```
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 SGD
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
```

```
len(train_data)
```

```
60000
```

```
len(test_data)
```

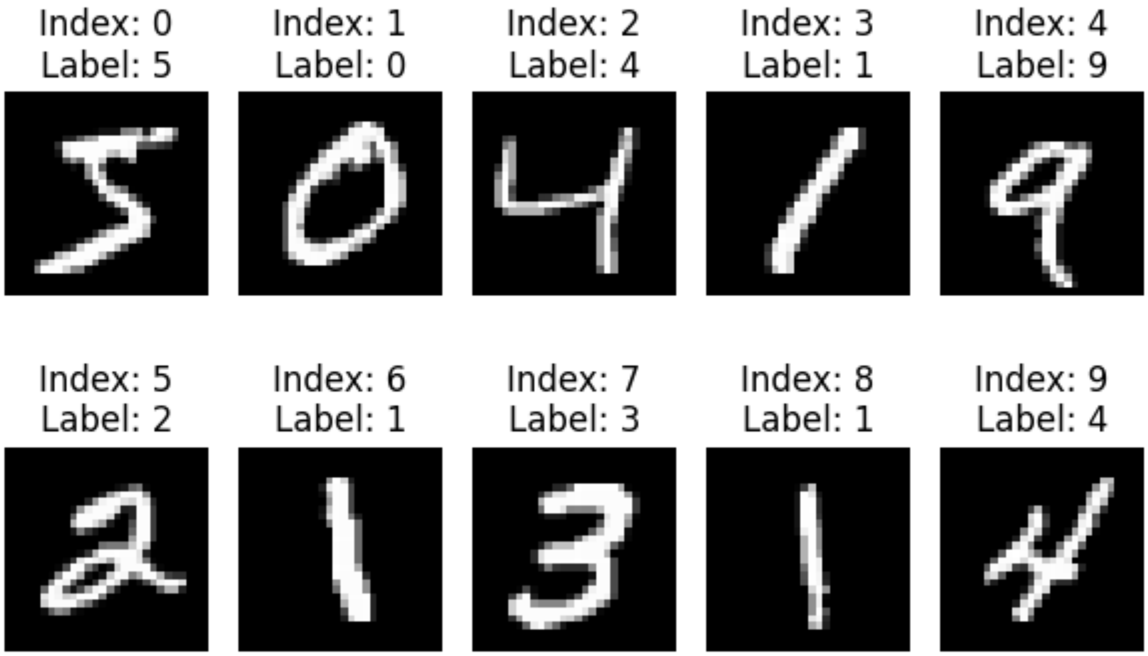
```
10000
```

```
# 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') # Display each image in grayscale
  ax.set_title(f"Index: {i}\nLabel: {train_data.targets[i].item()}")
  ax.axis('off')

plt.tight_layout() # Adjust layout to prevent overlap of titles
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":
      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":
      parameters['W' + str(layer)] = np.random.randn(layer_dimensions[layer], lay
      parameters['b' + str(layer)] = np.zeros((layer_dimensions[layer], 1))

    else:
      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
```

Activation Function

```
# tanh activation function
def tanh(x):
  return (np.exp(x) - np.exp(-x)) / (np.exp(x) + np.exp(-x))

def tanh_derivative(Z):
    """Compute the derivative of the tanh activation function."""
    return 1 - np.tanh(Z)**2
```

Forward Propagation

```
def forward_propagation(input_data, parameters, activation_function):
```

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

    forward_propagation['Z1'] = np.dot(parameters['W1'], input_data) + parameters

    for layer in range(2, num_layers): # Loop through layers 2 to (num_layers - 1
        # Activation from the previous layer
        forward_propagation['A' + str(layer - 1)] = activation_function(forward_p

        # Linear transformation for the current layer
        forward_propagation['Z' + str(layer)] = np.dot(parameters['W' + str(layer

    # Compute the final layer's activation
    forward_propagation['A' + str(num_layers - 1)] = activation_function(forward_
    forward_propagation['Z' + str(num_layers)] = np.dot(parameters['W' + str(num_

    # Output layer: apply softmax
    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_data, labels, cache):
    num_examples = input_data.shape[1]  # Number of examples in the batch (m)

    # Extract activations and parameters from cache
    forward_propagation, parameters = cache
    num_layers = len(parameters) // 2  # Number of layers (assuming W1, b1, ...,

