Experiment report: Pytorch

Done by student: **Bylkova Kristina (伯汀娜), 1820249049**

Course: **Big Data Analysis Technology**

Date: **October 2024**

MindMap

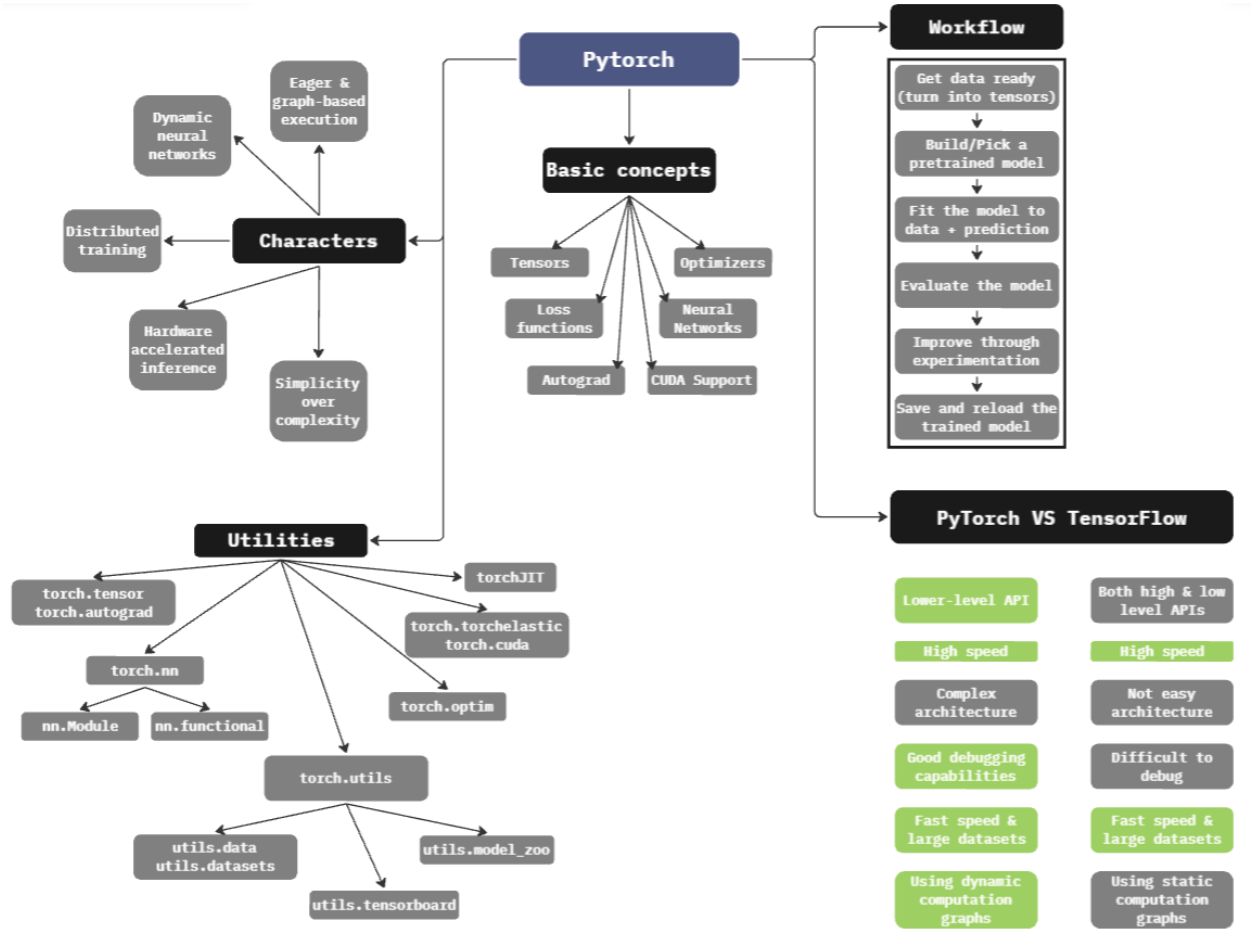

Figure 1: Mindmap

Experiment 1

I used Kaggle to run and edit the sample code:

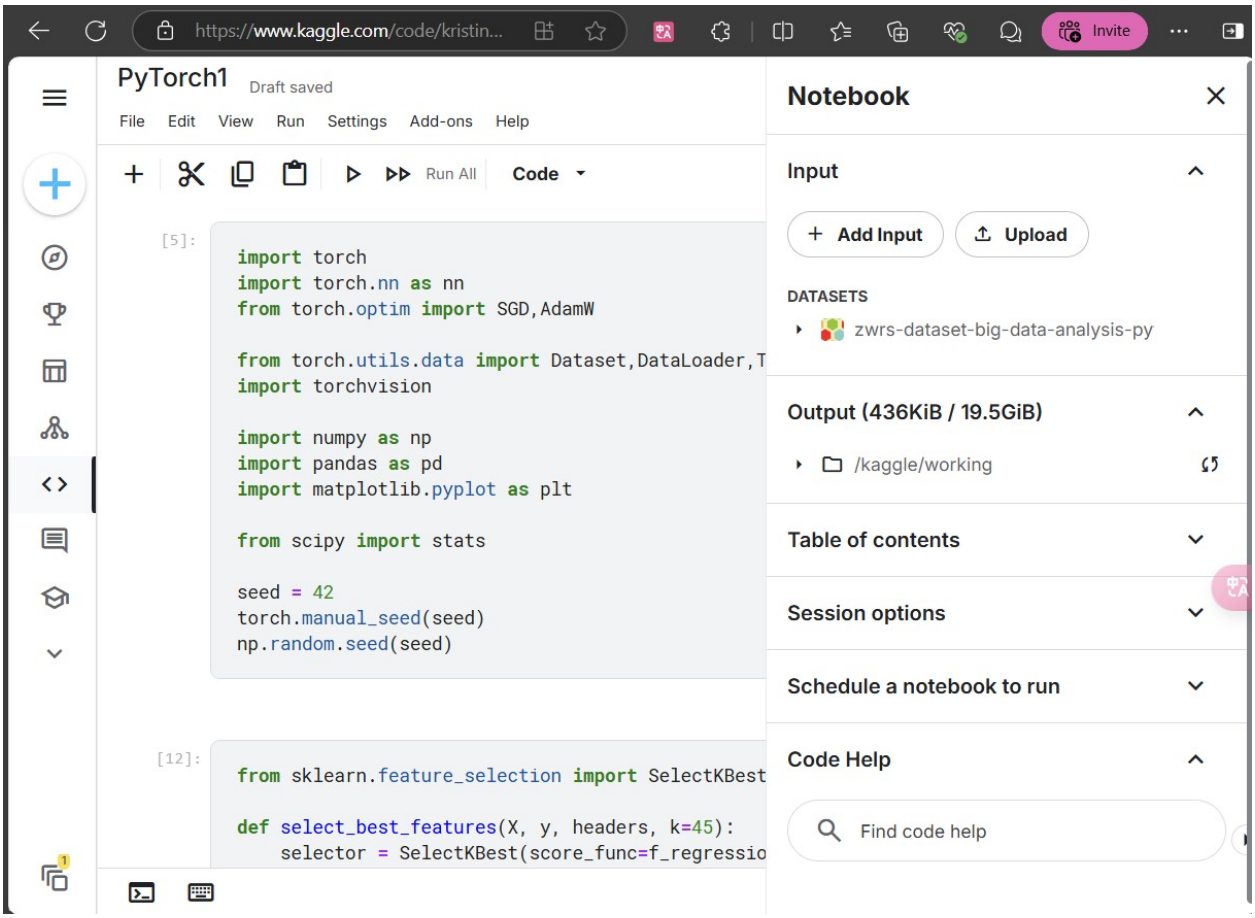

Figure 2: Notebook on Kaggle

Due to this experiment, I needed to:

1. solve a regression problem (covid-19 cases prediction) with deep neural network (DNN);
2. understand the basic DNN training steps;
3. get familiar with PyTorch.

