```
pip install torch
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Requirement already satisfied: torch in /usr/local/lib/python3.8/dist-packages (1.13.1+cu116)
Requirement already satisfied: typing-extensions in /usr/local/lib/python3.8/dist-packages (from torch) (4.5.0)
```

```
pip install torchvision
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Requirement already satisfied: torchvision in /usr/local/lib/python3.8/dist-packages (0.14.1+cu116)
Requirement already satisfied: requests in /usr/local/lib/python3.8/dist-packages (from torchvision) (2.25.1)
Requirement already satisfied: typing-extensions in /usr/local/lib/python3.8/dist-packages (from torchvision) (4.5.0)
Requirement already satisfied: pillow!=8.3.*,>=5.3.0 in /usr/local/lib/python3.8/dist-packages (from torchvision) (8.4.0)
Requirement already satisfied: torch==1.13.1 in /usr/local/lib/python3.8/dist-packages (from torchvision) (1.13.1+cu116)
Requirement already satisfied: numpy in /usr/local/lib/python3.8/dist-packages (from torchvision) (1.22.4)
Requirement already satisfied: urllib3<1.27,>=1.21.1 in /usr/local/lib/python3.8/dist-packages (from requests->torchvision) (1.24.3)
Requirement already satisfied: idna<3,>=2.5 in /usr/local/lib/python3.8/dist-packages (from requests->torchvision) (2.10)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.8/dist-packages (from requests->torchvision) (2022.12.7)
Requirement already satisfied: chardet<5,>=3.0.2 in /usr/local/lib/python3.8/dist-packages (from requests->torchvision) (4.0.0)
```

```
import torch, torchvision
from torch import nn
from torch import optim
from torchvision.transforms import ToTensor
import torch.nn.functional as A
import matplotlib.pyplot as plt
import requests
from PIL import Image
from io import BytesIO

import copy
from sklearn.metrics import confusion_matrix
import pandas as pd
import numpy as np


#Setting the batchsize to 64
numb_batch = 64


# Getting data
T = torchvision.transforms.Compose([
    torchvision.transforms.ToTensor()
])
train_data = torchvision.datasets.MNIST('mnist_data', train=True, download=True, transform=T)
val_data = torchvision.datasets.MNIST('mnist_data', train=False, download=True, transform=T)

train_dl = torch.utils.data.DataLoader(train_data, batch_size = numb_batch)
val_dl = torch.utils.data.DataLoader(val_data, batch_size = numb_batch)


#creating the model
def create_lenet():
    model = nn.Sequential(
        nn.Conv2d(1, 6, 5, padding=2),
        nn.ReLU(),
        nn.AvgPool2d(2, stride=2),
        nn.Conv2d(6, 16, 5, padding=0),
        nn.ReLU(),
        nn.AvgPool2d(2, stride=2),
        nn.Flatten(),
        nn.Linear(400, 120),
        nn.ReLU(),
        nn.Linear(120, 84),
        nn.ReLU(),
        nn.Linear(84, 10)
    )
    return model


# Validating the model
def validate(model, data):
    total = 0
    correct = 0
    for i, (images, labels) in enumerate(data):
        images = images.cuda()
        x = model(images)
```

```
            value, pred = torch.max(x,1)
            pred = pred.data.cpu()
            total += x.size(0)
            correct += torch.sum(pred == labels)
        return correct*100./total


    # Training function
    def train(numb_epoch=3, lr=1e-3, device="cpu"):
        accuracies = []
        cnn = create_lenet().to(device)
        cec = nn.CrossEntropyLoss()
        optimizer = optim.Adam(cnn.parameters(), lr=lr)
        max_accuracy = 0
        for epoch in range(numb_epoch):
            for i, (images, labels) in enumerate(train_dl):
                images = images.to(device)
                labels = labels.to(device)
                optimizer.zero_grad()
                pred = cnn(images)
                loss = cec(pred, labels)
                loss.backward()
                optimizer.step()
            accuracy = float(validate(cnn, val_dl))
            accuracies.append(accuracy)
            if accuracy > max_accuracy:
                best_model = copy.deepcopy(cnn)
                max_accuracy = accuracy
                print("Saving Best Model with Accuracy: ", accuracy)
            print('Epoch:', epoch+1, "Accuracy :", accuracy, '%')
        plt.plot(accuracies)
        return best_model


    # GPU Availability
    if torch.cuda.is_available():
        device = torch.device("cuda:0")
    else:
        device = torch.device("cpu")
        print("No Cuda Available")
    device
```

```
     device(type='cuda', index=0)
```

```
    # Training the model
    lenet = train(40, device=device)
```

```
Saving Best Model with Accuracy:  95.73999786376953
Epoch: 1 Accuracy : 95.73999786376953 %
