Name: Aditya Pradeep Waghmode

Z Number: Z23737910

Subject: Data Mining and Machince Learning

Project: Mini Project 2

Answer 1:

Loading the data of digits and converting from .mat to python accpectable data

Google Colab_link: https://colab.research.google.com/drive/1HXKeP7qlcEhXJqKgQV2UQoTXomYJ-b6y?usp=sharing

```
from google.colab import drive
drive.mount('/content/drive')

import scipy.io
data = scipy.io.loadmat('/content/drive/My Drive/digits.mat')

import numpy as np

# Reshaping the training and testing data
train_images = data['train'].reshape(28, 28, -1, order='F')
test_images = data['test'].reshape(28, 28, -1, order='F')

# Checking the shape of reshaped training and test images
train_images.shape, test_images.shape
```

```
((28, 28, 5000), (28, 28, 1000))
```

Defining all the function used in perceptorn

```
def initialize_weights(input_dim):
    # Initialize the weights for the perceptron.
    np.random.seed(42)  # For reproducibility
    return np.random.randn(input_dim, 1) * 0.01

def preprocess_labels(labels, target_digit):
```

```
    # Preprocess labels to +1 for target digit and -1 for others.
    return np.where(labels == target_digit, 1, -1)

def train_perceptron(X, y, learning_rate, epochs, report_every):
    # Train a single-layer perceptron.
    weights = initialize_weights(X.shape[0])  # Initialize weights
    n_samples = X.shape[1]
    error_history = []

    for epoch in range(epochs):
        total_error = 0
        for i in range(n_samples):
            xi = X[:, i:i+1]
            yi = y[i]
            output = np.sign(np.dot(weights.T, xi))
            update = learning_rate * (yi - output) * xi
            weights += update
            total_error += int(output != yi)

        if (epoch + 1) % report_every == 0 or epoch == epochs - 1:
            print(f"Epoch {epoch + 1}/{epochs}, Error: {total_error}")
            error_history.append(total_error)
            # Adjust learning rate
            learning_rate *= 0.9

    return weights, error_history
```

