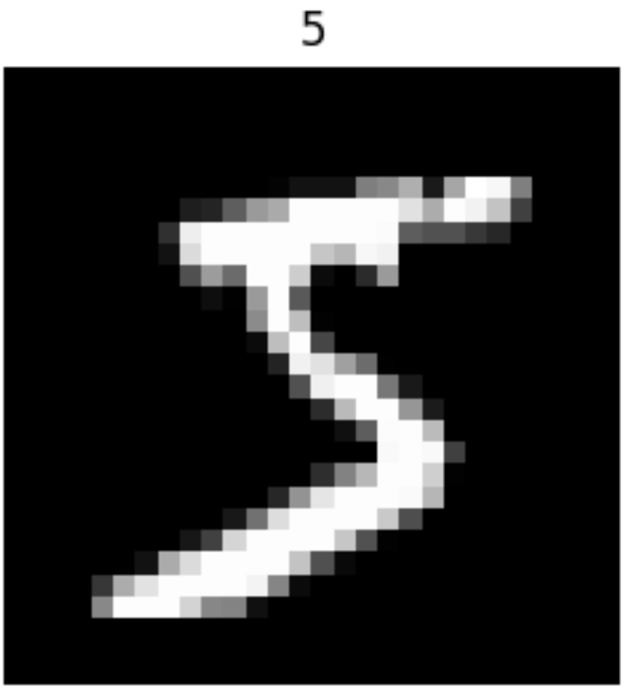
Input

```
import matplotlib.pyplot as plt
# melakukan iterasi sebanyak lima kali
```

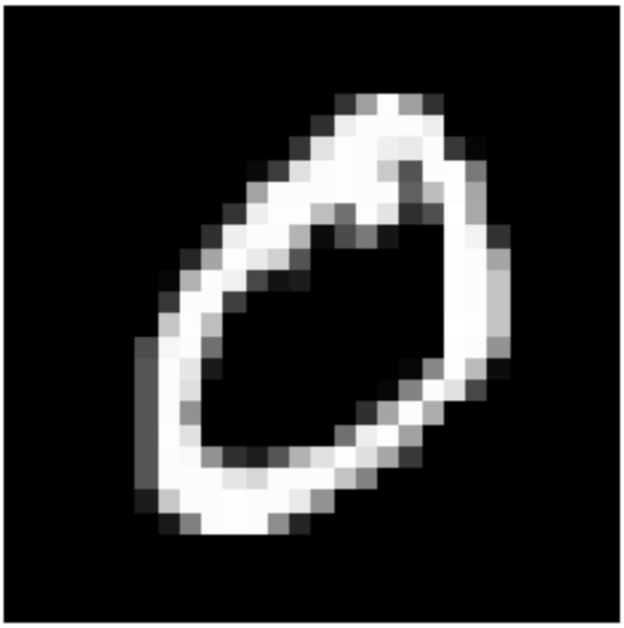
```
for i in range(5):
# mengambil tensor gambar dari elemen ke-i dalam dataset
  img = train_data[i][0]
  print(img.shape)
# menghilangkan dimensi yang bernilai 1 dari tensor gambar
  img_squeeze = img.squeeze()
  print(img_squeeze.shape)
# mengambil label kelas yang sesuai dengan gambar
  label = train_data[i][1]
# menyiapkan plot dengan ukuran tertentu
  plt.figure(figsize=(3, 3))
# menampilkan gambar menggunakan 'imshow' dari matplotlib'
  plt.imshow(img_squeeze, cmap="gray")
# memberikan judul plot dengan label kelas yang sesuai
  plt.title(label)
# menghilagkan sumbu koordinat dari lot untuk memperjelas gambar
  plt.axis(False);
```

Output

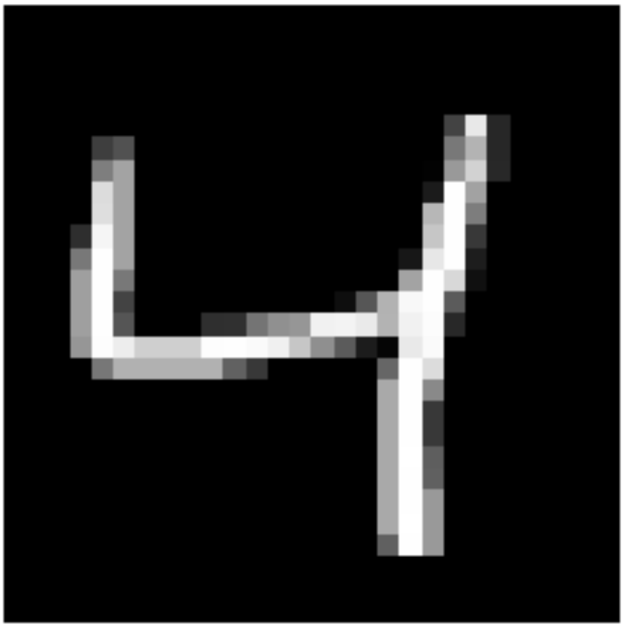
```
torch.Size([1, 28, 28])
torch.Size([28, 28])
torch.Size([1, 28, 28])
torch.Size([28, 28])
torch.Size([1, 28, 28])
torch.Size([28, 28])
torch.Size([1, 28, 28])
torch.Size([28, 28])
torch.Size([1, 28, 28])
torch.Size([28, 28])
```