    # Initialize a dictionary to store gradients
    grads = {}

    # Output layer gradient
    grads['dZ' + str(num_layers)] = forward_propagation['A' + str(num_layers)] -
    grads['dW' + str(num_layers)] = (1. / num_examples) * np.dot(grads['dZ' + str
    grads['db' + str(num_layers)] = (1. / num_examples) * np.sum(grads['dZ' + str

    # Backpropagate through all hidden layers (in reverse order)
    for layer in range(num_layers-1, 1, -1):
        # Compute gradients for weights and biases
        grads['dA' + str(layer)] = np.dot(parameters['W' + str(layer + 1)].T, gra
        grads['dZ' + str(layer)] = grads['dA' + str(layer)] * tanh_derivative(for
```

```
        grads['dW' + str(layer)] = (1. / num_examples) * np.dot(grads['dZ' + str(
        grads['db' + str(layer)] = (1. / num_examples) * np.sum(grads['dZ' + str(

    # First layer gradient
    grads['dA1'] = np.dot(parameters['W2'].T, grads['dZ2'])
    grads['dZ1'] = grads['dA1'] * tanh_derivative(forward_propagation['Z1'])
    grads['dW1'] = (1. / num_examples) * np.dot(grads['dZ1'], input_data.T)
    grads['db1'] = (1. / num_examples) * np.sum(grads['dZ1'], axis=1, keepdims=Tr

    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):
        # Update weights with regularization (if lambd > 0)
        parameters["W" + str(layer + 1)] -= (learning_rate * (grads["dW" + str(la

        # Update biases (biases are not regularized)
        parameters["b" + str(layer + 1)] -= (learning_rate * grads["db" + str(lay

    return parameters
```

Cost Funtion

```
def cross_entropy_cost(predictions, labels, epsilon=1e-10):
    # Ensure predictions are clipped to avoid log(0)
    predictions = np.clip(predictions, epsilon, 1. - epsilon)

    # Compute the multi-class cross-entropy loss
    loss_per_example = -np.sum(labels * np.log(predictions), axis=0)

    # Average the loss over all examples
    cost = np.mean(loss_per_example)

    return cost
```

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

```
            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
    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 = f"Epoch={epoch},alpha={learning_rate},Regularization={lambd},Batch={
print("Model Key: " + model_name)

# Train the model and store the learned parameters
learned_parameters[model_name] = model(
    train_dataloader,
    test_data,
    batch_size=batch_size,
    learning_rate=learning_rate,
    epoch=epoch,
    print_cost=True,
    lambd=lambd,
    initial=initial
)

# Find the model with 'zero' initialization dynamically
model_key = [key for key in learned_parameters.keys() if "Initialization=zero" in