For 0: Train, Test and Visiualzation

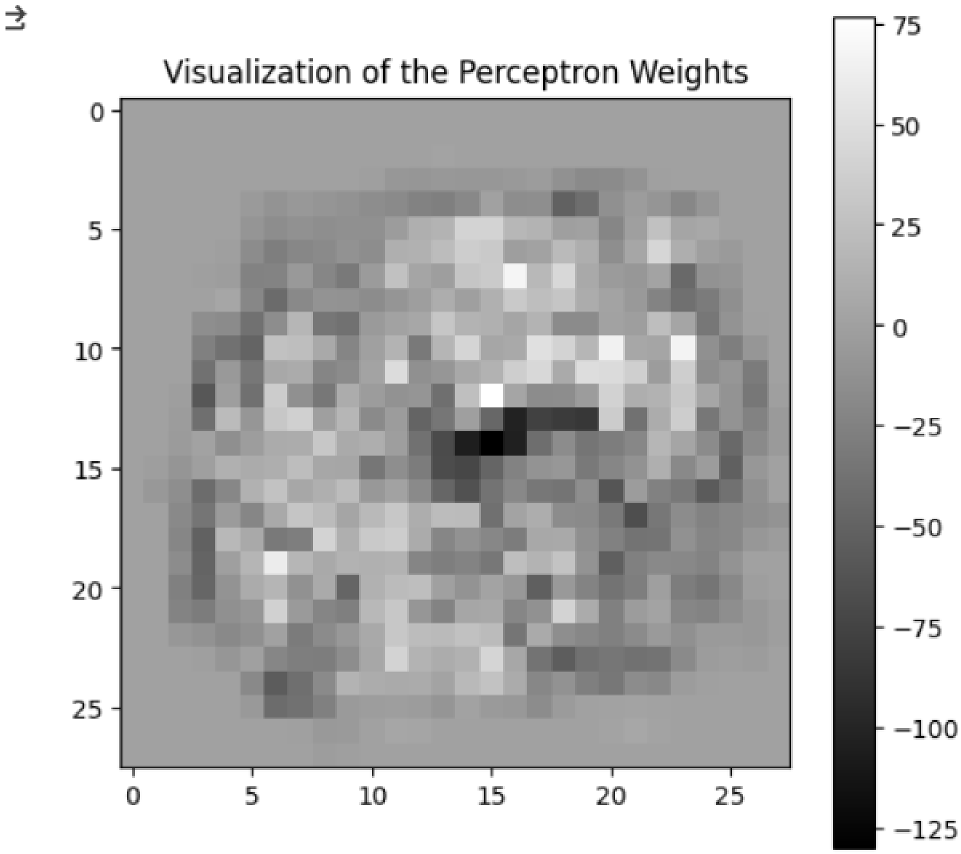
```
# Prepare data
X_train = train_images.reshape(-1, train_images.shape[2], order='F')
y_train = preprocess_labels(data['trainlabels'], 0)  # Preprocess labels
for digit '0'
X_test = test_images.reshape(-1, test_images.shape[2], order='F')
y_test = preprocess_labels(data['testlabels'], 0)  # Preprocess labels for
digit '0'

# Train the perceptron
weights, error_history = train_perceptron(X_train, y_train,
learning_rate=1e-2, epochs=100, report_every=1)

print("Shape of weights for zero:",weights.shape)
```

```python
print("Error history for dectecting 0:",error_history)
```

```python
def evaluate_perceptron(X, y, weights):
    # Evaluate the perceptron model on a given dataset.
    outputs = np.sign(np.dot(weights.T, X))
    accuracy = np.mean(outputs == y) * 100
    return accuracy

# Evaluate the perceptron on the test set
test_accuracy = evaluate_perceptron(X_test, y_test, weights)

print("Test accuracy for dectecting 0:",test_accuracy)
```

```
Test accuracy for dectecting 0: 84.36200000000001
```

```python
import matplotlib.pyplot as plt


# Reshaping the weights to visualize them as an image
weights_image = weights.reshape(28, 28)

plt.figure(figsize=(6, 6))
plt.imshow(weights_image, cmap='gray')
plt.colorbar()
plt.title('Visualization of the Perceptron Weights')
plt.show()
```