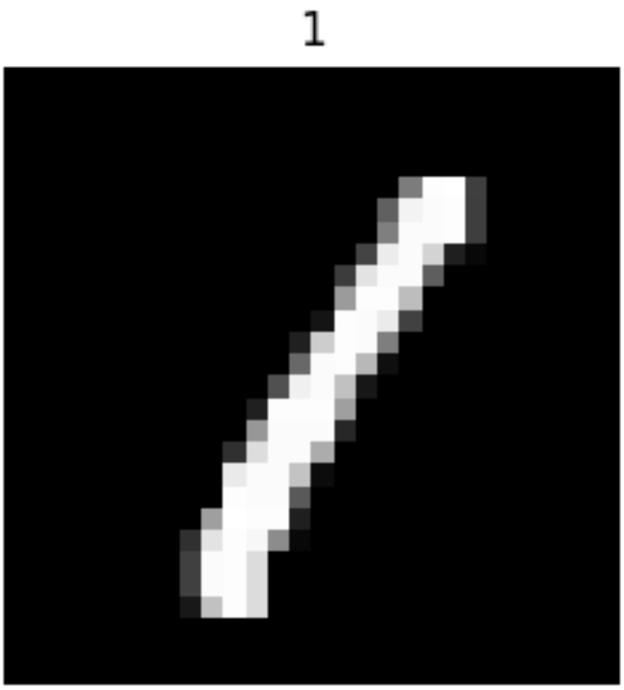


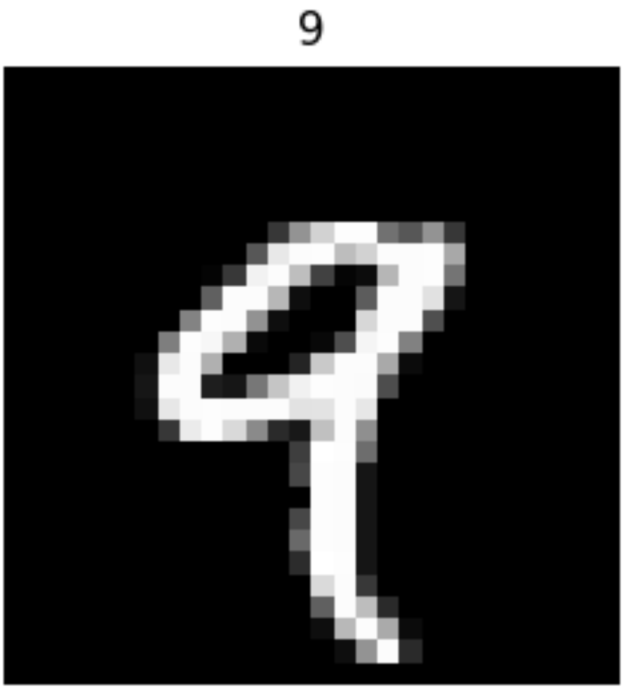
0



4







7. Turn the MNIST train and test datasets into dataloaders using torch.utils.data.DataLoader, set the batch_size=32.

Input

```
# Create train dataloader
from torch.utils.data import DataLoader

train_dataloader = DataLoader(dataset=train_data,
# menunjukkan setiap kali dataloader dipanggil
                              batch_size=32,
# mengacak urutan sampel dalam setiap epoch
```

```
                                   shuffle=True)

test_dataloader = DataLoader(dataset=test_data,
# menunjukkan setiap kali dataloader dipanggil
                             batch_size=32,
# urutan tidak perlu diacak
                             shuffle=False)
```

Input

```
# dataloader untuk dataset pelatihan MNIST
# dataloader untuk dataset pengujian MNIST
train_dataloader, test_dataloader
```

Output

(<torch.utils.data.dataloader.DataLoader at 0x7a8398965390>,
 <torch.utils.data.dataloader.DataLoader at 0x7a8398965180>)

Input

```
# mengambil satu batch pertama dari train_dataloader
for sample in next(iter(train_dataloader)):
  print(sample.shape)
```

Output

```
torch.Size([32, 1, 28, 28])
torch.Size([32])
```

Input

```
# memberikan jumlah total batch untuk dataset pengujian
len(train_dataloader), len(test_dataloader)
```

Output

```
(1875, 313)
```

8. Recreate model_2 used in notebook 03 (the same model from the CNN Explainer website, also known as TinyVGG) capable of fitting on the MNIST dataset.

Input

```
from torch import nn
# mendefinisikan kelas model CNN untuk melakukan prediksi pada dataset
MNIST
class MNIST_model(torch.nn.Module):
  """Model capable of predicting on MNIST dataset.
  """
# inisialisasi yang menerima parameter untuk menentukan struktur model
# 'input_shape' adalah umlah saluran dalam gambar masukan
# 'hidden_units' adalah jumlah unit di lapisan tersembunyi
# 'output_shape' adalah umlah kelas yang akan di prediksi
  def __init__(self, input_shape: int, hidden_units: int, output_shape:
int):
# memanggil konstruktor dari kelas induk
    super().__init__()
# mendefinisikan dua blok konvolusi yang terdiri dari konvolusi 2D,
aktivasi ReLU, dan operasi MacPooling
    self.conv_block_1 = nn.Sequential(
```

```
      nn.Conv2d(in_channels=input_shape,
                out_channels=hidden_units,
                kernel_size=3,
                stride=1,
                padding=1),
      nn.ReLU(),
      nn.Conv2d(in_channels=hidden_units,
                out_channels=hidden_units,
                kernel_size=3,
                stride=1,
                padding=1),
      nn.ReLU(),
      nn.MaxPool2d(kernel_size=2)
    )
    self.conv_block_2 = nn.Sequential(
      nn.Conv2d(in_channels=hidden_units,
                out_channels=hidden_units,
                kernel_size=3,
                stride=1,
                padding=1),
      nn.ReLU(),
      nn.Conv2d(in_channels=hidden_units,
                out_channels=hidden_units,
                kernel_size=3,
                stride=1,
                padding=1),
      nn.ReLU(),
      nn.MaxPool2d(kernel_size=2)
    )
# mendefinisikan blok klasifikasi yang terdiri dari lapisan penggulung
dan lapisan linear untuk menghasilkan keluaran berdasarkan jumlah kelas
yang diinginkan
    self.classifier = nn.Sequential(
      nn.Flatten(),
      nn.Linear(in_features=hidden_units*7*7,
                out_features=output_shape)
    )
# mengatur aliran data melalui model input'x' melewati gambar kedua
blok konvolusi kemudian melalui blok klasifikasi, menghasilkan keluaran
yang mempresentasikan prediksi kelas.
  def forward(self, x):
    x = self.conv_block_1(x)
    # print(f"Output shape of conv block 1: {x.shape}")
    x = self.conv_block_2(x)
    # print(f"Output shape of conv block 2: {x.shape}")
    x = self.classifier(x)
    # print(f"Output shape of classifier: {x.shape}")
    return x
```