Task 1

Try different activation function (ReLU, Softmax and Sigmoid)

ReLU:

```
class my_NN(nn.Module):
    def __init__(self):
```

```
        super().__init__()
        self.Matrix1 = nn.Linear(45,16)
        self.Matrix2 = nn.Linear(16,8)
        self.Matrix3 = nn.Linear(8,1)
        self.R = nn.ReLU()

    def forward(self,x):
        x = self.R(self.Matrix1(x))
        x = self.R(self.Matrix2(x))
        x = self.Matrix3(x)

        return x.squeeze()
```

Mean Squared Error (MSE) Loss: 56.578332726102545

Softmax:

```
class my_NN(nn.Module):
    def __init__(self):
        super().__init__()
        self.Matrix1 = nn.Linear(45,16)
        self.Matrix2 = nn.Linear(16,8)
        self.Matrix3 = nn.Linear(8,1)
        self.R = nn.Softmax()

    def forward(self,x):
        x = self.R(self.Matrix1(x))
        x = self.R(self.Matrix2(x))
        x = self.Matrix3(x)

        return x.squeeze()
```

Mean Squared Error (MSE) Loss: 39.02215756660381

Sigmoid:

```
class my_NN(nn.Module):
    def __init__(self):
        super().__init__()
        self.Matrix1 = nn.Linear(45,16)
        self.Matrix2 = nn.Linear(16,8)
        self.Matrix3 = nn.Linear(8,1)
        self.R = nn.Sigmoid()

    def forward(self,x):
        x = self.R(self.Matrix1(x))
        x = self.R(self.Matrix2(x))
        x = self.Matrix3(x)

        return x.squeeze()
```

Mean Squared Error (MSE) Loss: 4.155347884435653

Task 2

Try different network design

For example, adding more layers:

```
class my_novel_NN1(nn.Module):
    def __init__(self):
        super().__init__()
        self.Matrix1 = nn.Linear(45,16)
        self.Matrix2 = nn.Linear(32,16)
        self.Matrix3 = nn.Linear(16,8)
        self.R = nn.ReLU()
        self.Matrix4 = nn.Linear(8,4)
        self.Matrix5 = nn.Linear(4,2)
        self.Matrix6 = nn.Linear(2,1)
        self.R = nn.ReLU()

    def forward(self,x):
        x = self.R1(self.Matrix1(x))
        x = self.R1(self.Matrix2(x))
        x = self.R1(self.Matrix3(x))
        x = self.R2(self.Matrix4(x))
        x = self.R2(self.Matrix5(x))
        x = self.Matrix6(x)

        return x.squeeze()
```

Mean Squared Error (MSE) Loss: 56.578332726102545

For example, changing to a new activation function:

```
class my_novel_NN2(nn.Module):
    def __init__(self):
        super().__init__()
        self.Matrix1 = nn.Linear(45,16)
        self.Matrix2 = nn.Linear(32,16)
        self.Matrix3 = nn.Linear(16,8)
        self.R = nn.LeakyReLU()
        self.Matrix4 = nn.Linear(8,4)
        self.Matrix5 = nn.Linear(4,2)
        self.Matrix6 = nn.Linear(2,1)
        self.R = nn.LeakyReLU()

    def forward(self,x):
        x = self.R1(self.Matrix1(x))
        x = self.R1(self.Matrix2(x))
        x = self.R1(self.Matrix3(x))
        x = self.R2(self.Matrix4(x))
        x = self.R2(self.Matrix5(x))
        x = self.Matrix6(x)

        return x.squeeze()
```

Mean Squared Error (MSE) Loss: 56.57393866294657

Task 3

Try to implement a validate phase for the network

For this task, I implemented the validate phase by:

1. Splitting the training data before into training data and testing data:

```
class my_dataset(Dataset):
def __init__(self, filepath, phase, selected_headers):
    self.data = pd.read_csv(filepath, header = 0)
    self.data = self.data.drop(labels = ["id"], axis =1)
    self.phase = phase
    if self.phase == "train":
        self.data = self.data.iloc[100:]
    elif self.phase == "validate":
        self.data = self.data.iloc[:100]
    self.headers = self.data.columns.tolist()

    self.x = torch.tensor(self.data[selected_headers].values)
    self.y = torch.tensor(self.data[self.headers[-1]].values)

def __len__(self):
    return len(self.data)

def __getitem__(self, idx):
    return self.x[idx].double(),self.y[idx].double()

train_data = my_dataset("/kaggle/input/zwrs-dataset-big-data-analysis-pytorch/covid.train.csv"
, phase = "train", selected_headers)
val_data = my_dataset("/kaggle/input/zwrs-dataset-big-data-analysis-pytorch/covid.train.csv",
 phase = "validate", selected_headers)

```

2. Using validate dataloader

```
train_dl = DataLoader(train_data, batch_size=256, shuffle=True)
val_dl = DataLoader(val_data, batch_size=25, shuffle=True)

```

3. Adding the validate phase into the training phase

```
total_val_loss = 0
with torch.no_grad():
    for x, y in val_dl:
        valid_loss = L(f(x), y, f)
        total_val_loss += valid_loss.item()
val_losses.append(total_val_loss / len(val_dl))

epochs.append(epoch)
losses.append(epoch_loss / N)

```

Experiment 2

In this experiment, I needed to learn how to develop a digit recognition model using PyTorch and the MNIST dataset.