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

```
Model Key: Epoch=15,alpha=0.01,Regularization=0,Batch=64,Initialization=zero
Cost after epoch 0, iteration 0: Train Cost: 2.302585092994045, Test Cost: 2
Cost after epoch 0, iteration 200: Train Cost: 2.300873768295059, Test Cost:
Cost after epoch 0, iteration 400: Train Cost: 2.3013808147462433, Test Cost:
Cost after epoch 0, iteration 600: Train Cost: 2.299218592730832, Test Cost:
Cost after epoch 0, iteration 800: Train Cost: 2.303380887710829, Test Cost:
Cost after epoch 1, iteration 0: Train Cost: 2.304105200870981, Test Cost: 2.
Cost after epoch 1, iteration 200: Train Cost: 2.3025125842634337, Test Cost:
Cost after epoch 1, iteration 400: Train Cost: 2.3038835123980665, Test Cost:
Cost after epoch 1, iteration 600: Train Cost: 2.2957127574363985, Test Cost:
Cost after epoch 1, iteration 800: Train Cost: 2.3065381739559534, Test Cost:
Cost after epoch 2, iteration 0: Train Cost: 2.303211120130456, Test Cost: 2
Cost after epoch 2, iteration 200: Train Cost: 2.3028412334212556, Test Cost:
Cost after epoch 2, iteration 400: Train Cost: 2.3026754594360366, Test Cost:
Cost after epoch 2, iteration 600: Train Cost: 2.306183782993075, Test Cost:
Cost after epoch 2, iteration 800: Train Cost: 2.3052988692213088, Test Cost:
Cost after epoch 3, iteration 0: Train Cost: 2.289194025805478, Test Cost: 2.
Cost after epoch 3, iteration 200: Train Cost: 2.2947885807525434, Test Cost:
Cost after epoch 3, iteration 400: Train Cost: 2.303609289205146, Test Cost:
Cost after epoch 3, iteration 600: Train Cost: 2.2987016321068996, Test Cost:
Cost after epoch 3, iteration 800: Train Cost: 2.3009818996937916, Test Cost:
Cost after epoch 4, iteration 0: Train Cost: 2.309983706491201, Test Cost: 2.
Cost after epoch 4, iteration 200: Train Cost: 2.3077959352817508, Test Cost:
Cost after epoch 4, iteration 400: Train Cost: 2.3001370449552443, Test Cost:
Cost after epoch 4, iteration 600: Train Cost: 2.3110053198614136, Test Cost:
Cost after epoch 4, iteration 800: Train Cost: 2.3037528632899416, Test Cost:
Cost after epoch 5, iteration 0: Train Cost: 2.3074160338517657, Test Cost: 2
Cost after epoch 5, iteration 200: Train Cost: 2.3031978283300667, Test Cost:
Cost after epoch 5, iteration 400: Train Cost: 2.301650626344944, Test Cost:
Cost after epoch 5, iteration 600: Train Cost: 2.292385885511609, Test Cost:
Cost after epoch 5, iteration 800: Train Cost: 2.3061262168202035, Test Cost:
Cost after epoch 6, iteration 0: Train Cost: 2.310975043412473, Test Cost: 2.
Cost after epoch 6, iteration 200: Train Cost: 2.2994587320868556, Test Cost:
Cost after epoch 6, iteration 400: Train Cost: 2.303212687896452, Test Cost:
Cost after epoch 6, iteration 600: Train Cost: 2.2956294429057396, Test Cost:
Cost after epoch 6, iteration 800: Train Cost: 2.3173734356296234, Test Cost:
Cost after epoch 7, iteration 0: Train Cost: 2.2985786099683523, Test Cost: 2
Cost after epoch 7, iteration 200: Train Cost: 2.313649529280384, Test Cost:
Cost after epoch 7, iteration 400: Train Cost: 2.2950209445515677, Test Cost:
Cost after epoch 7, iteration 600: Train Cost: 2.3072287440386505, Test Cost:
Cost after epoch 7, iteration 800: Train Cost: 2.2971980454701257, Test Cost:
Cost after epoch 8, iteration 0: Train Cost: 2.308782953944934, Test Cost: 2.
Cost after epoch 8, iteration 200: Train Cost: 2.2939717313269936, Test Cost:
Cost after epoch 8, iteration 400: Train Cost: 2.3085310145584392, Test Cost:
Cost after epoch 8, iteration 600: Train Cost: 2.3033268559461058, Test Cost:
Cost after epoch 8, iteration 800: Train Cost: 2.3061480097885863, Test Cost:
Cost after epoch 9, iteration 0: Train Cost: 2.284848438935216, Test Cost: 2.
Cost after epoch 9, iteration 200: Train Cost: 2.2968038765364023, Test Cost:
Cost after epoch 9, iteration 400: Train Cost: 2.29706863035338, Test Cost: 2
Cost after epoch 9, iteration 600: Train Cost: 2.2926848573901646, Test Cost:
Cost after epoch 9, iteration 800: Train Cost: 2.296732614248952, Test Cost:
Cost after epoch 10, iteration 0: Train Cost: 2.295187895659339, Test Cost: 2
```

```
Cost after epoch 10, iteration 200: Train Cost: 2.2939568118830733, Test Cost
Cost after epoch 10, iteration 400: Train Cost: 2.3093768361704035, Test Cost
Cost after epoch 10, iteration 600: Train Cost: 2.3071627030187756, Test Cost
```

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

```
Cost after epoch 11, iteration 200: Train Cost: 2.3156088299520903, Test Cost
```

```
# 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 = f"Epoch={epoch},alpha={learning_rate},Regularization={lambd},Batch={
print("Model Key: " + model_name)

# Train the model and store the learned parameters
learned_parameters[model_name] = model(
    train_dataloader,
    test_data,
    batch_size=batch_size,
    learning_rate=learning_rate,
    epoch=epoch,
    print_cost=True,
    lambd=lambd,
    initial=initial
)

# Find the model with 'zero' initialization dynamically
model_key = [key for key in learned_parameters.keys() if "Initialization=random"

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

