```
import numpy as np
import torch
from torch.utils.data import Dataset
from torchvision import datasets
from torchvision.transforms import ToTensor
import matplotlib.pyplot as plt
from torchvision import transforms

import torch.nn as nn # torch.nn module, contains classes and functions to help bui

import torch.optim as optim # provides various optimization algorithms, such as SGI
from torch.utils.data import DataLoader, TensorDataset # Dataloader - helps to loac

from scipy.special import softmax
from sklearn.metrics import confusion_matrix, accuracy_score
```

Downloading the MNIST digit datasets

```
# Ensuring one-hot format
def one_hot_encoder(x):
  temp_array = np.zeros(10, dtype=float) # numpy arrays of zeros with length 10, 0
  temp_array[x] = 1 # element at index x in the temp array set to 1
  return temp_array

# To normalize the input
def transform(x):
  return np.array(x)/255.0
```

```
train_data = datasets.MNIST(root='./data', train = True , download=True, transform=
test_data = datasets.MNIST(root='./data', train = False ,download=True, transform=t
```

```
Downloading http://yann.lecun.com/exdb/mnist/train-images-idx3-ubyte.gz
Failed to download (trying next):
HTTP Error 403: Forbidden

Downloading https://ossci-datasets.s3.amazonaws.com/mnist/train-images-idx3-ul
Downloading https://ossci-datasets.s3.amazonaws.com/mnist/train-images-idx3-ul
100%|██████████| 9912422/9912422 [00:00<00:00, 50035215.10it/s]
Extracting ./data/MNIST/raw/train-images-idx3-ubyte.gz to ./data/MNIST/raw

Downloading http://yann.lecun.com/exdb/mnist/train-labels-idx1-ubyte.gz
Failed to download (trying next):
HTTP Error 403: Forbidden

Downloading https://ossci-datasets.s3.amazonaws.com/mnist/train-labels-idx1-ul
```

```
Downloading https://ossci-datasets.s3.amazonaws.com/mnist/train-labels-idx1-ul
Downloading https://ossci-datasets.s3.amazonaws.com/mnist/train-labels-idx1-ul
100%|██████████| 28881/28881 [00:00<00:00, 1852851.02it/s]Extracting ./data/MI

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

Failed to download (trying next):
HTTP Error 403: Forbidden

Downloading https://ossci-datasets.s3.amazonaws.com/mnist/t10k-images-idx3-uby
Downloading https://ossci-datasets.s3.amazonaws.com/mnist/t10k-images-idx3-uby
100%|██████████| 1648877/1648877 [00:00<00:00, 6123926.81it/s]
Extracting ./data/MNIST/raw/t10k-images-idx3-ubyte.gz to ./data/MNIST/raw

Downloading http://yann.lecun.com/exdb/mnist/t10k-labels-idx1-ubyte.gz
Failed to download (trying next):
HTTP Error 403: Forbidden

Downloading https://ossci-datasets.s3.amazonaws.com/mnist/t10k-labels-idx1-uby
Downloading https://ossci-datasets.s3.amazonaws.com/mnist/t10k-labels-idx1-uby
100%|██████████| 4542/4542 [00:00<00:00, 3924707.20it/s]Extracting ./data/MNIS
```

```
len(train_data)
```

```
60000
```

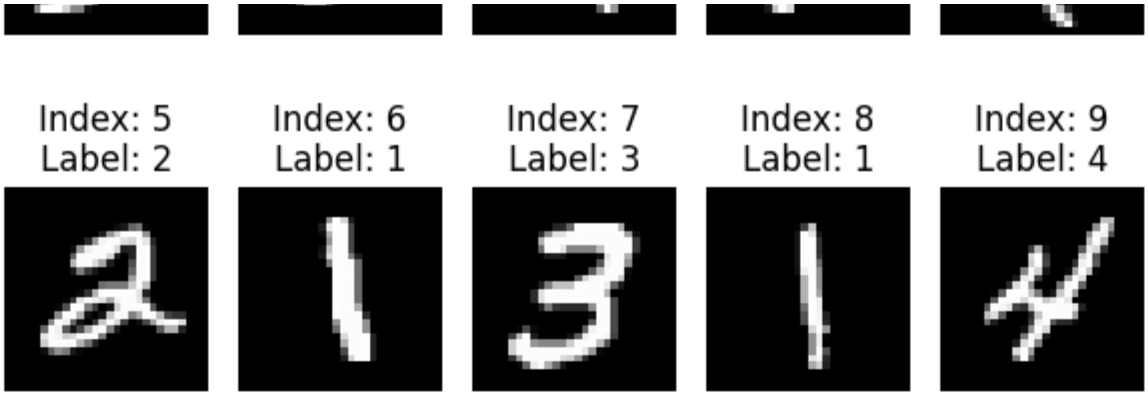
```
len(test_data)
```

```
10000
```

```
# Visualizing the data
fig, axes = plt.subplots(2, 5, figsize=(6, 4))  # 2 rows, 5 columns
for i in range(10): # Loop through the first 10 images
  ax = axes[i // 5, i % 5]  # Determine the position of the subplot (row, column)

  ax.imshow(train_data.data[i], cmap='gray')
  ax.set_title(f"Index: {i}\nLabel: {train_data.targets[i].item()}") # Set the ti
  ax.axis('off')

plt.tight_layout() # prevents from overlay
plt.show()
```




```
# organize the data in batches
# want to pass samples in “minibatches”, reshuffle the data at every epoch to red
train_dataloader = DataLoader(train_data, batch_size=64, shuffle=True)
test_dataloader = DataLoader(test_data, batch_size=64, shuffle=True)
```

```
len(train_dataloader)
```

```
938
```

```
len(test_dataloader)
```

```
157
```

Code from scratch

```
input_layer = train_data.data[i].flatten().shape[0]
hidden1_layer = 500
hidden2_layer = 250
hidden3_layer = 100
out_layer = train_data.train_labels.unique().shape[0]

layers_dims = [input_layer, hidden1_layer, hidden2_layer, hidden3_layer, out_laye
```

```
def initialize_parameters(layer_dimensions, initial):
  parameters = {}
  num_layers = len(layer_dimensions) # number of layers in the network

  for layer in range(1, num_layers):
    if initial == "glorot": # glorot intitalization