1. Importing PyTorch Modules for Image Data

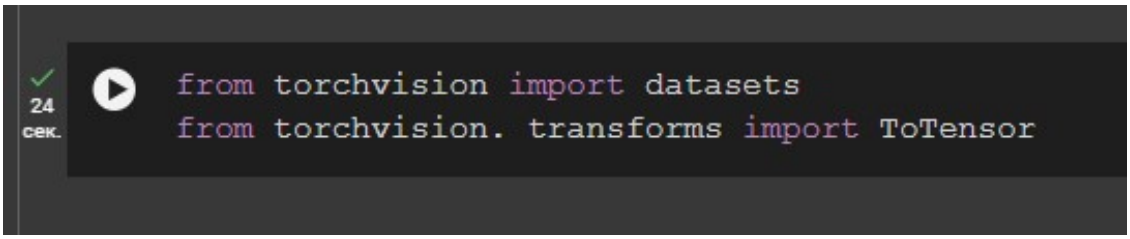

Figure 3: Step 1

2. Data Preparation (Downloading the dataset and splitting it)

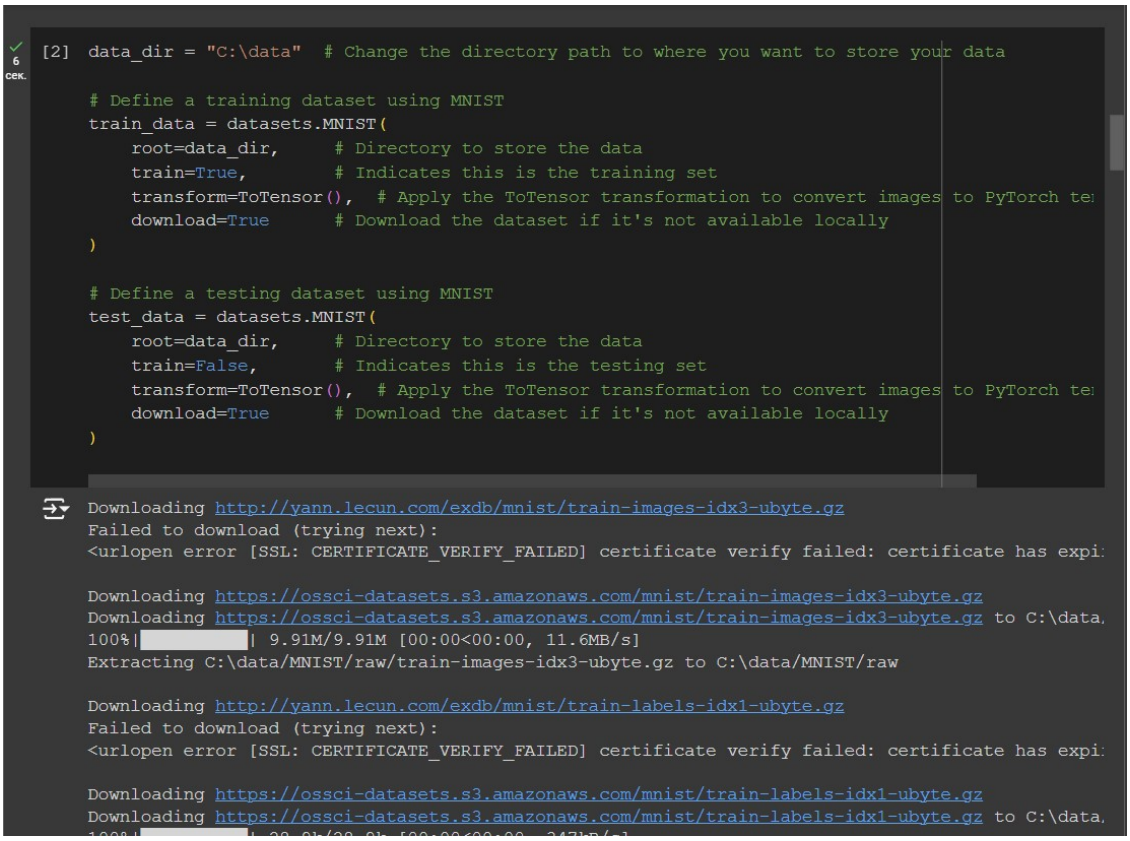

Figure 4: Step 2.1

Then setting up data loaders for efficient batch processing during training

```
[9]  from torch.utils.data import DataLoader

     # Define data loaders for training and testing datasets
     loaders = {
         'train': DataLoader(
             train_data,             # Training dataset
             batch_size=100,         # Number of samples in each mini-batch
             shuffle=True,           # Shuffle the data before each epoch
             num_workers=1           # Number of processes to use for data loading
         ),
         'test': DataLoader(
             test_data,              # Testing dataset
             batch_size=100,         # Number of samples in each mini-batch
             shuffle=True,           # Shuffle the data before each epoch
             num_workers=1           # Number of processes to use for data loading
         )
     }

[ ]  loaders

     {'train': <torch.utils.data.dataloader.DataLoader at 0x1468e9910>,
      'test': <torch.utils.data.dataloader.DataLoader at 0x146b0dbd0>}
```

Figure 5: Step 2.2

3. Defining a CNN model using PyTorch

```
[10]  import torch.nn as nn
      import torch.nn.functional as F
      import torch.optim as optim

      class CNN(nn.Module):
          def __init__(self):
              super(CNN, self).__init__()
              # Define the first convolutional layer
              self.conv1 = nn.Conv2d(1, 10, kernel_size=5)  # 1 input channel, 10 output channels, 5
              # Define the second convolutional layer
              self.conv2 = nn.Conv2d(10, 20, kernel_size=5)  # 10 input channels, 20 output channels
              # Apply dropout for regularization
              self.conv2_drop = nn.Dropout2d()
              # Define the first fully connected layer
              self.fc1 = nn.Linear(320, 50)  # 320 input features, 50 output features
              # Define the second fully connected layer (output layer)
              self.fc2 = nn.Linear(50, 10)  # 50 input features, 10 output features (for classificat

          def forward(self, x):
              # Apply first convolutional layer, followed by ReLU activation and max pooling
              x = F.relu(F.max_pool2d(self.conv1(x), 2))
              # Apply second convolutional layer, ReLU activation, dropout, and max pooling
              x = F.relu(F.max_pool2d(self.conv2_drop(self.conv2(x)), 2))
              # Flatten the tensor for input to fully connected layers