Input
```
device
```
Output
```
'cuda'
```

Input
```
# jumlah saluran pada gambar masukan
model = MNIST_model(input_shape=1,
# menunjukkan jumlah unit di lapisan tersembunyi dalam model
                    hidden_units=10,
# menunjukkan jumlah kelas yang akan diprediksi oleh model
                    output_shape=10).to(device)
model
```
Output
```
MNIST_model(
  (conv_block_1): Sequential(
    (0): Conv2d(1, 10, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): ReLU()
    (2): Conv2d(10, 10, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (3): ReLU()
    (4): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (conv_block_2): Sequential(
    (0): Conv2d(10, 10, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): ReLU()
    (2): Conv2d(10, 10, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (3): ReLU()
    (4): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (classifier): Sequential(
    (0): Flatten(start_dim=1, end_dim=-1)
    (1): Linear(in_features=490, out_features=10, bias=True)
  )
)
```

Input
```
# Try a dummy forward pass to see what shapes our data is
dummy_x = torch.rand(size=(1, 28, 28)).unsqueeze(dim=0).to(device)
# dummy_x.shape
model(dummy_x)
```
Output
```
tensor([[ 0.0039, -0.0252, -0.0615,  0.0378,  0.0227, -0.0068, -0.0665, -0.1041,
          0.0609, -0.0055]], device='cuda:0', grad_fn=<AddmmBackward0>)
```

Input
```
dummy_x_2 = torch.rand(size=([1, 10, 7, 7]))
dummy_x_2.shape
```
Output
```
torch.Size([1, 10, 7, 7])
```

Input
```
# menunjukkan tensor multidimensi telah diubah menjadi tensor satu
dimensi
```

```
flatten_layer = nn.Flatten()
flatten_layer(dummy_x_2).shape
```

Output

torch.Size([1, 490])

9. Train the model you built in exercise 8. for 5 epochs on CPU and GPU and see how long it takes on each.

Input

```
%%time
from tqdm.auto import tqdm

# Train on CPU
model_cpu = MNIST_model(input_shape=1,
                        hidden_units=10,
                        output_shape=10).to("cpu")

# Create a loss function and optimizer
loss_fn = nn.CrossEntropyLoss()
optimizer = torch.optim.SGD(model_cpu.parameters(), lr=0.1)

### Training loop
epochs = 5
for epoch in tqdm(range(epochs)):
  train_loss = 0
  for batch, (X, y) in enumerate(train_dataloader):
    model_cpu.train()

    # Put data on CPU
    X, y = X.to("cpu"), y.to("cpu")

    # Forward pass
    y_pred = model_cpu(X)

    # Loss calculation
    loss = loss_fn(y_pred, y)
    train_loss += loss

    # Optimizer zero grad
    optimizer.zero_grad()

    # Loss backward
    loss.backward()

    # Step the optimizer
    optimizer.step()

  # Adjust train loss for number of batches
  train_loss /= len(train_dataloader)

  ### Testing loop
```

```
  test_loss_total = 0

  # Put model in eval mode
  model_cpu.eval()

  # Turn on inference mode
  with torch.inference_mode():
    for batch, (X_test, y_test) in enumerate(test_dataloader):
      # Make sure test data on CPU
      X_test, y_test = X_test.to("cpu"), y_test.to("cpu")
      test_pred = model_cpu(X_test)
      test_loss = loss_fn(test_pred, y_test)

      test_loss_total += test_loss

    test_loss_total /= len(test_dataloader)

  # Print out what's happening
  print(f"Epoch: {epoch} | Loss: {train_loss:.3f} | Test loss:
{test_loss_total:.3f}")
```

Output

100%

5/5 [03:52<00:00, 45.22s/it]

```
Epoch: 0 | Loss: 0.357 | Test loss: 0.064
Epoch: 1 | Loss: 0.072 | Test loss: 0.050
Epoch: 2 | Loss: 0.059 | Test loss: 0.051
Epoch: 3 | Loss: 0.049 | Test loss: 0.045
Epoch: 4 | Loss: 0.045 | Test loss: 0.055
CPU times: user 3min 43s, sys: 1.67 s, total: 3min 45s
Wall time: 3min 52s
```

Input

```
%%time
from tqdm.auto import tqdm

device = "cuda" if torch.cuda.is_available() else "cpu"

# Train on GPU
model_gpu = MNIST_model(input_shape=1,
                        hidden_units=10,
                        output_shape=10).to(device)

# Create a loss function and optimizer
loss_fn = nn.CrossEntropyLoss()
optimizer = torch.optim.SGD(model_gpu.parameters(), lr=0.1)

# Training loop
epochs = 5
for epoch in tqdm(range(epochs)):
  train_loss = 0
```

```
  model_gpu.train()
  for batch, (X, y) in enumerate(train_dataloader):
    # Put data on target device
    X, y = X.to(device), y.to(device)

    # Forward pass
    y_pred = model_gpu(X)

    # Loss calculation
    loss = loss_fn(y_pred, y)
    train_loss += loss

    # Optimizer zero grad
    optimizer.zero_grad()

    # Loss backward
    loss.backward()

    # Step the optimizer
    optimizer.step()

  # Adjust train loss to number of batches
  train_loss /= len(train_dataloader)

  ### Testing loop
  test_loss_total = 0
  # Put model in eval mode and turn on inference mode
  model_gpu.eval()
  with torch.inference_mode():
    for batch, (X_test, y_test) in enumerate(test_dataloader):
      # Make sure test data on target device
      X_test, y_test = X_test.to(device), y_test.to(device)

      test_pred = model_gpu(X_test)
      test_loss = loss_fn(test_pred, y_test)

      test_loss_total += test_loss

    # Adjust test loss total for number of batches
    test_loss_total /= len(test_dataloader)

  # Print out what's happening
  print(f"Epoch: {epoch} | Loss: {train_loss:.3f} | Test loss:
{test_loss_total:.3f}")
```