```
Model Key: Epoch=15,alpha=0.01,Regularization=0,Batch=64,Initialization=randor
Cost after epoch 0, iteration 0: Train Cost: 2.299490621790012, Test Cost: 2.
Cost after epoch 0, iteration 200: Train Cost: 0.9619640599176462, Test Cost:
Cost after epoch 0, iteration 400: Train Cost: 0.7531023651961821, Test Cost:
Cost after epoch 0, iteration 600: Train Cost: 0.5872091773964412, Test Cost:
Cost after epoch 0, iteration 800: Train Cost: 0.46487822468113765, Test Cost
Cost after epoch 1, iteration 0: Train Cost: 0.4799072757315818, Test Cost: 0
Cost after epoch 1, iteration 200: Train Cost: 0.3600640954258434, Test Cost:
Cost after epoch 1, iteration 400: Train Cost: 0.5171792036145257, Test Cost:
Cost after epoch 1, iteration 600: Train Cost: 0.4895562544127641, Test Cost:
Cost after epoch 1, iteration 800: Train Cost: 0.37846450151691596, Test Cost
Cost after epoch 2, iteration 0: Train Cost: 0.30153244918016664, Test Cost: (
Cost after epoch 2, iteration 200: Train Cost: 0.349683487399958, Test Cost: (
```

```
Cost after epoch 2, iteration 200: Train Cost: 0.34968348739995?, Test Cost: (
Cost after epoch 2, iteration 400: Train Cost: 0.3516677795866071, Test Cost:
Cost after epoch 2, iteration 600: Train Cost: 0.1772396593196529, Test Cost:
Cost after epoch 2, iteration 800: Train Cost: 0.33046677384594925, Test Cost
Cost after epoch 3, iteration 0: Train Cost: 0.26231410983496406, Test Cost: (
Cost after epoch 3, iteration 200: Train Cost: 0.17105534659184116, Test Cost
Cost after epoch 3, iteration 400: Train Cost: 0.2310566934315167, Test Cost:
Cost after epoch 3, iteration 600: Train Cost: 0.27548477458096066, Test Cost
Cost after epoch 3, iteration 800: Train Cost: 0.33388597865542624, Test Cost
Cost after epoch 4, iteration 0: Train Cost: 0.3553443989950801, Test Cost: 0
Cost after epoch 4, iteration 200: Train Cost: 0.27426389695195547, Test Cost
Cost after epoch 4, iteration 400: Train Cost: 0.2575072987413438, Test Cost:
Cost after epoch 4, iteration 600: Train Cost: 0.22272883820322198, Test Cost
Cost after epoch 4, iteration 800: Train Cost: 0.45481950895213696, Test Cost
Cost after epoch 5, iteration 0: Train Cost: 0.18649801384295767, Test Cost: (
Cost after epoch 5, iteration 200: Train Cost: 0.19558094323994213, Test Cost
Cost after epoch 5, iteration 400: Train Cost: 0.2583182113233412, Test Cost:
Cost after epoch 5, iteration 600: Train Cost: 0.123061745707543, Test Cost: (
Cost after epoch 5, iteration 800: Train Cost: 0.11333487876249332, Test Cost
Cost after epoch 6, iteration 0: Train Cost: 0.15684946112310869, Test Cost: (
Cost after epoch 6, iteration 200: Train Cost: 0.38647016514535526, Test Cost
Cost after epoch 6, iteration 400: Train Cost: 0.18964686266826375, Test Cost
Cost after epoch 6, iteration 600: Train Cost: 0.28841795194015374, Test Cost
Cost after epoch 6, iteration 800: Train Cost: 0.15758610814752522, Test Cost
Cost after epoch 7, iteration 0: Train Cost: 0.13476500072119352, Test Cost: (
Cost after epoch 7, iteration 200: Train Cost: 0.11078212163972107, Test Cost
Cost after epoch 7, iteration 400: Train Cost: 0.1188678474360543, Test Cost:
Cost after epoch 7, iteration 600: Train Cost: 0.08180431780396213, Test Cost
Cost after epoch 7, iteration 800: Train Cost: 0.19062507628481898, Test Cost
Cost after epoch 8, iteration 0: Train Cost: 0.18499691916997585, Test Cost: (
Cost after epoch 8, iteration 200: Train Cost: 0.12290036442748756, Test Cost
Cost after epoch 8, iteration 400: Train Cost: 0.3558304294354943, Test Cost:
Cost after epoch 8, iteration 600: Train Cost: 0.4738405087335171, Test Cost:
Cost after epoch 8, iteration 800: Train Cost: 0.308208496802502, Test Cost: (
Cost after epoch 9, iteration 0: Train Cost: 0.1794536428421953, Test Cost: 0
Cost after epoch 9, iteration 200: Train Cost: 0.2150863367933371, Test Cost:
Cost after epoch 9, iteration 400: Train Cost: 0.2290152876134442, Test Cost:
Cost after epoch 9, iteration 600: Train Cost: 0.08594577350199381, Test Cost
Cost after epoch 9, iteration 800: Train Cost: 0.2959289821462331, Test Cost:
Cost after epoch 10, iteration 0: Train Cost: 0.1774168352456043, Test Cost: (
Cost after epoch 10, iteration 200: Train Cost: 0.21172033686038774, Test Cos
Cost after epoch 10, iteration 400: Train Cost: 0.14291337236722745, Test Cos
Cost after epoch 10, iteration 600: Train Cost: 0.07859936388767755, Test Cos
```