              x = x.view(-1, 320)
              # Apply first fully connected layer, followed by ReLU activation
              x = F.relu(self.fc1(x))
              # Apply dropout for regularization
              x = F.dropout(x, training=self.training)
              # Apply the second fully connected layer (output layer)
              x = self.fc2(x)
              # Apply softmax activation for classification
              return F.softmax(x)
```

Figure 6: Step 3

4. Model training

```
[11] import torch

    # Check if a CUDA-enabled GPU is available, and set the device accordingly
    device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
    # Instantiate the CNN model and move it to the specified device
    model = CNN().to(device)
    # Define an optimizer (Adam) to update the model's weights during training
    optimizer = optim.Adam(model.parameters(), lr=0.001)
    # Define a loss function (Cross Entropy Loss) for classification tasks
    loss_fn = nn.CrossEntropyLoss()

    # Define a training function
    def train(epoch):
        model.train()  # Set the model to training mode
        for batch_idx, (data, target) in enumerate(loaders['train']):
            # Move data and target to the specified device (GPU or CPU)
            data, target = data.to(device), target.to(device)
            optimizer.zero_grad()  # Zero out the gradients of the model parameters
            output = model(data)  # Forward pass to get the model's predictions
            loss = loss_fn(output, target)  # Compute the loss
            loss.backward()  # Backpropagate the gradients
            optimizer.step()  # Update the model's weights
            if batch_idx % 20 == 0:
                # Print training progress
                print(f'Train Epoch: {epoch} [{batch_idx * len(data)}/{len(loaders["train"].dataset)} ({100. * batch_idx / le

    # Define an evaluation function for the test set
    def test():
        model.eval()  # Set the model to evaluation mode
        test_loss = 0
        correct = 0
        with torch.no_grad():
            for data, target in loaders['test']:
                data, target = data.to(device), target.to(device)
                output = model(data)
                test_loss += loss_fn(output, target).item()
                pred = output.argmax(dim=1, keepdim=True)
                correct += pred.eq(target.view_as(pred)).sum().item()
        test_loss /= len(loaders['test'].dataset)
        # Print test set results
        print(f'\nTest set: Average loss: {test_loss:.4f}, Accuracy {correct}/{len(loaders["test"].dataset)} ({100. * correct
```