Output

100%

5/5 [01:00<00:00, 11.97s/it]

```
Epoch: 0 | Loss: 0.261 | Test loss: 0.062
Epoch: 1 | Loss: 0.075 | Test loss: 0.054
```

```
Epoch: 2 | Loss: 0.060 | Test loss: 0.044
Epoch: 3 | Loss: 0.051 | Test loss: 0.043
Epoch: 4 | Loss: 0.046 | Test loss: 0.044
CPU times: user 58.7 s, sys: 627 ms, total: 59.4 s
Wall time: 1min
```

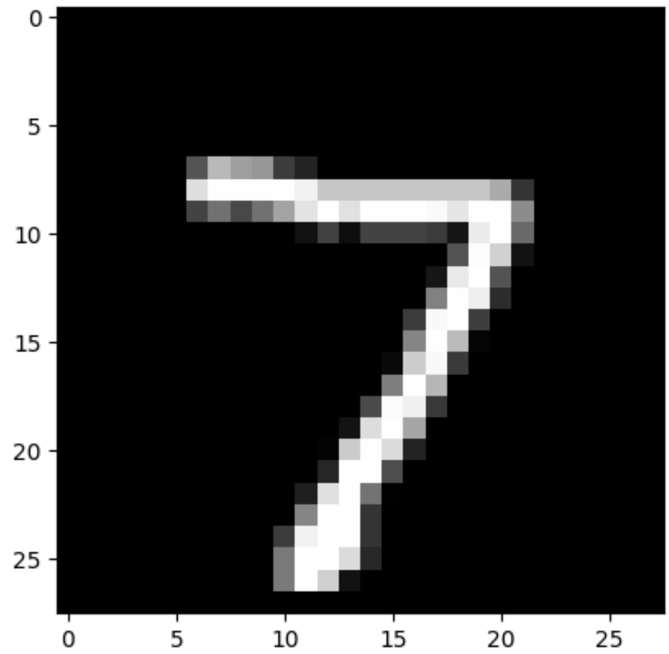
10. Make predictions using your trained model and visualize at least 5 of them comparing the prediciton to the target label.

Input

```
# Make predictions with the trained model
plt.imshow(test_data[0][0].squeeze(), cmap="gray")
```

Output

```
<matplotlib.image.AxesImage at 0x7c3839c658a0>
```



Input

```
# membuat prediksi terhadap gambar pertama
model_pred_logits =
model_gpu(test_data[0][0].unsqueeze(dim=0).to(device))
# mengonversi nilai numerik menjadi distribusi probabilitas untuk
setiap kelas
model_pred_probs = torch.softmax(model_pred_logits, dim=1)
# menghasilkan label prediksi yang sesuai dengan gambar yang diprediksi
model_pred_label = torch.argmax(model_pred_probs, dim=1)
model_pred_label
```

Output

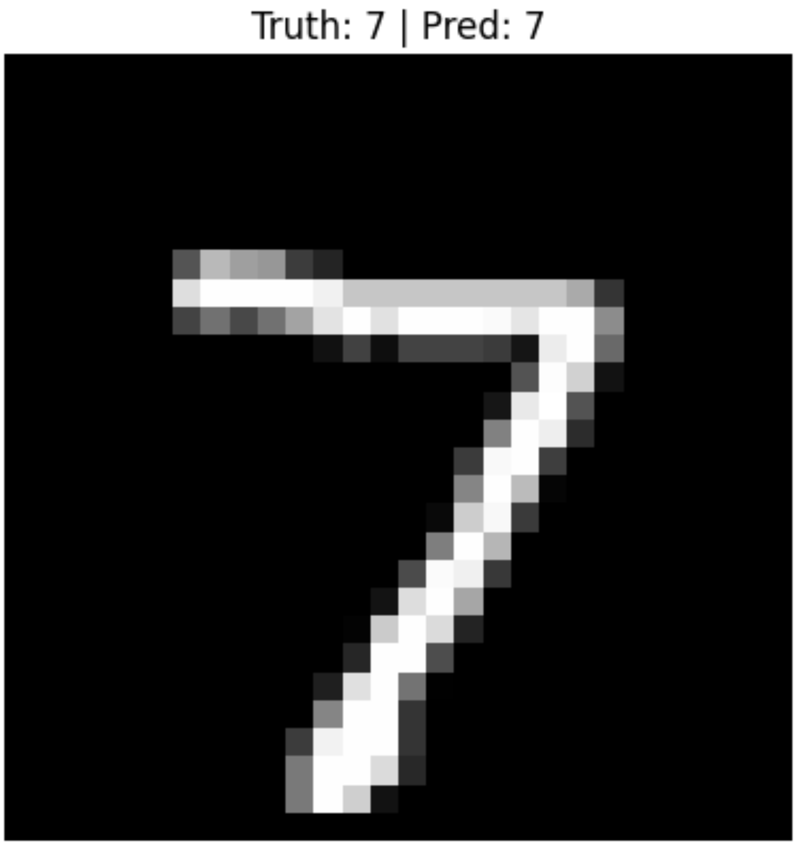
tensor([7], device='cuda:0')