```
# 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 = f"Epoch={epoch},alpha={learning_rate},Regularization={lambd},Batch={
print("Model Key: " + model_name)

# Train the model and store the learned parameters
learned_parameters[model_name] = model(
    train_dataloader,
    test_data,
    batch_size=batch_size,
    learning_rate=learning_rate,
    epoch=epoch,
    print_cost=True,
    lambd=lambd,
    initial=initial
)

# Find the model with 'zero' initialization dynamically
model_key = [key for key in learned_parameters.keys() if "Initialization=glorot"

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

```
Model Key: Epoch=15,alpha=0.01,Regularization=0,Batch=64,Initialization=gloro
Cost after epoch 0, iteration 0: Train Cost: 2.402181451199647, Test Cost: 2.
Cost after epoch 0, iteration 200: Train Cost: 0.7597189753021367, Test Cost:
Cost after epoch 0, iteration 400: Train Cost: 0.5009029235565605, Test Cost:
Cost after epoch 0, iteration 600: Train Cost: 0.427422233175055, Test Cost:
Cost after epoch 0, iteration 800: Train Cost: 0.38994489518285935, Test Cost
Cost after epoch 1, iteration 0: Train Cost: 0.4826853395253341, Test Cost: 0
Cost after epoch 1, iteration 200: Train Cost: 0.44253956101279707, Test Cost
Cost after epoch 1, iteration 400: Train Cost: 0.2425591228097011, Test Cost:
Cost after epoch 1, iteration 600: Train Cost: 0.2964783136875335, Test Cost:
Cost after epoch 1, iteration 800: Train Cost: 0.47316717557677773, Test Cost
Cost after epoch 2, iteration 0: Train Cost: 0.31883396369486156, Test Cost:
Cost after epoch 2, iteration 200: Train Cost: 0.33098906570317876, Test Cost
Cost after epoch 2, iteration 400: Train Cost: 0.49820865500600425, Test Cost
Cost after epoch 2, iteration 600: Train Cost: 0.4735931939065407, Test Cost:
Cost after epoch 2, iteration 800: Train Cost: 0.19637426103510758, Test Cost
Cost after epoch 3, iteration 0: Train Cost: 0.25428618986432244, Test Cost:
Cost after epoch 3, iteration 200: Train Cost: 0.10418063004790451, Test Cost
Cost after epoch 3, iteration 400: Train Cost: 0.26636177102013947, Test Cost
Cost after epoch 3, iteration 600: Train Cost: 0.23965492580599812, Test Cost
Cost after epoch 3, iteration 800: Train Cost: 0.09725358317125839, Test Cost
Cost after epoch 4, iteration 0: Train Cost: 0.2724025000548532, Test Cost: 0
Cost after epoch 4, iteration 200: Train Cost: 0.20934943292364233, Test Cost
Cost after epoch 4, iteration 400: Train Cost: 0.38847068699429826, Test Cost
Cost after epoch 4, iteration 600: Train Cost: 0.2022389839392051, Test Cost:
Cost after epoch 4, iteration 800: Train Cost: 0.16979512503677308, Test Cost
Cost after epoch 5, iteration 0: Train Cost: 0.25499758985103427, Test Cost:
Cost after epoch 5, iteration 200: Train Cost: 0.16186140062106374, Test Cost
```

```
Cost after epoch 5, iteration 400: Train Cost: 0.39503618297028126, Test Cost
Cost after epoch 5, iteration 600: Train Cost: 0.21593375059289632, Test Cost
Cost after epoch 5, iteration 800: Train Cost: 0.12725798991789947, Test Cost
Cost after epoch 6, iteration 0: Train Cost: 0.32092111683308294, Test Cost:
Cost after epoch 6, iteration 200: Train Cost: 0.28592695461423867, Test Cost
Cost after epoch 6, iteration 400: Train Cost: 0.29769566198442876, Test Cost
Cost after epoch 6, iteration 600: Train Cost: 0.16449316784635798, Test Cost
Cost after epoch 6, iteration 800: Train Cost: 0.12579928544725183, Test Cost
Cost after epoch 7, iteration 0: Train Cost: 0.1619593096705585, Test Cost: 0
Cost after epoch 7, iteration 200: Train Cost: 0.20507146549628166, Test Cost
Cost after epoch 7, iteration 400: Train Cost: 0.10656241042473534, Test Cost
Cost after epoch 7, iteration 600: Train Cost: 0.15971868357125407, Test Cost
Cost after epoch 7, iteration 800: Train Cost: 0.27069430323773014, Test Cost
Cost after epoch 8, iteration 0: Train Cost: 0.14110659918473512, Test Cost:
Cost after epoch 8, iteration 200: Train Cost: 0.1499146309263445, Test Cost:
Cost after epoch 8, iteration 400: Train Cost: 0.1452026572402019, Test Cost:
Cost after epoch 8, iteration 600: Train Cost: 0.2531904203783415, Test Cost:
Cost after epoch 8, iteration 800: Train Cost: 0.08089478055168221, Test Cost
Cost after epoch 9, iteration 0: Train Cost: 0.034721871346365625, Test Cost:
Cost after epoch 9, iteration 200: Train Cost: 0.07984070069528976, Test Cost
Cost after epoch 9, iteration 400: Train Cost: 0.2592173850245802, Test Cost:
Cost after epoch 9, iteration 600: Train Cost: 0.21443737925160777, Test Cost
Cost after epoch 9, iteration 800: Train Cost: 0.08720348970947939, Test Cost
Cost after epoch 10, iteration 0: Train Cost: 0.29499434266156843, Test Cost:
Cost after epoch 10, iteration 200: Train Cost: 0.11347382238172968, Test Cos
Cost after epoch 10, iteration 400: Train Cost: 0.19836739898068292, Test Cos
Cost after epoch 10, iteration 600: Train Cost: 0.0605396555361132, Test Cost
```