      M = np.sqrt(6*(1/(layer_dimensions[layer]+layer_dimensions[layer-1])))
      parameters['W' + str(layer)] = np.random.uniform(low = -M, high = M, size =
      parameters['b' + str(layer)] = np.zeros((layer_dimensions[layer], 1))
```

```
    elif initial == "random": # Random Initialization
      parameters['W' + str(layer)] = np.random.randn(layer_dimensions[layer], lay
      parameters['b' + str(layer)] = np.zeros((layer_dimensions[layer], 1))

    else:  # Zero Initialization
      parameters['W' + str(layer)] = np.zeros((layer_dimensions[layer], layer_dim
      parameters['b' + str(layer)] = np.zeros((layer_dimensions[layer], 1))

    assert(parameters['W' + str(layer)].shape == (layer_dimensions[layer], layer_
    assert(parameters['b' + str(layer)].shape == (layer_dimensions[layer], 1))

  return parameters
```

```
/usr/local/lib/python3.10/dist-packages/torchvision/datasets/mnist.py:66: Use
  warnings.warn("train_labels has been renamed targets")
```

Activation Function

```
# ReLu
def relu(x):
  return np.maximum(x, 0)
```

```
def relu_derivative(Z):
    return Z > 0  # This will return 1 where Z > 0, and 0 elsewhere
```

Forward Propagation

```
def forward_propagation(input_data, parameters, activation_function):
    forward_propagation = {}
    num_layers = int(len(parameters) / 2)  # Total number of layers (excluding in

    # Linear transformation for the first layer
    forward_propagation['Z1'] = np.dot(parameters['W1'], input_data) + parameters
    forward_propagation['A1'] = activation_function(forward_propagation['Z1'])

    # Loop through layers 2 to (num_layers - 1) (hidden layers)
    for layer in range(2, num_layers):
        # Activation from the previous layer
        forward_propagation['Z' + str(layer)] = np.dot(parameters['W' + str(layer
        forward_propagation['A' + str(layer)] = activation_function(forward_propa

    # For the final layer, apply softmax directly to the output
```

```
    # For the final layer, apply softmax directly to the output
    forward_propagation['Z' + str(num_layers)] = np.dot(parameters['W' + str(num_
    forward_propagation['A' + str(num_layers)] = softmax(forward_propagation['Z'

    # Store forward pass results and parameters for backpropagation
    cache = (forward_propagation, parameters)

    return forward_propagation['A' + str(num_layers)], cache
```

Backpropagation

```
def back_propagation(input, labels, cache):
    num_examples = input.shape[1]  # Number of examples in the batch (m)
    forward_propagation, parameters = cache
    num_layers = len(parameters) // 2  # Number of layers (assuming W1, b1, ...,
    grads = {}
    grads['dZ' + str(num_layers)] = forward_propagation['A' + str(num_layers)] -

    for layer in range(num_layers - 1, 0, -1):
        grads['dW' + str(layer + 1)] = (1. / num_examples) * np.dot(grads['dZ' +
        grads['db' + str(layer + 1)] = (1. / num_examples) * np.sum(grads['dZ' +
        grads['dA' + str(layer)] = np.dot(parameters['W' + str(layer + 1)].T, gra
        grads['dZ' + str(layer)] = grads['dA' + str(layer)] * np.where(forward_pr

    grads['dW1'] = 1. / num_examples * np.dot(grads['dZ1'], input.T)
    grads['db1'] = 1. / num_examples * np.sum(grads['dZ1'], axis=1, keepdims=True

    return grads
```

Update parameters

```
def update_parameters(parameters, grads, learning_rate, lambd=0):
    num_layers = len(parameters) // 2  # Number of layers in the network
    for layer in range(num_layers):
        parameters["W" + str(layer + 1)] -= (learning_rate * (grads["dW" + str(la
        parameters["b" + str(layer + 1)] -= (learning_rate * grads["db" + str(lay
    return parameters
```

Cost Funtion

```
def cross_entropy_cost(predictions, labels):
    num_examples = labels.shape[1]

    # Compute the cross-entropy loss
    loss_per_example = -np.sum(labels * np.log(predictions), axis=0)
    average_cost = 1. / num_examples * np.sum(loss_per_example)

    return average_cost
```

```
train_dataloader = DataLoader(train_data, batch_size=64, shuffle=True)
test_dataloader = DataLoader(test_data, batch_size=64, shuffle=True)
```

Accuracy and Confusion matrix

```
def accuracy(parameter, test_data, function):
  size = test_data.data.shape[0]