For 8 Train, Test and Visiualzation

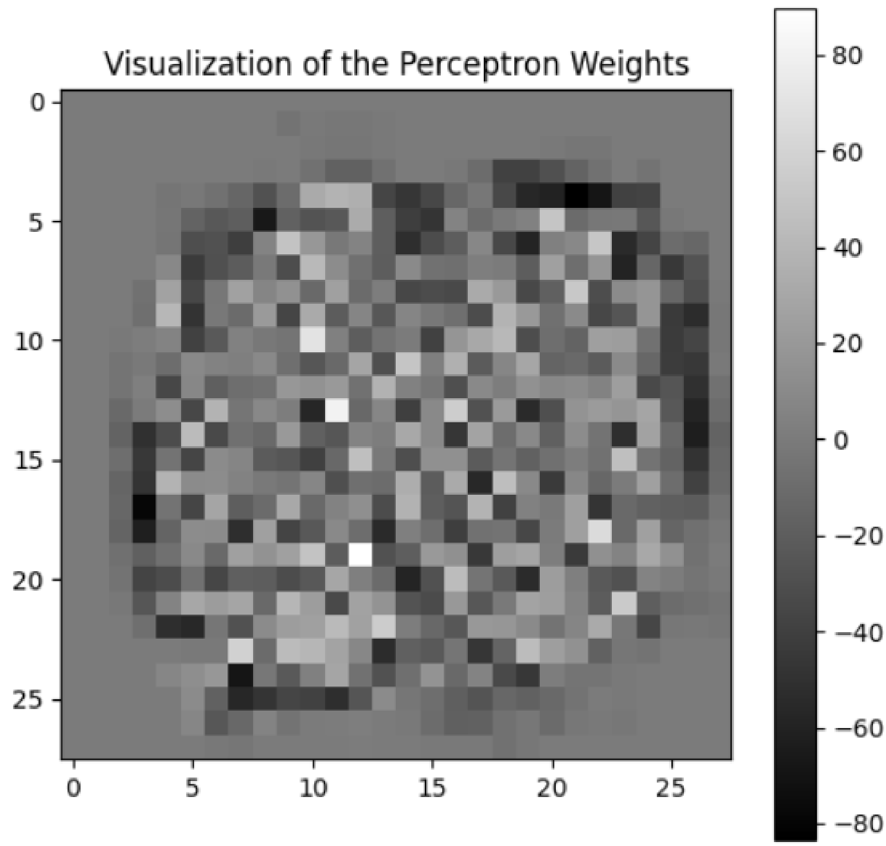
```
# Prepare data
X_train_8 = train_images.reshape(-1, train_images.shape[2], order='F')
y_train_8 = preprocess_labels(data['trainlabels'], 8)  # Preprocess labels
for digit '8'
X_test_8 = test_images.reshape(-1, test_images.shape[2], order='F')
y_test_8 = preprocess_labels(data['testlabels'], 8)  # Preprocess labels
for digit '8'

# Train the perceptron
weights, error_history = train_perceptron(X_train_8, y_train_8,
learning_rate=0.01, epochs=100, report_every=1)

print("Shape of weights for Eight:",weights.shape)
print("Error history for dectecting 8:",error_history)
```

```
def evaluate_perceptron(X, y, weights):
```

```python
    # Evaluate the perceptron model on a given dataset.
    outputs = np.sign(np.dot(weights.T, X))
    accuracy = np.mean(outputs == y) * 100
    return accuracy

# Evaluate the perceptron on the test set
test_accuracy = evaluate_perceptron(X_test_8, y_test_8, weights)

print("Test accuracy for dectecting 8:",test_accuracy)
```

```
Test accuracy for dectecting 8: 81.4826
```

```python
import matplotlib.pyplot as plt


# Reshaping the weights to visualize them as an image
weights_image = weights.reshape(28, 28)

plt.figure(figsize=(6, 6))
plt.imshow(weights_image, cmap='gray')
plt.colorbar()
plt.title('Visualization of the Perceptron Weights')
plt.show()
```