Input

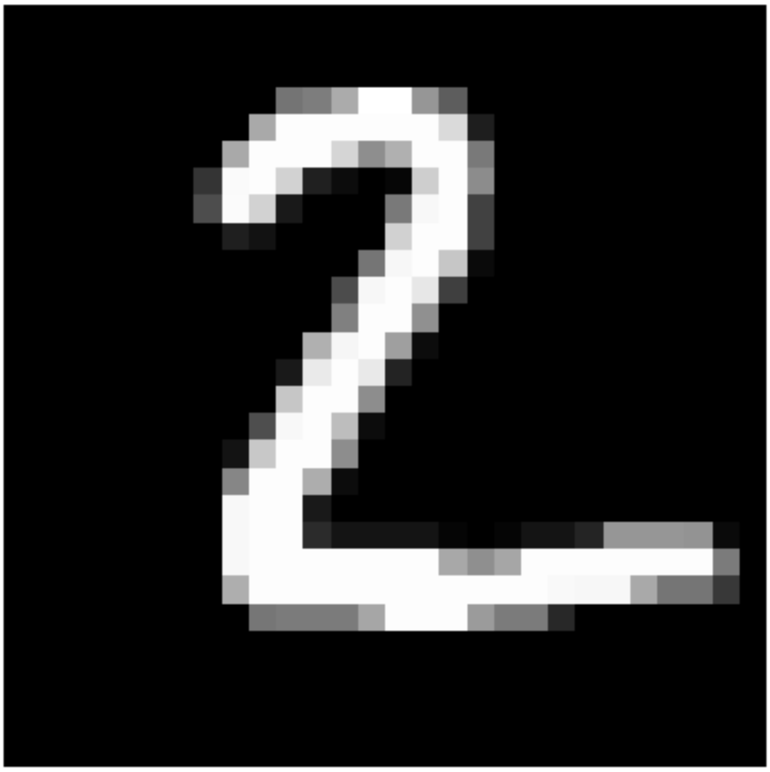
```
# menunjukkan jumlah gambar dari dataset uji yang akan di plot
num_to_plot = 5
# loop yang berjalan sebanyak 'num'
for i in range(num_to_plot):
  # Get image and labels from the test data
```

```
img = test_data[i][0]
label = test_data[i][1]

# Make prediction on image
model_pred_logits = model_gpu(img.unsqueeze(dim=0).to(device))
model_pred_probs = torch.softmax(model_pred_logits, dim=1)
model_pred_label = torch.argmax(model_pred_probs, dim=1)

# Plot the image and prediction
plt.figure()
plt.imshow(img.squeeze(), cmap="gray")
plt.title(f"Truth: {label} | Pred: {model_pred_label.cpu().item()}")
plt.axis(False);
```