  img_size = test_data.data.shape[1] * test_data.data.shape[2]

  test_dataloader = next(iter(DataLoader(test_data, batch_size=size, shuffle=True
  X = np.swapaxes(np.array(test_dataloader[0]),0,2).reshape(img_size, size)

  pred = np.swapaxes(forward_propagation(X, parameter, function)[0], 0, 1)
  Y = np.array(test_dataloader[1])

  accuracy = accuracy_score(np.argmax(Y, axis=1), np.argmax(pred, axis=1))
  return accuracy
```

```
def confusion_mat(parameter, test_data, function):
  size = test_data.data.shape[0]
  img_size = test_data.data.shape[1] * test_data.data.shape[2]

  test_dataloader = next(iter(DataLoader(test_data, batch_size=size, shuffle=True
  X = np.swapaxes(np.array(test_dataloader[0]),0,2).reshape(img_size, size)

  pred = np.swapaxes(forward_propagation(X, parameter, function)[0], 0, 1)
  Y = np.array(test_dataloader[1])

  confu_matrix = confusion_matrix(np.argmax(Y, axis=1), np.argmax(pred, axis=1))
  return confu_matrix
```

Training the model

```
def model(train_dataloader, test_data, batch_size=64, learning_rate=0.01, epoch=1
    grads = {}
    train_costs = []  # To store training costs
    test_costs = []   # To store test costs
    layers_dims = [input_layer, hidden1_layer, hidden2_layer, hidden3_layer, out_
    parameters = initialize_parameters(layers_dims, initial)
    count = 0

    for i in range(epoch):
        for (batch_idx, batch) in enumerate(train_dataloader):
            batch_x, batch_y = batch
            X = np.swapaxes(np.array(batch_x), 0, 2).reshape(batch_x.shape[1]*bat
            Y = np.swapaxes(np.array(batch_y), 0, 1)

            # Forward propagation
            a3, cache = forward_propagation(X, parameters, function)
            train_cost = cross_entropy_cost(a3, Y)

            # Backward propagation and parameter update
            grads = back_propagation(X, Y, cache)
            parameters = update_parameters(parameters, grads, learning_rate, lamb

            if batch_idx % 200 == 0:
                train_costs.append(train_cost)

                # Calculate test loss at every 200th batch
                test_dataloader = next(iter(DataLoader(test_data, batch_size=batc
                test_x = np.swapaxes(np.array(test_dataloader[0]), 0, 2).reshape(
                test_y = np.swapaxes(np.array(test_dataloader[1]), 0, 1)
                test_a3, _ = forward_propagation(test_x, parameters, function)
                test_cost = cross_entropy_cost(test_a3, test_y)
                test_costs.append(test_cost)

            if print_cost and batch_idx % 200 == 0:
                print(f"Cost after epoch {i}, iteration {batch_idx}: Train Cost:

    return parameters, train_costs, test_costs
```

```
def plotting(parameters, test_data, train_data, function):
    # Calculate test and train accuracy, passing the 'function' parameter
    test_acc = accuracy(parameters[0], test_data, function)
    train_acc = accuracy(parameters[0], train_data, function)

    # Generate confusion matrix for the test data
    conf_matrix = confusion_mat(parameters[0], test_data, function)

    # Create two subplots: one for the confusion matrix, one for the loss curves
```

```
    # Create two subplots: one for the confusion matrix, one for the loss curves
    fig, (ax, bx) = plt.subplots(1, 2, figsize=(20, 8))

    # Plot the confusion matrix
    ax.matshow(conf_matrix, cmap='viridis', alpha=0.3)
    for i in range(conf_matrix.shape[0]):
        for j in range(conf_matrix.shape[1]):
            ax.text(x=j, y=i, s=conf_matrix[i, j], va='center', ha='center', size

    ax.set_xlabel('Predicted Label', fontsize=18)
    ax.set_ylabel('True Label', fontsize=18)
    ax.set_title('Confusion Matrix', fontsize=18)

    # Plot the cost curve over iterations (training and test)
    bx.plot(range(0, len(parameters[1])), parameters[1], label='Train Loss', colo
    bx.plot(range(0, len(parameters[2])), parameters[2], label='Test Loss', color

    bx.set_xlabel('Iteration (x 200)', fontsize=18)
    bx.set_ylabel('Loss', fontsize=18)
    bx.set_title('Training and Test Loss Over Iterations', fontsize=18)
    bx.legend()

    # Combine test and train accuracy in a label for the plot
    label = f"Test acc. = {test_acc * 100:.2f}%, Train acc. = {train_acc * 100:.2