For 1 Train, Test and Visiualzation

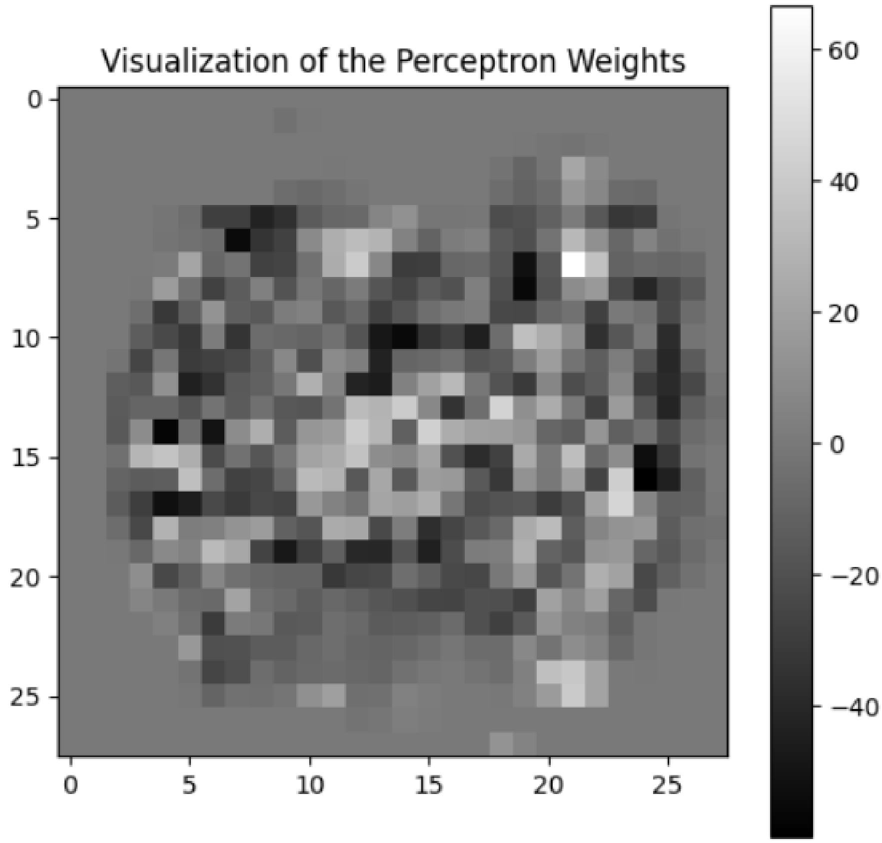
```
# Prepare data
X_train = train_images.reshape(-1, train_images.shape[2], order='F')
y_train = preprocess_labels(data['trainlabels'], 1)  # Preprocess labels
for digit '1'
X_test = test_images.reshape(-1, test_images.shape[2], order='F')
y_test = preprocess_labels(data['testlabels'], 1)  # Preprocess labels for
digit '1'

# Train the perceptron
weights, error_history = train_perceptron(X_train, y_train,
learning_rate=0.01, epochs=100, report_every=1)

print("Shape of weights for One:",weights.shape)
print("Error history for dectecting 1:",error_history)
```

```
def evaluate_perceptron(X, y, weights):
```

```python
    # Evaluate the perceptron model on a given dataset.
    outputs = np.sign(np.dot(weights.T, X))
    accuracy = np.mean(outputs == y) * 100
    return accuracy

# Evaluate the perceptron on the test set
test_accuracy = evaluate_perceptron(X_test, y_test, weights)

print("Test accuracy for dectecting 1:",test_accuracy)
```

```
Test accuracy for dectecting 1: 77.676
```

```python
import matplotlib.pyplot as plt


# Reshaping the weights to visualize them as an image
weights_image = weights.reshape(28, 28)  # Use all weights

plt.figure(figsize=(6, 6))
plt.imshow(weights_image, cmap='gray')
plt.colorbar()
plt.title('Visualization of the Perceptron Weights')
plt.show()
```



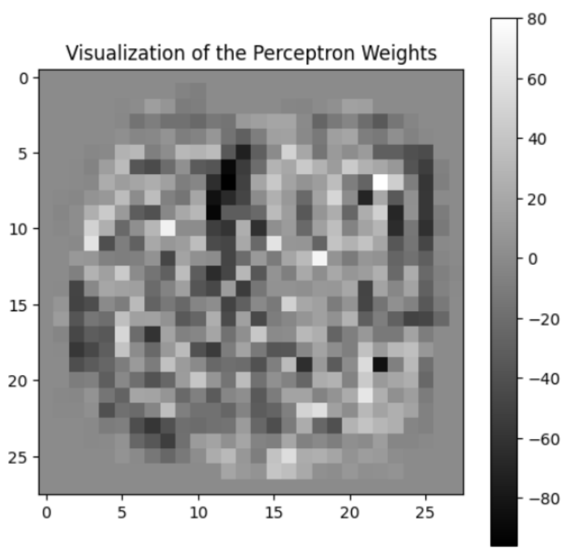
For 2 Train, Test and Visiualzation

```
# Prepare data
X_train = train_images.reshape(-1, train_images.shape[2], order='F')   #
Flatten images to 784x5000
y_train = preprocess_labels(data['trainlabels'], 2)   # Preprocess labels
for digit '2'
X_test = test_images.reshape(-1, test_images.shape[2], order='F')   #
Flatten images to 784x1000
y_test = preprocess_labels(data['testlabels'], 2)   # Preprocess labels for
digit '2'

# Train the perceptron
weights, error_history = train_perceptron(X_train, y_train,
learning_rate=0.01, epochs=100, report_every=1)

print("Shape of weights for Two:",weights.shape)
print("Error history for dectecting 2:",error_history)
```

```python
def evaluate_perceptron(X, y, weights):
    """Evaluate the perceptron model on a given dataset."""
    outputs = np.sign(np.dot(weights.T, X))
    accuracy = np.mean(outputs == y) * 100
    return accuracy

# Evaluate the perceptron on the test set
test_accuracy = evaluate_perceptron(X_test, y_test, weights)

print("Test accuracy for dectecting 2:",test_accuracy)
```

```
Test accuracy for dectecting 2: 79.1072
```

```python
import matplotlib.pyplot as plt


# Correctly reshape the weights to visualize them as an image
weights_image = weights.reshape(28, 28)  # Use all weights

plt.figure(figsize=(6, 6))
plt.imshow(weights_image, cmap='gray')
plt.colorbar()
plt.title('Visualization of the Perceptron Weights')
plt.show()
```