Figure 7: Step 4.1

Then training the model for 10 epochs

```
[12] for epoch in range(1, 11):
         train(epoch)
         test()
```

```
<ipython-input-10-9913cbdb0d11>:33: UserWarning: Implicit dimension choice for softmax has bee
  return F.softmax(x)
Train Epoch: 1 [0/60000 (0%)]   2.30257
Train Epoch: 1 [2000/60000 (3%)]        2.29309
Train Epoch: 1 [4000/60000 (7%)]        2.14843
Train Epoch: 1 [6000/60000 (10%)]       1.96351
Train Epoch: 1 [8000/60000 (13%)]       1.83979
Train Epoch: 1 [10000/60000 (17%)]      1.92311
Train Epoch: 1 [12000/60000 (20%)]      1.85857
Train Epoch: 1 [14000/60000 (23%)]      1.73407
Train Epoch: 1 [16000/60000 (27%)]      1.74905
Train Epoch: 1 [18000/60000 (30%)]      1.72304
Train Epoch: 1 [20000/60000 (33%)]      1.75846
Train Epoch: 1 [22000/60000 (37%)]      1.71518
Train Epoch: 1 [24000/60000 (40%)]      1.68317
Train Epoch: 1 [26000/60000 (43%)]      1.72163
Train Epoch: 1 [28000/60000 (47%)]      1.71989
Train Epoch: 1 [30000/60000 (50%)]      1.72841
Train Epoch: 1 [32000/60000 (53%)]      1.71401
Train Epoch: 1 [34000/60000 (57%)]      1.69763
Train Epoch: 1 [36000/60000 (60%)]      1.65606
Train Epoch: 1 [38000/60000 (63%)]      1.6065
Train Epoch: 1 [40000/60000 (67%)]      1.63181
Train Epoch: 1 [42000/60000 (70%)]      1.63692
Train Epoch: 1 [44000/60000 (73%)]      1.63101
Train Epoch: 1 [46000/60000 (77%)]      1.58045
```

Figure 8: Step 4.2

5. Evaluation and visualization

```
import matplotlib.pyplot as plt
model.eval()
data, target = test_data[9]
data = data.unsqueeze(0).to(device)
output = model(data)
prediction = output.argmax(dim=1, keepdim=True).item()
print (f'Prediction: {prediction}')
image = data.squeeze(0).squeeze(0).cpu().numpy()
plt.imshow(image, cmap='gray')
plt.show()
```

```
Prediction: 9
<ipython-input-10-9913cbdb0d11>:33: UserWarning: Implicit dimension choice for softmax has been deprecated. Change th
  return F.softmax(x)
```

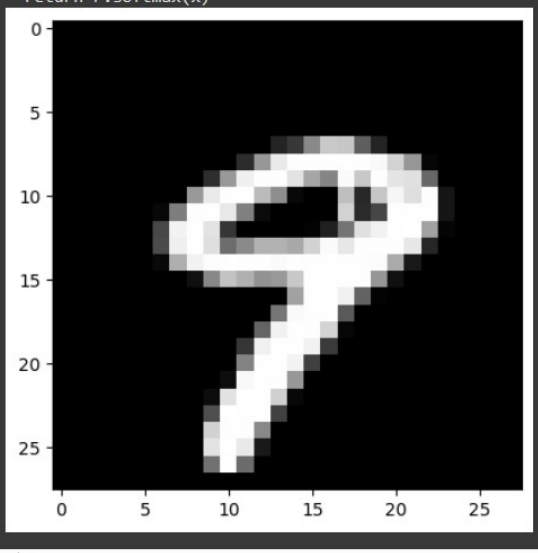

Figure 9: Step 5