Saving Best Model with Accuracy:  97.41999816894531
Epoch: 2 Accuracy : 97.41999816894531 %
Saving Best Model with Accuracy:  97.73999786376953
Epoch: 3 Accuracy : 97.73999786376953 %
Saving Best Model with Accuracy:  98.01000213623047
Epoch: 4 Accuracy : 98.01000213623047 %
Saving Best Model with Accuracy:  98.33999633789062
Epoch: 5 Accuracy : 98.33999633789062 %
Epoch: 6 Accuracy : 98.27999877929688 %
Epoch: 7 Accuracy : 98.16999816894531 %
Epoch: 8 Accuracy : 97.93000030517578 %
Epoch: 9 Accuracy : 98.12000274658203 %
Saving Best Model with Accuracy:  98.80999755859375
Epoch: 10 Accuracy : 98.80999755859375 %
Epoch: 11 Accuracy : 98.44000244140625 %
Epoch: 12 Accuracy : 98.76000213623047 %
Epoch: 13 Accuracy : 98.72000122070312 %
Epoch: 14 Accuracy : 98.69000244140625 %
Saving Best Model with Accuracy:  98.83000183105469
Epoch: 15 Accuracy : 98.83000183105469 %
Epoch: 16 Accuracy : 98.70999908447266 %
Saving Best Model with Accuracy:  98.91999816894531
Epoch: 17 Accuracy : 98.91999816894531 %
Epoch: 18 Accuracy : 98.69999694824219 %
Epoch: 19 Accuracy : 98.86000061035156 %
Epoch: 20 Accuracy : 98.91999816894531 %
Epoch: 21 Accuracy : 98.86000061035156 %
Saving Best Model with Accuracy:  98.98999786376953
Epoch: 22 Accuracy : 98.98999786376953 %
Epoch: 23 Accuracy : 98.91000366210938 %
Epoch: 24 Accuracy : 98.91999816894531 %
```

```
# Saving the model
torch.save(lenet.state_dict(), "lenet.pth")
```

```
Epoch: 28 Accuracy : 98.97000122070312 %
```

```
#  Loading the saved model
lenet = create_lenet().to(device)
lenet.load_state_dict(torch.load("lenet.pth"))
lenet.eval()
```

```
Sequential(
  (0): Conv2d(1, 6, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2))
  (1): ReLU()
  (2): AvgPool2d(kernel_size=2, stride=2, padding=0)
  (3): Conv2d(6, 16, kernel_size=(5, 5), stride=(1, 1))
  (4): ReLU()
  (5): AvgPool2d(kernel_size=2, stride=2, padding=0)
  (6): Flatten(start_dim=1, end_dim=-1)
  (7): Linear(in_features=400, out_features=120, bias=True)
  (8): ReLU()
  (9): Linear(in_features=120, out_features=84, bias=True)
  (10): ReLU()
  (11): Linear(in_features=84, out_features=10, bias=True)
)
```

```
# Creating the function to test validation data
def predict_dl(model, data):
    y_pred = []
    y_true = []
    for i, (images, labels) in enumerate(data):
        images = images.cuda()
        x = model(images)
        value, pred = torch.max(x, 1)
        pred = pred.data.cpu()
        y_pred.extend(list(pred.numpy()))
        y_true.extend(list(labels.numpy()))
    return np.array(y_pred), np.array(y_true)
```

```
y_pred, y_true = predict_dl(lenet, val_dl)
```

```
# confusion matrix
pd.DataFrame(confusion_matrix(y_true, y_pred, labels=np.arange(0,10)))
```

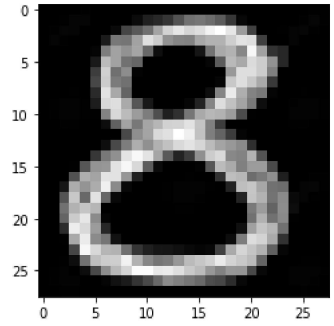
| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 978 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 |
| **1** | 0 | 1133 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 |
| **2** | 0 | 1 | 1029 | 0 | 0 | 0 | 0 | 1 | 1 | 0 |
| **3** | 0 | 0 | 1 | 1001 | 0 | 2 | 0 | 1 | 4 | 1 |
| **4** | 0 | 0 | 1 | 0 | 976 | 0 | 2 | 0 | 0 | 3 |
| **5** | 1 | 0 | 0 | 7 | 0 | 883 | 1 | 0 | 0 | 0 |
| **6** | 2 | 3 | 0 | 1 | 0 | 4 | 947 | 0 | 1 | 0 |
| **7** | 0 | 1 | 3 | 0 | 1 | 0 | 0 | 1021 | 1 | 1 |
| **8** | 3 | 0 | 2 | 1 | 0 | 2 | 0 | 0 | 966 | 0 |

```
# defining function to get prediction
def inference(path, model, device):
    r = requests.get(path)
    with BytesIO(r.content) as f:
        img = Image.open(f).convert(mode="L")
        img = img.resize((28, 28))
        x = (255 - np.expand_dims(np.array(img), -1))/255.
    with torch.no_grad():
        pred = model(torch.unsqueeze(T(x), axis=0).float().to(device))
        return F.softmax(pred, dim=-1).cpu().numpy()
```

```
# getting the image from web
path = "https://previews.123rf.com/images/aroas/aroas1704/aroas170400068/79321959-handwritten-sketch-black-number-8-on-white-background.jpg"
r = requests.get(path)
with BytesIO(r.content) as f:
    img = Image.open(f).convert(mode="L")
    img = img.resize((28, 28))
x = (255 - np.expand_dims(np.array(img), -1))/255.
```

```
# showing the image
plt.imshow(x.squeeze(-1), cmap="gray")
```

```
<matplotlib.image.AxesImage at 0x7f72a5f887c0>
```




```
# predictions
pred = inference(path, lenet, device=device)
pred_idx = np.argmax(pred)
print(f"Predicted: {pred_idx}, Prob: {pred[0][pred_idx]*100} %")
```

```
Predicted: 8, Prob: 88.76972198486328 %
```

```
pred
```

```
array([[4.5617226e-09, 1.6272187e-08, 2.4771585e-09, 7.6190181e-02,
        3.9601301e-11, 3.6106311e-02, 6.2214353e-06, 3.2907920e-08,
        8.8769722e-01, 1.9174619e-10]], dtype=float32)
```