    plt.suptitle(label, fontsize=20)

    # Show the plots
    plt.tight_layout()
    plt.show()



# Dictionary to store learned parameters for different models
learned_parameters = {}

learning_rate = 0.01
lambd = 0
epoch = 15
batch_size = 64
initial = "zero"
train_dataloader = DataLoader(train_data, batch_size=batch_size, shuffle=True)

# Create a model name (key) based on training parameters
model_name = "Epoch=" + str(epoch) + ",alpha=" + str(learning_rate) + ",Regulariz
print("Model Key: " + model_name)

# Train the model and store the learned parameters
learned_parameters[model_name] = model(train_dataloader, test_data, batch_size=ba

# Find the model with 'zero' initialization dynamically
i = [key for key in learned_parameters.keys() if "Initilization=zero" in key][0]
```

```
# Plotting the losses and confusion matrix for the 'zero' initialization model
plotting(learned_parameters[i], test_data, train_data, relu)
```

```
Model Key: Epoch=15,alpha=0.01,Regularization=0,Batch=64,Initilization=zero
Cost after epoch 0, iteration 0: Train Cost: 3.2508297339144825, Test Cost: 3
Cost after epoch 0, iteration 200: Train Cost: 3.24939795028802, Test Cost: 3
Cost after epoch 0, iteration 400: Train Cost: 3.251901229631395, Test Cost:
Cost after epoch 0, iteration 600: Train Cost: 3.2440223417335554, Test Cost:
Cost after epoch 0, iteration 800: Train Cost: 3.2489579698084645, Test Cost:
Cost after epoch 1, iteration 0: Train Cost: 3.2505041051334485, Test Cost: 3
Cost after epoch 1, iteration 200: Train Cost: 3.2503799521085806, Test Cost:
Cost after epoch 1, iteration 400: Train Cost: 3.2589958170135027, Test Cost:
Cost after epoch 1, iteration 600: Train Cost: 3.2495542345414776, Test Cost:
Cost after epoch 1, iteration 800: Train Cost: 3.245617694521777, Test Cost:
Cost after epoch 2, iteration 0: Train Cost: 3.2499223296210906, Test Cost: 3
Cost after epoch 2, iteration 200: Train Cost: 3.241424625158282, Test Cost:
Cost after epoch 2, iteration 400: Train Cost: 3.2558035561979395, Test Cost:
Cost after epoch 2, iteration 600: Train Cost: 3.2460361215755746, Test Cost:
Cost after epoch 2, iteration 800: Train Cost: 3.2514404687101757, Test Cost:
Cost after epoch 3, iteration 0: Train Cost: 3.252257454911647, Test Cost: 3.
Cost after epoch 3, iteration 200: Train Cost: 3.244598785999183, Test Cost:
Cost after epoch 3, iteration 400: Train Cost: 3.24500309553063, Test Cost: 3
Cost after epoch 3, iteration 600: Train Cost: 3.246884667018073, Test Cost:
Cost after epoch 3, iteration 800: Train Cost: 3.2413919985375936, Test Cost:
Cost after epoch 4, iteration 0: Train Cost: 3.239118796549061, Test Cost: 3.
Cost after epoch 4, iteration 200: Train Cost: 3.255939337124784, Test Cost:
Cost after epoch 4, iteration 400: Train Cost: 3.251981065737858, Test Cost:
Cost after epoch 4, iteration 600: Train Cost: 3.247334781840356, Test Cost:
Cost after epoch 4, iteration 800: Train Cost: 3.251790408152299, Test Cost:
Cost after epoch 5, iteration 0: Train Cost: 3.240044406119909, Test Cost: 3.
Cost after epoch 5, iteration 200: Train Cost: 3.253877140425434, Test Cost:
Cost after epoch 5, iteration 400: Train Cost: 3.25974545525141, Test Cost: 3
Cost after epoch 5, iteration 600: Train Cost: 3.2403146498969146, Test Cost:
Cost after epoch 5, iteration 800: Train Cost: 3.247596279553224, Test Cost:
Cost after epoch 6, iteration 0: Train Cost: 3.2394318048586572, Test Cost: 3
Cost after epoch 6, iteration 200: Train Cost: 3.2545592407175716, Test Cost:
Cost after epoch 6, iteration 400: Train Cost: 3.2479329992143047, Test Cost:
Cost after epoch 6, iteration 600: Train Cost: 3.25471659764796, Test Cost: 3
Cost after epoch 6, iteration 800: Train Cost: 3.235751374988454, Test Cost:
Cost after epoch 7, iteration 0: Train Cost: 3.246358680566535, Test Cost: 3.
Cost after epoch 7, iteration 200: Train Cost: 3.242750880516536, Test Cost:
Cost after epoch 7, iteration 400: Train Cost: 3.2463167183569297, Test Cost:
Cost after epoch 7, iteration 600: Train Cost: 3.2524655130636333, Test Cost:
Cost after epoch 7, iteration 800: Train Cost: 3.2494049149372826, Test Cost:
Cost after epoch 8, iteration 0: Train Cost: 3.252164143617321, Test Cost: 3.