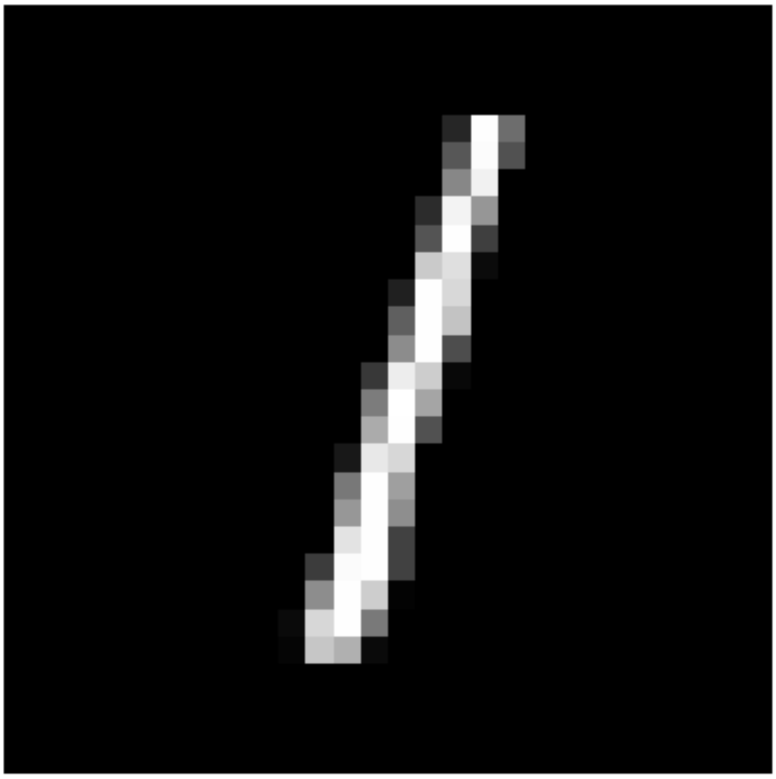
Output



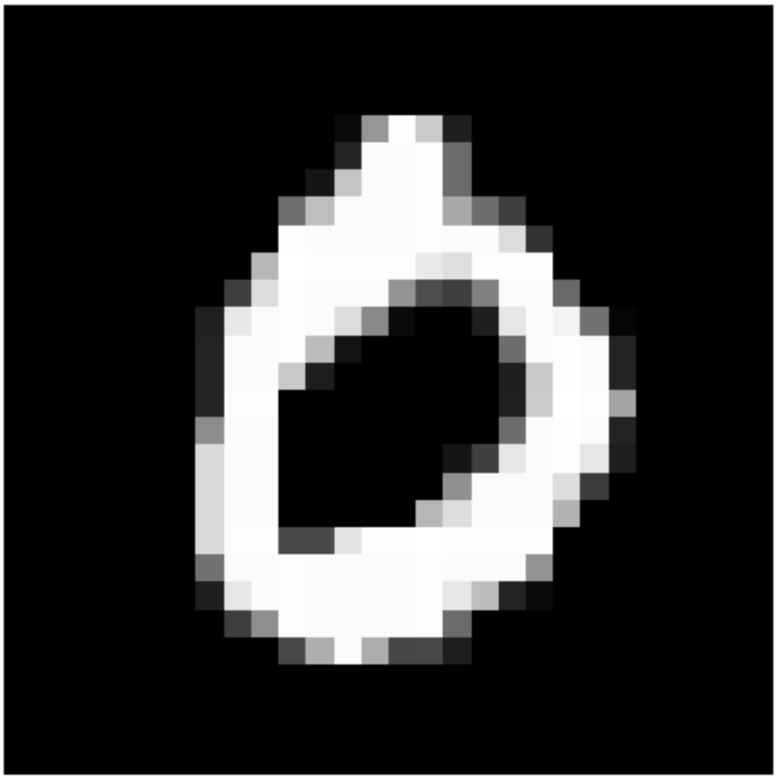
Truth: 2 | Pred: 2



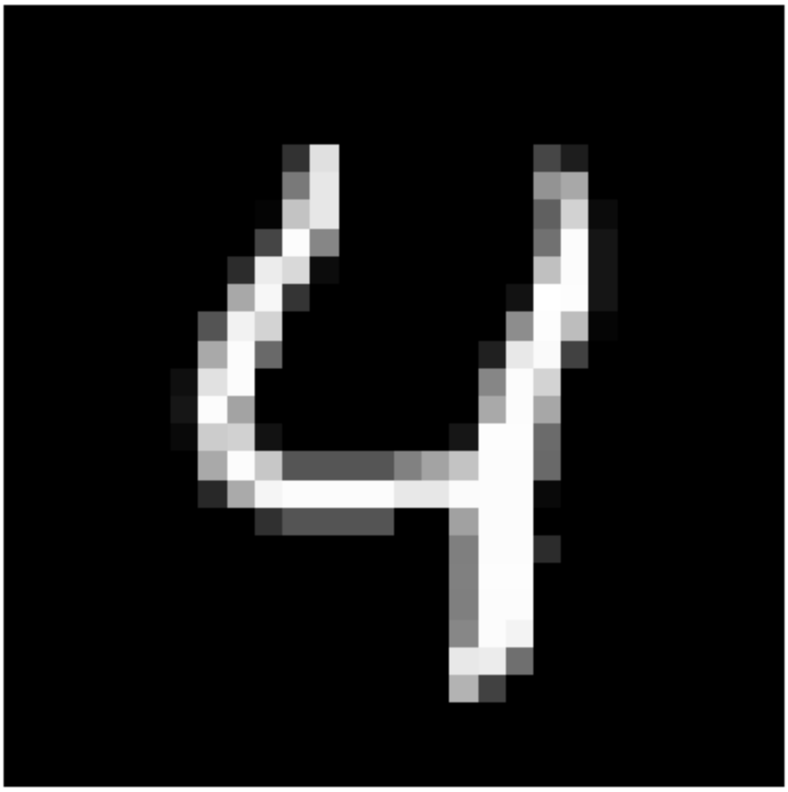
Truth: 1 | Pred: 1



Truth: 0 | Pred: 0



Truth: 4 | Pred: 4



11. Plot a confusion matrix comparing your model's predictions to the truth labels.

Input

```
# See if torchmetrics exists, if not, install it
try:
    import torchmetrics, mlxtend
    print(f"mlxtend version: {mlxtend.__version__}")
    assert int(mlxtend.__version__.split(".")[1]) >= 19, "mlxtend
verison should be 0.19.0 or higher"
except:
    !pip install -q torchmetrics -U mlxtend # <- Note: If you're using
Google Colab, this may require restarting the runtime
    import torchmetrics, mlxtend
    print(f"mlxtend version: {mlxtend.__version__}")
```

Output

806.1/806.1 kB 4.7 MB/s eta 0:00:00

1.4/1.4 MB 25.1 MB/s eta 0:00:00
mlxtend version: 0.23.0

Input

```
# Import mlxtend upgraded version
import mlxtend
print(mlxtend.__version__)
assert int(mlxtend.__version__.split(".")[1]) >= 19 # should be version
0.19.0 or higher
```

Output

```
0.23.0
```

Input

```
# Make predictions across all test data
from tqdm.auto import tqdm
model_gpu.eval()
y_preds = []
with torch.inference_mode():
  for batch, (X, y) in tqdm(enumerate(test_dataloader)):
    # Make sure data on right device
    X, y = X.to(device), y.to(device)
    # Forward pass
    y_pred_logits = model_gpu(X)
    # Logits -> Pred probs -> Pred label
    y_pred_labels = torch.argmax(torch.softmax(y_pred_logits, dim=1),
dim=1)
    # Append the labels to the preds list
    y_preds.append(y_pred_labels)
  y_preds=torch.cat(y_preds).cpu()
len(y_preds)
```