The system is operating steadily. The imbalance in the dataset, which results in a significantly smaller number of positive test cases than negative test instances, can be used to explain this behavior. As a result, the model was unable to detect the number's presence and only trained to predict its absence (ZERO). It always predicts the out as -1 as a result. Because of this, the MSE seems to be steady.

Answer 2:

```
import numpy as np
import scipy.io

# Extract variables
train_data = data['train']
train_labels = data['trainlabels']
test_data = data['test']
test_labels = data['testlabels']

# Normalizing pixel values
train_data_normalized = train_data / 255
test_data_normalized = test_data / 255

num_classes = 10
train_labels_onehot = np.zeros((num_classes, len(train_labels)))
for i in range(len(train_labels)):
    train_labels_onehot[train_labels[i], i] = 1

# Definin Hyperparameters
input_size = 784
hidden_size = 25
output_size = num_classes
learning_rate = 0.01
num_epochs = 100

# Initializing weights and biases
weights_input_hidden = np.random.randn(hidden_size, input_size)
biases_input_hidden = np.random.randn(hidden_size, 1)
weights_hidden_output = np.random.randn(output_size, hidden_size)
biases_hidden_output = np.random.randn(output_size, 1)

# Sigmoid function
sigmoid = lambda x: 1.0 / (1.0 + np.exp(-x))

for epoch in range(num_epochs):
    epoch_errors = []
    num_correct = 0
    for sample in range(5000):
        # Forward propagation
        x = train_data_normalized[:, sample].reshape(-1, 1)
        hidden_layer_input = np.dot(weights_input_hidden, x) +
biases_input_hidden
```

```
        hidden_layer_output = sigmoid(hidden_layer_input)
        output_layer_input = np.dot(weights_hidden_output,
hidden_layer_output) + biases_hidden_output
        output_layer_output = sigmoid(output_layer_input)
        output_error = output_layer_output - train_labels_onehot[:,
sample].reshape(-1, 1)
        hidden_error = np.dot(weights_hidden_output.T, output_error) *
(hidden_layer_output * (1 - hidden_layer_output))

        # Backpropagation
        weights_hidden_output -= learning_rate * np.dot(output_error,
hidden_layer_output.T)
        biases_hidden_output -= learning_rate * output_error
        weights_input_hidden -= learning_rate * np.dot(hidden_error, x.T)
        biases_input_hidden -= learning_rate * hidden_error

        # Calculating epoch error (MSE)
        epoch_errors.append(np.mean(output_error ** 2))

        # Calculating accuracy
        predicted_class = np.argmax(output_layer_output)
        true_class = np.argmax(train_labels_onehot[:, sample])
        if predicted_class == true_class:
            num_correct += 1

    epoch_accuracy = num_correct / 5000
