1. **Title + Name**
2. **Introduction**

We are doing classification of MNIST dataset using different classifiers like Multi-Layer Perceptron (MLP), k-nearest neighbors (knn) and a convolutional neural network with the predefined functions available in MATLAB.

3. **Methods that were implemented**
    - Multi-Layer Perceptron (MLP)
    - K-nearest neighbor (KNN)
    - Convolutional Neural Network (CNN)
4. **Presentation of the results**
    - ***Multi-Layer Perceptron (MLP)***

The complete code has been attached in Appendix A. The MNIST dataset is loaded into MATLAB using CSV files, the dataset is divided into features and labels, the input data is normalized, the labels are converted to one-hot encoding, the weights and biases are initialized, the model is trained using stochastic gradient descent with backpropagation, the model is tested on the test set, and the confusion matrix is computed and plotted. One hidden layer with 100 units makes up the neural network architecture employed in this case. The hidden layer's activation function is sigmoid, while the output layer's activation function is softmax. A sigmoid function utilized in the neural network is also defined in the code.

- ***K-nearest neighbor (KNN)***

The complete code has been attached in Appendix B. The MNIST dataset is loaded into MATLAB using CSV files, the dataset is divided into features and labels, the input data is normalised, the labels are converted to one-hot encoding, the weights and biases are initialised, the model is trained using stochastic gradient descent with backpropagation, the model is tested on the test set, and the confusion matrix is computed and plotted. One hidden layer with 100 units makes up the neural network architecture employed in this case. The hidden layer's activation function is sigmoid, while the output layer's activation function is softmax. A sigmoid function utilised in the neural network is also defined in the code.

- ***Convolutional Neural Network (CNN)***

The complete code has been attached in Appendix C. The deep learning toolbox is used in this code to create a convolutional neural network (CNN) in MATLAB. A number of convolutional layers, max pooling layers, and fully connected layers are all included in the network architecture. The handwritten digits may be classified with a high degree of accuracy thanks to the model's training on the MNIST dataset.

5. Discussion of the results

On the MNIST dataset, the k-NN classifier performs significantly better than the MLP and CNN models in terms of accuracy. This might be as a result of the non-parametric nature of the k-NN algorithm, which makes no assumptions regarding the underlying distribution of the data. As a result, it is more adaptable and can manage interactions between the characteristics and labels that are complex and non-linear. MLP and CNN, on the other hand, are parametric models that call for a pre-established architecture and presuppose specific data distributions. The models may underfit or overfit the data, which would result in subpar performance, if the assumptions are not met. Furthermore, for large datasets, k-NN can be computationally expensive and memory-intensive, whereas MLP and CNN can handle larger datasets more effectively. Globally, the

- The output accuracy and the confusion matrix by applying the MLP classifier are as follows:

>> correct_MLP

Epoch 1/50

Epoch 2/50

Epoch 3/50

Epoch 4/50

Epoch 5/50

Epoch 6/50

Epoch 7/50

Epoch 8/50

Epoch 9/50

Epoch 10/50

Epoch 11/50

Epoch 12/50

Epoch 13/50

Epoch 14/50

Epoch 15/50

Epoch 16/50

Epoch 17/50

Epoch 18/50

Epoch 19/50

Epoch 20/50

Epoch 21/50

Epoch 22/50

Epoch 23/50

Epoch 24/50

Epoch 25/50

Epoch 26/50

Epoch 27/50

Epoch 28/50

Epoch 29/50

Epoch 30/50

Epoch 31/50

Epoch 32/50

Epoch 33/50

Epoch 34/50

Epoch 35/50

Epoch 36/50

Epoch 37/50

Epoch 38/50

Epoch 39/50

Epoch 40/50

Epoch 41/50

Epoch 42/50

Epoch 43/50

Epoch 44/50

Epoch 45/50

Epoch 46/50

Epoch 47/50

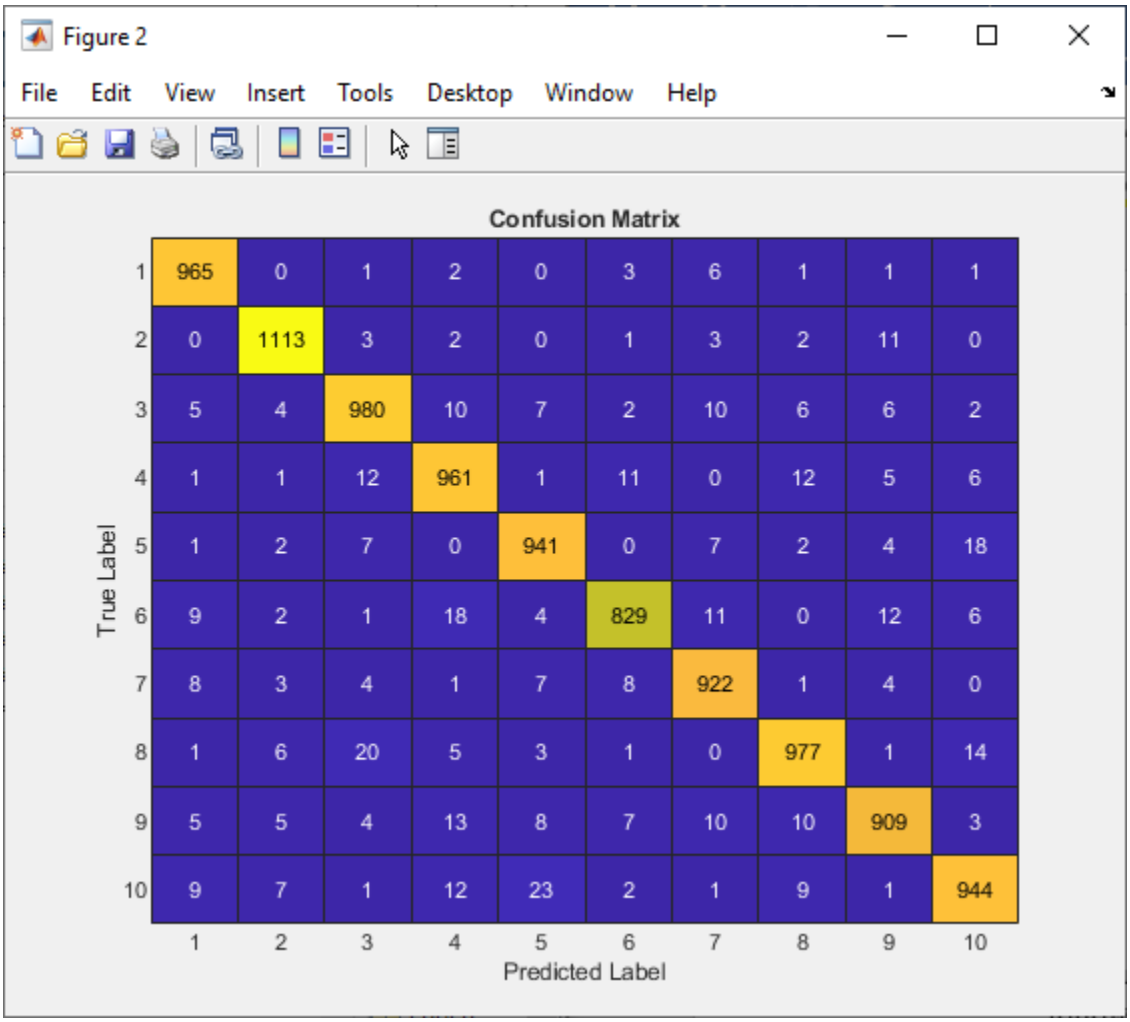
Epoch 48/50

Epoch 49/50

Epoch 50/50

Accuracy: 0.39%

The Confusion Matrix:



- The output accuracy and the confusion matrix by applying the knn classifier are as follows:

```
Confusion Matrix:
   174     0     0     0     0     0     1     0     0     0
     0   234     0     0     0     0     0     0     0     0
     5     6   194     0     0     0     3     8     3     0
     0     1     0   190     0     4     2     4     3     3
     0     3     0     0   199     0     3     0     0    12
     1     1     0     7     2   165     1     1     0     1
     4     1     0     0     2     1   170     0     0     0
     0    13     0     1     2     1     0   183     0     5
     3     3     2    10     2     6     3     3   155     5
     1     0     0     4     5     0     0     5     2   177

Accuracy: 92.05%
```

- The complete output in Matlab by applying the CNN classifier are as follows:

>> cnn

Training on single GPU.

Initializing input data normalization.