Cost after epoch 8, iteration 200: Train Cost: 3.254002451062099, Test Cost:
Cost after epoch 8, iteration 400: Train Cost: 3.251114730240908, Test Cost:
Cost after epoch 8, iteration 600: Train Cost: 3.24392239281945, Test Cost: 3
Cost after epoch 8, iteration 800: Train Cost: 3.248767820087349, Test Cost:
Cost after epoch 9, iteration 0: Train Cost: 3.2558388712342894, Test Cost: 3
Cost after epoch 9, iteration 200: Train Cost: 3.2409320665896466, Test Cost:
```

```
Cost after epoch 9, iteration 400: Train Cost: 3.2488011704494983, Test Cost:
Cost after epoch 9, iteration 600: Train Cost: 3.2541022444418664, Test Cost:
Cost after epoch 9, iteration 800: Train Cost: 3.2606907727909613, Test Cost:
Cost after epoch 10, iteration 0: Train Cost: 3.253096320132792, Test Cost: 3
Cost after epoch 10, iteration 200: Train Cost: 3.243405648395294, Test Cost:
Cost after epoch 10, iteration 400: Train Cost: 3.260652927698164, Test Cost:
Cost after epoch 10, iteration 600: Train Cost: 3.254340383468567, Test Cost:
```

```
print("Available model keys:", learned_parameters.keys())
```

```
Available model keys: dict_keys(['Epoch=15,alpha=0.01,Regularization=0,Batch=(
```

```
# Dictionary to store learned parameters for different models
learned_parameters = {}

learning_rate = 0.01
lambd = 0
epoch = 15
batch_size = 64
initial = "random"
train_dataloader = DataLoader(train_data, batch_size=batch_size, shuffle=True)

# Create a model name (key) based on training parameters
model_name = "Epoch=" + str(epoch) + ",alpha=" + str(learning_rate) + ",Regulariz
print("Model Key: " + model_name)

# Train the model and store the learned parameters
learned_parameters[model_name] = model(train_dataloader, test_data, batch_size=ba

# Find the model with 'zero' initialization dynamically
i = [key for key in learned_parameters.keys() if "Initilization=random" in key][0

# Plotting the losses and confusion matrix for the 'zero' initialization model
plotting(learned_parameters[i], test_data, train_data, relu)
```

```
Model Key: Epoch=15,alpha=0.01,Regularization=0,Batch=64,Initilization=random
Cost after epoch 0, iteration 0: Train Cost: 3.218838252648877, Test Cost: 3.:
Cost after epoch 0, iteration 200: Train Cost: 2.413697506918888, Test Cost: :
Cost after epoch 0, iteration 400: Train Cost: 1.2041585253188556, Test Cost:
Cost after epoch 0, iteration 600: Train Cost: 1.057225404453035, Test Cost: (
Cost after epoch 0, iteration 800: Train Cost: 0.5800738218132238, Test Cost:
Cost after epoch 1, iteration 0: Train Cost: 0.7040102898899068, Test Cost: 0
Cost after epoch 1, iteration 200: Train Cost: 0.5670297315924172, Test Cost:
Cost after epoch 1, iteration 400: Train Cost: 0.8156301033129001, Test Cost:
Cost after epoch 1, iteration 600: Train Cost: 0.7189690987912521, Test Cost:
Cost after epoch 1, iteration 800: Train Cost: 0.6359562172954933, Test Cost:
Cost after epoch 2, iteration 0: Train Cost: 0.5125752483536619, Test Cost: 0
Cost after epoch 2, iteration 200: Train Cost: 0.4365782834040839, Test Cost:
Cost after epoch 2, iteration 400: Train Cost: 0.29189614844356154, Test Cost
Cost after epoch 2, iteration 600: Train Cost: 0.6576563867808148, Test Cost:
```

```
Cost after epoch 2, iteration 800: Train Cost: 0.3635436541834244, Test Cost:
Cost after epoch 3, iteration 0: Train Cost: 0.5032459585614885, Test Cost: 0
Cost after epoch 3, iteration 200: Train Cost: 0.3582470747974458, Test Cost:
Cost after epoch 3, iteration 400: Train Cost: 0.4206006299722157, Test Cost:
Cost after epoch 3, iteration 600: Train Cost: 0.562134326839552, Test Cost: (
Cost after epoch 3, iteration 800: Train Cost: 0.7226689346103653, Test Cost:
Cost after epoch 4, iteration 0: Train Cost: 0.6081322813335993, Test Cost: 0
Cost after epoch 4, iteration 200: Train Cost: 0.3319938623081593, Test Cost:
Cost after epoch 4, iteration 400: Train Cost: 0.4057924107689761, Test Cost:
Cost after epoch 4, iteration 600: Train Cost: 0.3567684047732557, Test Cost:
Cost after epoch 4, iteration 800: Train Cost: 0.7807678373207007, Test Cost:
Cost after epoch 5, iteration 0: Train Cost: 0.19310537612886658, Test Cost: (
Cost after epoch 5, iteration 200: Train Cost: 0.10688969777833322, Test Cost
Cost after epoch 5, iteration 400: Train Cost: 0.13249244430729765, Test Cost
Cost after epoch 5, iteration 600: Train Cost: 0.1602265522985510 4, Test Cost
Cost after epoch 5, iteration 800: Train Cost: 0.5126728773086064, Test Cost:
Cost after epoch 6, iteration 0: Train Cost: 0.10247412214091248, Test Cost: (