    mean_epoch_error = np.mean(epoch_errors)
    print(f'Epoch {epoch + 1}: MSE={mean_epoch_error:.4f},
Accuracy={epoch_accuracy * 100:.2f}%')

# Testing
test_labels_onehot = np.zeros((num_classes, len(test_labels)))
for i in range(len(test_labels)):
    test_labels_onehot[test_labels[i], i] = 1

num_correct = 0
for sample in range(1000):  # Assuming 'test' has 1000 examples
    x_test = test_data_normalized[:, sample].reshape(-1, 1)
    hidden_layer_input_test = np.dot(weights_input_hidden, x_test) +
biases_input_hidden
    hidden_layer_output_test = sigmoid(hidden_layer_input_test)
    output_layer_input_test = np.dot(weights_hidden_output,
hidden_layer_output_test) + biases_hidden_output
    output_layer_output_test = sigmoid(output_layer_input_test)
```

```python
    predicted_class = np.argmax(output_layer_output_test)
    true_class = np.argmax(test_labels_onehot[:, sample])

    if predicted_class == true_class:
        num_correct += 1

# Calculating accuracy
accuracy = num_correct / 1000
print(f'Testing Accuracy: {accuracy * 100:.2f}%')
```

```
Epoch 1: MSE=0.0820, Accuracy=38.70%
Epoch 2: MSE=0.0552, Accuracy=64.04%
Epoch 3: MSE=0.0443, Accuracy=72.88%
Epoch 4: MSE=0.0375, Accuracy=76.98%
Epoch 5: MSE=0.0327, Accuracy=80.46%
Epoch 6: MSE=0.0292, Accuracy=82.78%
Epoch 7: MSE=0.0266, Accuracy=84.64%
Epoch 8: MSE=0.0246, Accuracy=85.60%
Epoch 9: MSE=0.0230, Accuracy=86.50%
Epoch 10: MSE=0.0216, Accuracy=87.20%
Epoch 11: MSE=0.0205, Accuracy=88.00%
Epoch 12: MSE=0.0195, Accuracy=88.56%
Epoch 13: MSE=0.0186, Accuracy=89.42%
Epoch 14: MSE=0.0178, Accuracy=90.06%
Epoch 15: MSE=0.0171, Accuracy=90.40%
Epoch 16: MSE=0.0164, Accuracy=90.86%
Epoch 17: MSE=0.0157, Accuracy=91.30%
Epoch 18: MSE=0.0151, Accuracy=91.78%
Epoch 19: MSE=0.0146, Accuracy=92.04%
Epoch 20: MSE=0.0141, Accuracy=92.42%
Epoch 21: MSE=0.0136, Accuracy=92.68%
Epoch 22: MSE=0.0132, Accuracy=92.84%
Epoch 23: MSE=0.0127, Accuracy=93.04%
Epoch 24: MSE=0.0123, Accuracy=93.32%
Epoch 25: MSE=0.0120, Accuracy=93.58%
Epoch 26: MSE=0.0116, Accuracy=93.88%
Epoch 27: MSE=0.0113, Accuracy=94.12%
Epoch 28: MSE=0.0110, Accuracy=94.24%
Epoch 29: MSE=0.0107, Accuracy=94.40%
Epoch 30: MSE=0.0104, Accuracy=94.66%
Epoch 31: MSE=0.0102, Accuracy=94.68%
Epoch 32: MSE=0.0099, Accuracy=94.74%
Epoch 33: MSE=0.0097, Accuracy=94.88%
Epoch 34: MSE=0.0094, Accuracy=95.12%
Epoch 35: MSE=0.0092, Accuracy=95.28%
Epoch 36: MSE=0.0090, Accuracy=95.46%
Epoch 37: MSE=0.0088, Accuracy=95.54%
Epoch 38: MSE=0.0086, Accuracy=95.60%
Epoch 39: MSE=0.0084, Accuracy=95.72%
Epoch 40: MSE=0.0082, Accuracy=95.80%
```

```
Epoch 41: MSE=0.0081, Accuracy=95.94%
Epoch 42: MSE=0.0079, Accuracy=95.98%
Epoch 43: MSE=0.0077, Accuracy=96.14%
Epoch 44: MSE=0.0076, Accuracy=96.18%
Epoch 45: MSE=0.0074, Accuracy=96.30%