Output

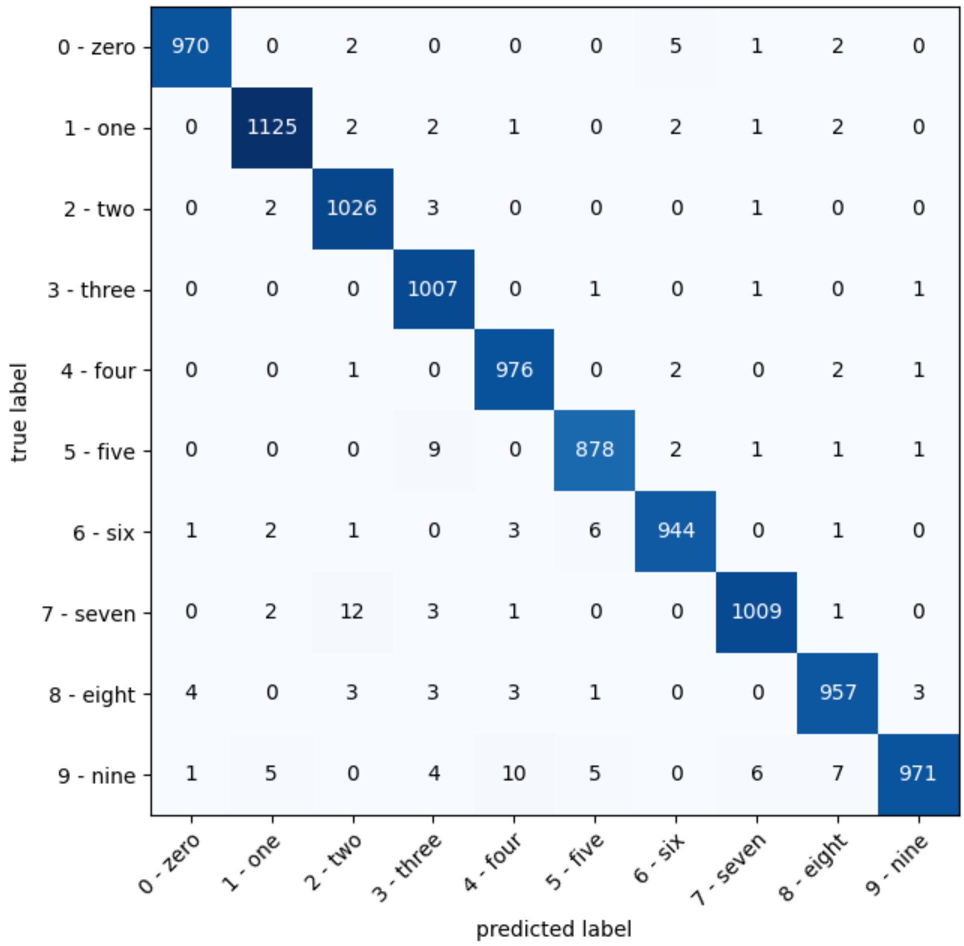
313/? [00:01<00:00, 176.83it/s]

```
10000
```

Input

```
from torchmetrics import ConfusionMatrix
from mlxtend.plotting import plot_confusion_matrix

# Setup confusion matrix
confmat = ConfusionMatrix(task="multiclass",
num_classes=len(class_names))
confmat_tensor = confmat(preds=y_preds,
                         target=test_data.targets)

# Plot the confusion matrix
fix, ax = plot_confusion_matrix(
    conf_mat=confmat_tensor.numpy(),
    class_names=class_names,
    figsize=(10, 7)
)
```

Output



12. Create a random tensor of shape [1, 3, 64, 64] and pass it through a nn.Conv2d() layer with various hyperparameter settings (these can be any settings you choose), what do you notice if the kernel_size parameter goes up and down?

Input

```
# membuat tensor yang berisi bilangan acak dari distribusi
random_tensor = torch.rand([1, 3, 64, 64])
# mengembalikan bentuk dari tensor
random_tensor.shape
```

Output

torch.Size([1, 3, 64, 64])

Input

```
# membuat lapisan konvolusi dengan menggunakan modul dari PyTorch
# 'in_channels=3' adalah jumlah channel masukan, dalam hal ini adalah 3
channel warna (RGB)
conv_layer = nn.Conv2d(in_channels=3,
#jumlah channel keluaran dari konvolusi, diatur menjadi 64
                       out_channels=64,
# ukuran kernel konvolusi, dalam hal ini adalah kernel 3x3
                       kernel_size=3,
# langkah atau pergeseran dari kernel selama proses konvolusi
                       stride=2,
# nilai padding yang ditambahkan di sekitar ambar input
                       padding=1)

# mencetak bentuk dari tensor acak sebelum memulai lapisan konvolusi
print(f"Random tensor original shape: {random_tensor.shape}")
# memasukkan tensor acak ke dalam lapisan konvolusi yang telah dibuat
sebelumnya
random_tensor_through_conv_layer = conv_layer(random_tensor)
# mencetak bentuk dari tensor acak setelah melewati lapisan konvolusi
print(f"Random tensor through conv layer shape:
{random_tensor_through_conv_layer.shape}")
```

Output

```
Random tensor original shape: torch.Size([1, 3, 64, 64])
Random tensor through conv layer shape: torch.Size([1, 64, 32, 32])
```

13. Use a model similar to the trained model_2 from notebook 03 to make predictions on the test torchvision.datasets.FashionMNIST dataset

Input

```
# Download FashionMNIST train & test
from torchvision import datasets
from torchvision import transforms

fashion_mnist_train = datasets.FashionMNIST(root=".",
                                            download=True,
                                            train=True,
```

```
transform=transforms.ToTensor())

fashion_mnist_test = datasets.FashionMNIST(root=".",
                                            train=False,
                                            download=True,

transform=transforms.ToTensor())

len(fashion_mnist_train), len(fashion_mnist_test)
```

Output

Downloading http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/train-images-idx3-ubyte.gz
Downloading http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/train-images-idx3-ubyte.gz
to ./FashionMNIST/raw/train-images-idx3-ubyte.gz
100%|██████████| 26421880/26421880 [00:01<00:00, 16036127.71it/s]
Extracting ./FashionMNIST/raw/train-images-idx3-ubyte.gz to ./FashionMNIST/raw

Downloading http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/train-labels-idx1-ubyte.gz
Downloading http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/train-labels-idx1-ubyte.gz
to ./FashionMNIST/raw/train-labels-idx1-ubyte.gz
100%|██████████| 29515/29515 [00:00<00:00, 264349.53it/s]
Extracting ./FashionMNIST/raw/train-labels-idx1-ubyte.gz to ./FashionMNIST/raw