Cost after epoch 6, iteration 200: Train Cost: 0.40687463597016804, Test Cost
Cost after epoch 6, iteration 400: Train Cost: 0.22157176649333063, Test Cost
Cost after epoch 6, iteration 600: Train Cost: 0.4014069384382166, Test Cost:
Cost after epoch 6, iteration 800: Train Cost: 0.17941908555966, Test Cost: 0
Cost after epoch 7, iteration 0: Train Cost: 0.21794142582447723, Test Cost: (
Cost after epoch 7, iteration 200: Train Cost: 0.34749761163143655, Test Cost
Cost after epoch 7, iteration 400: Train Cost: 0.1720653529580548, Test Cost:
Cost after epoch 7, iteration 600: Train Cost: 0.13178126705766363, Test Cost
Cost after epoch 7, iteration 800: Train Cost: 0.19290935661276448, Test Cost
Cost after epoch 8, iteration 0: Train Cost: 0.3531511431039227, Test Cost: 0
Cost after epoch 8, iteration 200: Train Cost: 0.1984169492680791, Test Cost:
Cost after epoch 8, iteration 400: Train Cost: 0.1807641178875249, Test Cost:
Cost after epoch 8, iteration 600: Train Cost: 0.15917684237388124, Test Cost
Cost after epoch 8, iteration 800: Train Cost: 0.34888602272877567, Test Cost
Cost after epoch 9, iteration 0: Train Cost: 0.15576323469250536, Test Cost: (
Cost after epoch 9, iteration 200: Train Cost: 0.26227835409606515, Test Cost
Cost after epoch 9, iteration 400: Train Cost: 0.23013684765634226, Test Cost
Cost after epoch 9, iteration 600: Train Cost: 0.217578897255082, Test Cost: (
Cost after epoch 9, iteration 800: Train Cost: 0.18163370472597137, Test Cost
Cost after epoch 10, iteration 0: Train Cost: 0.21122324581251659, Test Cost:
Cost after epoch 10, iteration 200: Train Cost: 0.11537614958231102, Test Cos
Cost after epoch 10, iteration 400: Train Cost: 0.2815264730660627 7, Test Cos
Cost after epoch 10, iteration 600: Train Cost: 0.2439299670195545, Test Cost
```

```
# Dictionary to store learned parameters for different models
learned_parameters = {}

learning_rate = 0.01
lambd = 0
epoch = 15
batch_size = 64
initial = "glorot"
train_dataloader = DataLoader(train_data, batch_size=batch_size, shuffle=True)

# Create a model name (key) based on training parameters
model_name = "Epoch=" + str(epoch) + ",alpha=" + str(learning_rate) + ",Regulariz
```

```
print("Model Key: " + model_name)

# Train the model and store the learned parameters
learned_parameters[model_name] = model(train_dataloader, test_data, batch_size=ba

# Find the model with 'zero' initialization dynamically
i = [key for key in learned_parameters.keys() if "Initilization=glorot" in key][0

# Plotting the losses and confusion matrix for the 'zero' initialization model
plotting(learned_parameters[i], test_data, train_data, relu)
```

```
Model Key: Epoch=15,alpha=0.01,Regularization=0,Batch=64,Initilization=glorot
Cost after epoch 0, iteration 0: Train Cost: 3.1491358721260605, Test Cost: 3
Cost after epoch 0, iteration 200: Train Cost: 1.831596902920069, Test Cost:
Cost after epoch 0, iteration 400: Train Cost: 0.9655877335065852, Test Cost:
Cost after epoch 0, iteration 600: Train Cost: 0.7634845119596534, Test Cost:
Cost after epoch 0, iteration 800: Train Cost: 0.5908568732004384, Test Cost:
Cost after epoch 1, iteration 0: Train Cost: 0.6118175353574904, Test Cost: 0
Cost after epoch 1, iteration 200: Train Cost: 0.5498190296460304, Test Cost:
Cost after epoch 1, iteration 400: Train Cost: 0.5206075203096745, Test Cost:
Cost after epoch 1, iteration 600: Train Cost: 0.6809460988549756, Test Cost:
Cost after epoch 1, iteration 800: Train Cost: 0.3877021344296081, Test Cost:
Cost after epoch 2, iteration 0: Train Cost: 0.34464586818803783, Test Cost:
Cost after epoch 2, iteration 200: Train Cost: 0.4104972634903842, Test Cost:
Cost after epoch 2, iteration 400: Train Cost: 0.49687317983546775, Test Cost
Cost after epoch 2, iteration 600: Train Cost: 0.45994111707027197, Test Cost
Cost after epoch 2, iteration 800: Train Cost: 0.5976215364519388, Test Cost:
Cost after epoch 3, iteration 0: Train Cost: 0.3512507305334221, Test Cost: 0
Cost after epoch 3, iteration 200: Train Cost: 0.3284242873549258, Test Cost:
Cost after epoch 3, iteration 400: Train Cost: 0.47185553304544925, Test Cost
Cost after epoch 3, iteration 600: Train Cost: 0.5085847028630269, Test Cost:
Cost after epoch 3, iteration 800: Train Cost: 0.45176985980048256, Test Cost
Cost after epoch 4, iteration 0: Train Cost: 0.30846728663556455, Test Cost:
Cost after epoch 4, iteration 200: Train Cost: 0.26026372406943415, Test Cost
Cost after epoch 4, iteration 400: Train Cost: 0.5345633462014411, Test Cost:
Cost after epoch 4, iteration 600: Train Cost: 0.24420732996009523, Test Cost