Epoch 46: MSE=0.0073, Accuracy=96.38%
Epoch 47: MSE=0.0071, Accuracy=96.42%
Epoch 48: MSE=0.0070, Accuracy=96.42%
Epoch 49: MSE=0.0068, Accuracy=96.54%
Epoch 50: MSE=0.0067, Accuracy=96.68%
Epoch 51: MSE=0.0066, Accuracy=96.74%
Epoch 52: MSE=0.0065, Accuracy=96.84%
Epoch 53: MSE=0.0064, Accuracy=96.92%
Epoch 54: MSE=0.0062, Accuracy=97.00%
Epoch 55: MSE=0.0061, Accuracy=97.06%
Epoch 56: MSE=0.0060, Accuracy=97.08%
Epoch 57: MSE=0.0059, Accuracy=97.22%
Epoch 58: MSE=0.0058, Accuracy=97.30%
Epoch 59: MSE=0.0057, Accuracy=97.34%
Epoch 60: MSE=0.0056, Accuracy=97.52%
Epoch 61: MSE=0.0055, Accuracy=97.58%
Epoch 62: MSE=0.0054, Accuracy=97.64%
Epoch 63: MSE=0.0053, Accuracy=97.66%
Epoch 64: MSE=0.0053, Accuracy=97.66%
Epoch 65: MSE=0.0052, Accuracy=97.70%
Epoch 66: MSE=0.0051, Accuracy=97.82%
Epoch 67: MSE=0.0050, Accuracy=97.84%
Epoch 68: MSE=0.0049, Accuracy=97.84%
Epoch 69: MSE=0.0049, Accuracy=97.88%
Epoch 70: MSE=0.0048, Accuracy=97.88%
Epoch 71: MSE=0.0047, Accuracy=97.94%
Epoch 72: MSE=0.0047, Accuracy=97.94%
Epoch 73: MSE=0.0046, Accuracy=97.98%
Epoch 74: MSE=0.0045, Accuracy=97.98%
Epoch 75: MSE=0.0045, Accuracy=97.98%
Epoch 76: MSE=0.0044, Accuracy=98.06%
Epoch 77: MSE=0.0043, Accuracy=98.12%
Epoch 78: MSE=0.0043, Accuracy=98.18%
Epoch 79: MSE=0.0042, Accuracy=98.20%
Epoch 80: MSE=0.0041, Accuracy=98.24%
Epoch 81: MSE=0.0041, Accuracy=98.28%
Epoch 82: MSE=0.0040, Accuracy=98.30%
Epoch 83: MSE=0.0040, Accuracy=98.30%
Epoch 84: MSE=0.0039, Accuracy=98.34%
Epoch 85: MSE=0.0039, Accuracy=98.38%
Epoch 86: MSE=0.0038, Accuracy=98.42%
Epoch 87: MSE=0.0038, Accuracy=98.44%
Epoch 88: MSE=0.0037, Accuracy=98.48%
Epoch 89: MSE=0.0037, Accuracy=98.52%
Epoch 90: MSE=0.0036, Accuracy=98.52%
Epoch 91: MSE=0.0036, Accuracy=98.52%
Epoch 92: MSE=0.0035, Accuracy=98.52%
Epoch 93: MSE=0.0035, Accuracy=98.52%
Epoch 94: MSE=0.0034, Accuracy=98.54%
```

```
Epoch 95: MSE=0.0034, Accuracy=98.54%
Epoch 96: MSE=0.0034, Accuracy=98.56%
Epoch 97: MSE=0.0033, Accuracy=98.58%
Epoch 98: MSE=0.0033, Accuracy=98.58%
Epoch 99: MSE=0.0032, Accuracy=98.60%
Epoch 100: MSE=0.0032, Accuracy=98.62%
Testing Accuracy: 87.50%
```

The model exhibits signs of overfitting, as evidenced by the high training accuracy, which suggests that it may have learned the training data too well, capturing noise or irrelevant patterns. However, despite the drop in accuracy during testing, the model still performs reasonably well on unseen data. This suggests that while overfitting may be present, the model's generalization capability is not severely compromised. Further regularization techniques or model complexity reduction may be explored to mitigate overfitting while maintaining satisfactory performance on both training and testing data