Start coding or generate with AI.

```
Cost after epoch 11, iteration 200: Train Cost: 0.06979486986593889, Test Cos
Cost after epoch 11, iteration 400: Train Cost: 0.10440395277775749, Test Cos
Cost after epoch 11, iteration 600: Train Cost: 0.18049221975924803, Test Cos
Cost after epoch 11, iteration 800: Train Cost: 0.3113889831580931, Test Cost
Cost after epoch 12, iteration 0: Train Cost: 0.04696197595800954, Test Cost:
Cost after epoch 12, iteration 200: Train Cost: 0.11934676436102891, Test Cos
Cost after epoch 12, iteration 400: Train Cost: 0.25422790813308854, Test Cos
Cost after epoch 12, iteration 600: Train Cost: 0.2684953924249412, Test Cost
Cost after epoch 12, iteration 800: Train Cost: 0.1400247137725533, Test Cost
Cost after epoch 13, iteration 0: Train Cost: 0.21678937513764004, Test Cost:
Cost after epoch 13, iteration 200: Train Cost: 0.14177121377555868, Test Cos
Cost after epoch 13, iteration 400: Train Cost: 0.14907144646343734, Test Cos
Cost after epoch 13, iteration 600: Train Cost: 0.08575965510943112, Test Cos
Cost after epoch 13, iteration 800: Train Cost: 0.0851903363221394, Test Cost
Cost after epoch 14, iteration 0: Train Cost: 0.11076892156523983, Test Cost:
Cost after epoch 14, iteration 200: Train Cost: 0.12449395464191965, Test Cos
Cost after epoch 14, iteration 400: Train Cost: 0.29909108217457386, Test Cos
Cost after epoch 14, iteration 600: Train Cost: 0.06103076181148288, Test Cos
Cost after epoch 14, iteration 800: Train Cost: 0.14900136915978465, Test Cos
```

Test acc. = 95.99%, Train acc. = 96.49%