Cost after epoch 4, iteration 800: Train Cost: 0.5709085827945755, Test Cost:
Cost after epoch 5, iteration 0: Train Cost: 0.3595037044839017, Test Cost: 0
Cost after epoch 5, iteration 200: Train Cost: 0.5478994018856431, Test Cost:
Cost after epoch 5, iteration 400: Train Cost: 0.23647373542270617, Test Cost
Cost after epoch 5, iteration 600: Train Cost: 0.19067087682743852, Test Cost
Cost after epoch 5, iteration 800: Train Cost: 0.19879792994022727, Test Cost
Cost after epoch 6, iteration 0: Train Cost: 0.2992277018920634, Test Cost: 0
Cost after epoch 6, iteration 200: Train Cost: 0.8498496675018274, Test Cost:
Cost after epoch 6, iteration 400: Train Cost: 0.3452860636442318, Test Cost:
Cost after epoch 6, iteration 600: Train Cost: 0.06726467503762161, Test Cost
Cost after epoch 6, iteration 800: Train Cost: 0.608364023404508, Test Cost:
Cost after epoch 7, iteration 0: Train Cost: 0.2917398046589178, Test Cost: 0
Cost after epoch 7, iteration 200: Train Cost: 0.15951046749328324, Test Cost
Cost after epoch 7, iteration 400: Train Cost: 0.20362379639751477, Test Cost
Cost after epoch 7, iteration 600: Train Cost: 0.14934907802590197, Test Cost
```

```
Cost after epoch 7, iteration 800: Train Cost: 0.1623741130768061, Test Cost:
Cost after epoch 8, iteration 0: Train Cost: 0.33758793218500543, Test Cost: (
Cost after epoch 8, iteration 200: Train Cost: 0.2000534328302841, Test Cost:
Cost after epoch 8, iteration 400: Train Cost: 0.12350466019225821, Test Cost
Cost after epoch 8, iteration 600: Train Cost: 0.27665385240446444, Test Cost
Cost after epoch 8, iteration 800: Train Cost: 0.1449106261989766, Test Cost:
Cost after epoch 9, iteration 0: Train Cost: 0.20477861576452056, Test Cost: (
Cost after epoch 9, iteration 200: Train Cost: 0.13069380267328978, Test Cost
Cost after epoch 9, iteration 400: Train Cost: 0.2071698474542938, Test Cost:
Cost after epoch 9, iteration 600: Train Cost: 0.15108479023472632, Test Cost
Cost after epoch 9, iteration 800: Train Cost: 0.14895334064614957, Test Cost
Cost after epoch 10, iteration 0: Train Cost: 0.21936855122285925, Test Cost:
Cost after epoch 10, iteration 200: Train Cost: 0.1762505972522387, Test Cost
Cost after epoch 10, iteration 400: Train Cost: 0.3679280024539424, Test Cost
Cost after epoch 10, iteration 600: Train Cost: 0.24238483044144857, Test Cos
```

```
# Dictionary to store learned parameters for different models
learned_parameters = {}

learning_rate = 0.3
lambd = 0
epoch = 15
batch_size = 64
initial = "glorot"
train_dataloader = DataLoader(train_data, batch_size=batch_size, shuffle=True)

# Create a model name (key) based on training parameters
model_name = "Epoch=" + str(epoch) + ",alpha=" + str(learning_rate) + ",Regulariz
print("Model Key: " + model_name)

# Train the model and store the learned parameters
learned_parameters[model_name] = model(train_dataloader, test_data, batch_size=ba

# Find the model with 'zero' initialization dynamically
i = [key for key in learned_parameters.keys() if "Initilization=glorot" in key][0

# Plotting the losses and confusion matrix for the 'zero' initialization model
plotting(learned_parameters[i], test_data, train_data, relu)
```

```
Model Key: Epoch=15,alpha=0.3,Regularization=0,Batch=64,Initilization=glorot
Cost after epoch 0, iteration 0: Train Cost: 3.2883372797503925, Test Cost: 3
Cost after epoch 0, iteration 200: Train Cost: 0.5495445617672459, Test Cost:
Cost after epoch 0, iteration 400: Train Cost: 0.4678886868713272, Test Cost:
Cost after epoch 0, iteration 600: Train Cost: 0.3875411149755045, Test Cost:
Cost after epoch 0, iteration 800: Train Cost: 0.081398409098252, Test Cost: (
Cost after epoch 1, iteration 0: Train Cost: 0.5620622842952194, Test Cost: 0
Cost after epoch 1, iteration 200: Train Cost: 0.17902305458827256, Test Cost
Cost after epoch 1, iteration 400: Train Cost: 0.23716930740681794, Test Cost
Cost after epoch 1, iteration 600: Train Cost: 0.07327337981182346, Test Cost
Cost after epoch 1, iteration 800: Train Cost: 0.1052758087526708, Test Cost:
Cost after epoch 2, iteration 0: Train Cost: 0.1510840934273147 4, Test Cost: (
```

```
Cost after epoch 2, iteration 200: Train Cost: 0.06632165160399123, Test Cost
Cost after epoch 2, iteration 400: Train Cost: 0.1313256453318109, Test Cost:
Cost after epoch 2, iteration 600: Train Cost: 0.08622667457611868, Test Cost
Cost after epoch 2, iteration 800: Train Cost: 0.026170405177640972, Test Cos
Cost after epoch 3, iteration 0: Train Cost: 0.290955516573197, Test Cost: 0.