Downloading http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/t10k-images-idx3-ubyte.gz
Downloading http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/t10k-images-idx3-ubyte.gz
to ./FashionMNIST/raw/t10k-images-idx3-ubyte.gz
100%|██████████| 4422102/4422102 [00:00<00:00, 5087724.38it/s]
Extracting ./FashionMNIST/raw/t10k-images-idx3-ubyte.gz to ./FashionMNIST/raw

Downloading http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/t10k-labels-idx1-ubyte.gz
Downloading http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/t10k-labels-idx1-ubyte.gz
to ./FashionMNIST/raw/t10k-labels-idx1-ubyte.gz
100%|██████████| 5148/5148 [00:00<00:00, 15958815.22it/s]
Extracting ./FashionMNIST/raw/t10k-labels-idx1-ubyte.gz to ./FashionMNIST/raw

(60000, 10000)

Input

```
# Get the class names of the Fashion MNIST dataset
fashion_mnist_class_names = fashion_mnist_train.classes
fashion_mnist_class_names
```

Output

['T-shirt/top',
 'Trouser',
 'Pullover',
 'Dress',
 'Coat',
 'Sandal',
 'Shirt',
 'Sneaker',
 'Bag',
 'Ankle boot']

Input

```
# Turn FashionMNIST datasets into dataloaders
from torch.utils.data import DataLoader

fashion_mnist_train_dataloader = DataLoader(fashion_mnist_train,
                                            batch_size=32,
                                            shuffle=True)

fashion_mnist_test_dataloader = DataLoader(fashion_mnist_test,
                                           batch_size=32,
                                           shuffle=False)

len(fashion_mnist_train_dataloader), len(fashion_mnist_test_dataloader)
```

Output

```
(1875, 313)
```

Input

```
# model_2 is the same architecture as MNIST_model
model_2 = MNIST_model(input_shape=1,
                      hidden_units=10,
                      output_shape=10).to(device)
model_2
```

Output

```
MNIST_model(
  (conv_block_1): Sequential(
    (0): Conv2d(1, 10, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): ReLU()
    (2): Conv2d(10, 10, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (3): ReLU()
    (4): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (conv_block_2): Sequential(
    (0): Conv2d(10, 10, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): ReLU()
    (2): Conv2d(10, 10, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (3): ReLU()
    (4): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (classifier): Sequential(
    (0): Flatten(start_dim=1, end_dim=-1)
    (1): Linear(in_features=490, out_features=10, bias=True)
  )
)
```

Input

```
# Setup metrics
from tqdm.auto import tqdm
from torchmetrics import Accuracy

acc_fn =
Accuracy(num_classes=len(fashion_mnist_class_names)).to(device)
```

```
# Setup training/testing loop
epochs = 5
for epoch in tqdm(range(epochs)):
  train_loss, test_loss_total = 0, 0
  train_acc, test_acc = 0, 0

  ### Training
  model_2.train()
  for batch, (X_train, y_train) in
enumerate(fashion_mnist_train_dataloader):
    X_train, y_train = X_train.to(device), y_train.to(device)

    # Forward pass and loss
    y_pred = model_2(X_train)
    loss = loss_fn(y_pred, y_train)
    train_loss += loss
    train_acc += acc_fn(y_pred, y_train)

    # Backprop and gradient descent
    optimizer.zero_grad()
    loss.backward()
    optimizer.step()

  # Adjust the loss/acc (find the loss/acc per epoch)
  train_loss /= len(fashion_mnist_train_dataloader)
  train_acc /= len(fashion_mnist_train_dataloader)

  ### Testing
  model_2.eval()
  with torch.inference_mode():
    for batch, (X_test, y_test) in
enumerate(fashion_mnist_test_dataloader):
      X_test, y_test = X_test.to(device), y_test.to(device)

      # Forward pass and loss
      y_pred_test = model_2(X_test)
      test_loss = loss_fn(y_pred_test, y_test)
      test_loss_total += test_loss

      test_acc += acc_fn(y_pred_test, y_test)

    # Adjust the loss/acc (find the loss/acc per epoch)
    test_loss /= len(fashion_mnist_test_dataloader)
    test_acc /= len(fashion_mnist_test_dataloader)

  # Print out what's happening
```

```
  print(f"Epoch: {epoch} | Train loss: {train_loss:.3f} | Train acc:
{train_acc:.2f} | Test loss: {test_loss_total:.3f} | Test acc:
{test_acc:.2f}")
```

Input

```
# Make predictions with trained model_2
test_preds = []
model_2.eval()
with torch.inference_mode():
  for X_test, y_test in tqdm(fashion_mnist_test_dataloader):
    y_logits = model_2(X_test.to(device))
    y_pred_probs = torch.softmax(y_logits, dim=1)
    y_pred_labels = torch.argmax(y_pred_probs, dim=1)
    test_preds.append(y_pred_labels)
test_preds = torch.cat(test_preds).cpu() # matplotlib likes CPU
test_preds[:10], len(test_preds)
```

Input

```
# Get wrong prediction indexes
import numpy as np
wrong_pred_indexes = np.where(test_preds !=
fashion_mnist_test.targets)[0]
len(wrong_pred_indexes)
```

Input

```
# Select random 9 wrong predictions and plot them
import random
random_selection = random.sample(list(wrong_pred_indexes), k=9)

plt.figure(figsize=(10, 10))
for i, idx in enumerate(random_selection):
  # Get true and pred labels
  true_label = fashion_mnist_class_names[fashion_mnist_test[idx][1]]
  pred_label = fashion_mnist_class_names[test_preds[idx]]

  # Plot the wrong prediction with its original label
  plt.subplot(3, 3, i+1)
  plt.imshow(fashion_mnist_test[idx][0].squeeze(), cmap="gray")
  plt.title(f"True: {true_label} | Pred: {pred_label}", c="r")
  plt.axis(False);
```