Cost after epoch 3, iteration 200: Train Cost: 0.01486086893487867, Test Cost
Cost after epoch 3, iteration 400: Train Cost: 0.05369989823238475, Test Cost
Cost after epoch 3, iteration 600: Train Cost: 0.04219601683331178, Test Cost
Cost after epoch 3, iteration 800: Train Cost: 0.09346042351039112, Test Cost
Cost after epoch 4, iteration 0: Train Cost: 0.010323726144585679, Test Cost:
Cost after epoch 4, iteration 200: Train Cost: 0.028458494155136717, Test Cos
Cost after epoch 4, iteration 400: Train Cost: 0.026677042358065672, Test Cos
Cost after epoch 4, iteration 600: Train Cost: 0.02991863177870607, Test Cost
Cost after epoch 4, iteration 800: Train Cost: 0.04228453297874096, Test Cost
Cost after epoch 5, iteration 0: Train Cost: 0.0680849311732705, Test Cost: 0
Cost after epoch 5, iteration 200: Train Cost: 0.07760830452508581, Test Cost
Cost after epoch 5, iteration 400: Train Cost: 0.0012077765962206768, Test Co
Cost after epoch 5, iteration 600: Train Cost: 0.01585410862772407, Test Cost
Cost after epoch 5, iteration 800: Train Cost: 0.1606070033799586, Test Cost:
Cost after epoch 6, iteration 0: Train Cost: 0.02247175697009891, Test Cost: (
Cost after epoch 6, iteration 200: Train Cost: 0.015101035704521499, Test Cos
Cost after epoch 6, iteration 400: Train Cost: 0.014312871274018434, Test Cos
Cost after epoch 6, iteration 600: Train Cost: 0.0015974537630823876, Test Co
Cost after epoch 6, iteration 800: Train Cost: 0.00822445891858425, Test Cost
Cost after epoch 7, iteration 0: Train Cost: 0.02127341259740205, Test Cost: (
Cost after epoch 7, iteration 200: Train Cost: 0.007298332305587385, Test Cos
Cost after epoch 7, iteration 400: Train Cost: 0.00031887706831030437, Test C
Cost after epoch 7, iteration 600: Train Cost: 0.005892542065704087, Test Cos
Cost after epoch 7, iteration 800: Train Cost: 0.002279177223042106, Test Cos
Cost after epoch 8, iteration 0: Train Cost: 0.0020043471084213835, Test Cost
Cost after epoch 8, iteration 200: Train Cost: 0.06723883359271239, Test Cost
Cost after epoch 8, iteration 400: Train Cost: 0.00922175680453876, Test Cost
Cost after epoch 8, iteration 600: Train Cost: 0.002005618196221882, Test Cos
Cost after epoch 8, iteration 800: Train Cost: 0.001818159981027596, Test Cos
Cost after epoch 9, iteration 0: Train Cost: 0.3411576298138145, Test Cost: 0
Cost after epoch 9, iteration 200: Train Cost: 0.001831891396416096, Test Cos
Cost after epoch 9, iteration 400: Train Cost: 0.0029425386660178157, Test Co
Cost after epoch 9, iteration 600: Train Cost: 0.08846788244833238, Test Cost
Cost after epoch 9, iteration 800: Train Cost: 0.05727149590123957, Test Cost
Cost after epoch 10, iteration 0: Train Cost: 0.0886160971289469, Test Cost: (
Cost after epoch 10, iteration 200: Train Cost: 0.05679101527429631, Test Cos
Cost after epoch 10, iteration 400: Train Cost: 0.0003263169258596925, Test C
Cost after epoch 10, iteration 600: Train Cost: 0.0008393084755179711, Test C
Cost after epoch 10, iteration 800: Train Cost: 0.039680085721876014, Test Co
Cost after epoch 11, iteration 0: Train Cost: 0.0006279343403679876, Test Cos
Cost after epoch 11, iteration 200: Train Cost: 7.664829604568723e-05, Test C
Cost after epoch 11, iteration 400: Train Cost: 0.00016820222644945698, Test (
Cost after epoch 11, iteration 600: Train Cost: 0.0006608678116941085, Test C
Cost after epoch 11, iteration 800: Train Cost: 0.039439527024230696, Test Co
Cost after epoch 12, iteration 0: Train Cost: 0.031588308295021725, Test Cost
Cost after epoch 12, iteration 200: Train Cost: 0.0009037683289377626, Test C
Cost after epoch 12, iteration 400: Train Cost: 7.25536699269101e-05, Test Co
```

```
Cost after epoch 12, iteration 600: Train Cost: 0.002228786974600552, Test Co:
Cost after epoch 12, iteration 800: Train Cost: 0.0009971788226654959, Test C(
Cost after epoch 13, iteration 0: Train Cost: 0.0003584469136539211, Test Cos
Cost after epoch 13, iteration 200: Train Cost: 6.316140080368613e-05, Test C(
Cost after epoch 13, iteration 400: Train Cost: 0.000737593806415879, Test Co:
Cost after epoch 13, iteration 600: Train Cost: 0.00024780013576339744, Test (
Cost after epoch 13, iteration 800: Train Cost: 0.001141811004781436, Test Co:
Cost after epoch 14, iteration 0: Train Cost: 1.910523533507803e-05, Test Cos
Cost after epoch 14, iteration 200: Train Cost: 0.0004575698134305698, Test C(
Cost after epoch 14, iteration 400: Train Cost: 0.0001884520753786402, Test C(
Cost after epoch 14, iteration 600: Train Cost: 0.006086689135928737, Test Co:
Cost after epoch 14, iteration 800: Train Cost: 0.00018914863540800056, Test (
```

Test acc. = 98.56%, Train acc. = 100.00%