|==
====================|

| Epoch | Iteration | Time Elapsed | Mini-batch | Mini-batch | Base Learning |

| | | (hh:mm:ss) | Accuracy | Loss | Rate |

|==
====================|

| 1 | 1 | 00:00:00 | 18.75% | 2.3036 | 0.0100 |

| 1 | 50 | 00:00:00 | 80.47% | 0.6270 | 0.0100 |

| 1 | 100 | 00:00:01 | 90.62% | 0.2887 | 0.0100 |

| 1 | 150 | 00:00:01 | 92.97% | 0.2084 | 0.0100 |

| 1 | 200 | 00:00:02 | 96.88% | 0.1159 | 0.0100 |

| 1 | 250 | 00:00:02 | 95.31% | 0.2227 | 0.0100 |

| 1 | 300 | 00:00:03 | 95.31% | 0.2169 | 0.0100 |

1	350	00:00:04	92.97%	0.1942	0.0100
1	400	00:00:04	97.66%	0.0804	0.0100
1	450	00:00:05	99.22%	0.0627	0.0100
2	500	00:00:05	94.53%	0.1781	0.0100
2	550	00:00:06	97.66%	0.1064	0.0100
2	600	00:00:07	100.00%	0.0285	0.0100
2	650	00:00:07	96.88%	0.0942	0.0100
2	700	00:00:08	97.66%	0.0565	0.0100
2	750	00:00:08	99.22%	0.0332	0.0100
2	800	00:00:09	98.44%	0.0756	0.0100
2	850	00:00:10	98.44%	0.0365	0.0100
2	900	00:00:10	99.22%	0.0272	0.0100
3	950	00:00:11	97.66%	0.1159	0.0100
3	1000	00:00:12	97.66%	0.0556	0.0100
3	1050	00:00:12	98.44%	0.0458	0.0100
3	1100	00:00:13	97.66%	0.0530	0.0100
3	1150	00:00:14	98.44%	0.0329	0.0100
3	1200	00:00:14	97.66%	0.0538	0.0100
3	1250	00:00:15	98.44%	0.0416	0.0100
3	1300	00:00:15	100.00%	0.0217	0.0100
3	1350	00:00:16	100.00%	0.0204	0.0100
3	1400	00:00:17	97.66%	0.1051	0.0100
4	1450	00:00:17	99.22%	0.0246	0.0100
4	1500	00:00:18	98.44%	0.0397	0.0100
4	1550	00:00:18	96.88%	0.0896	0.0100
4	1600	00:00:19	98.44%	0.0722	0.0100
4	1650	00:00:19	99.22%	0.0215	0.0100
4	1700	00:00:20	99.22%	0.0429	0.0100

4	1750	00:00:21	96.88%	0.1012	0.0100
4	1800	00:00:21	98.44%	0.0394	0.0100
4	1850	00:00:22	97.66%	0.0473	0.0100
5	1900	00:00:23	99.22%	0.0257	0.0100
5	1950	00:00:23	99.22%	0.0616	0.0100
5	2000	00:00:24	98.44%	0.0287	0.0100
5	2050	00:00:25	99.22%	0.0319	0.0100
5	2100	00:00:25	99.22%	0.0277	0.0100
5	2150	00:00:26	99.22%	0.0171	0.0100
5	2200	00:00:26	100.00%	0.0161	0.0100
5	2250	00:00:27	99.22%	0.0262	0.0100
5	2300	00:00:27	100.00%	0.0108	0.0100
6	2350	00:00:28	97.66%	0.0471	0.0100
6	2400	00:00:29	97.66%	0.0477	0.0100
6	2450	00:00:29	100.00%	0.0138	0.0100
6	2500	00:00:30	99.22%	0.0155	0.0100
6	2550	00:00:30	100.00%	0.0099	0.0100
6	2600	00:00:31	99.22%	0.0171	0.0100
6	2650	00:00:32	98.44%	0.0354	0.0100
6	2700	00:00:32	99.22%	0.0233	0.0100
6	2750	00:00:33	99.22%	0.0177	0.0100
6	2800	00:00:33	97.66%	0.0566	0.0100
7	2850	00:00:34	99.22%	0.0113	0.0100
7	2900	00:00:34	99.22%	0.0260	0.0100
7	2950	00:00:35	97.66%	0.0455	0.0100
7	3000	00:00:36	98.44%	0.0224	0.0100
7	3050	00:00:36	98.44%	0.0552	0.0100
7	3100	00:00:37	99.22%	0.0394	0.0100

7	3150	00:00:37	98.44%	0.0258	0.0100
7	3200	00:00:38	99.22%	0.0126	0.0100
7	3250	00:00:38	100.00%	0.0106	0.0100
8	3300	00:00:39	99.22%	0.0203	0.0100
8	3350	00:00:40	100.00%	0.0065	0.0100
8	3400	00:00:40	98.44%	0.0855	0.0100
8	3450	00:00:41	99.22%	0.0152	0.0100
8	3500	00:00:41	100.00%	0.0075	0.0100
8	3550	00:00:42	99.22%	0.0276	0.0100
8	3600	00:00:42	100.00%	0.0079	0.0100
8	3650	00:00:43	99.22%	0.0193	0.0100
8	3700	00:00:44	99.22%	0.0233	0.0100
9	3750	00:00:44	100.00%	0.0077	0.0100
9	3800	00:00:45	98.44%	0.0594	0.0100
9	3850	00:00:45	96.88%	0.0548	0.0100
9	3900	00:00:46	98.44%	0.0655	0.0100
9	3950	00:00:46	99.22%	0.0240	0.0100
9	4000	00:00:47	99.22%	0.0120	0.0100
9	4050	00:00:48	100.00%	0.0015	0.0100
9	4100	00:00:48	99.22%	0.0203	0.0100
9	4150	00:00:49	100.00%	0.0078	0.0100
9	4200	00:00:49	99.22%	0.0483	0.0100
10	4250	00:00:50	100.00%	0.0056	0.0100
10	4300	00:00:51	100.00%	0.0049	0.0100
10	4350	00:00:52	100.00%	0.0115	0.0100
10	4400	00:00:52	100.00%	0.0076	0.0100
10	4450	00:00:53	99.22%	0.0302	0.0100
10	4500	00:00:54	100.00%	0.0042	0.0100

| 10 | 4550 | 00:00:54 | 100.00% | 0.0014 | 0.0100 |

| 10 | 4600 | 00:00:55 | 99.22% | 0.0688 | 0.0100 |

| 10 | 4650 | 00:00:56 | 99.22% | 0.0143 | 0.0100 |

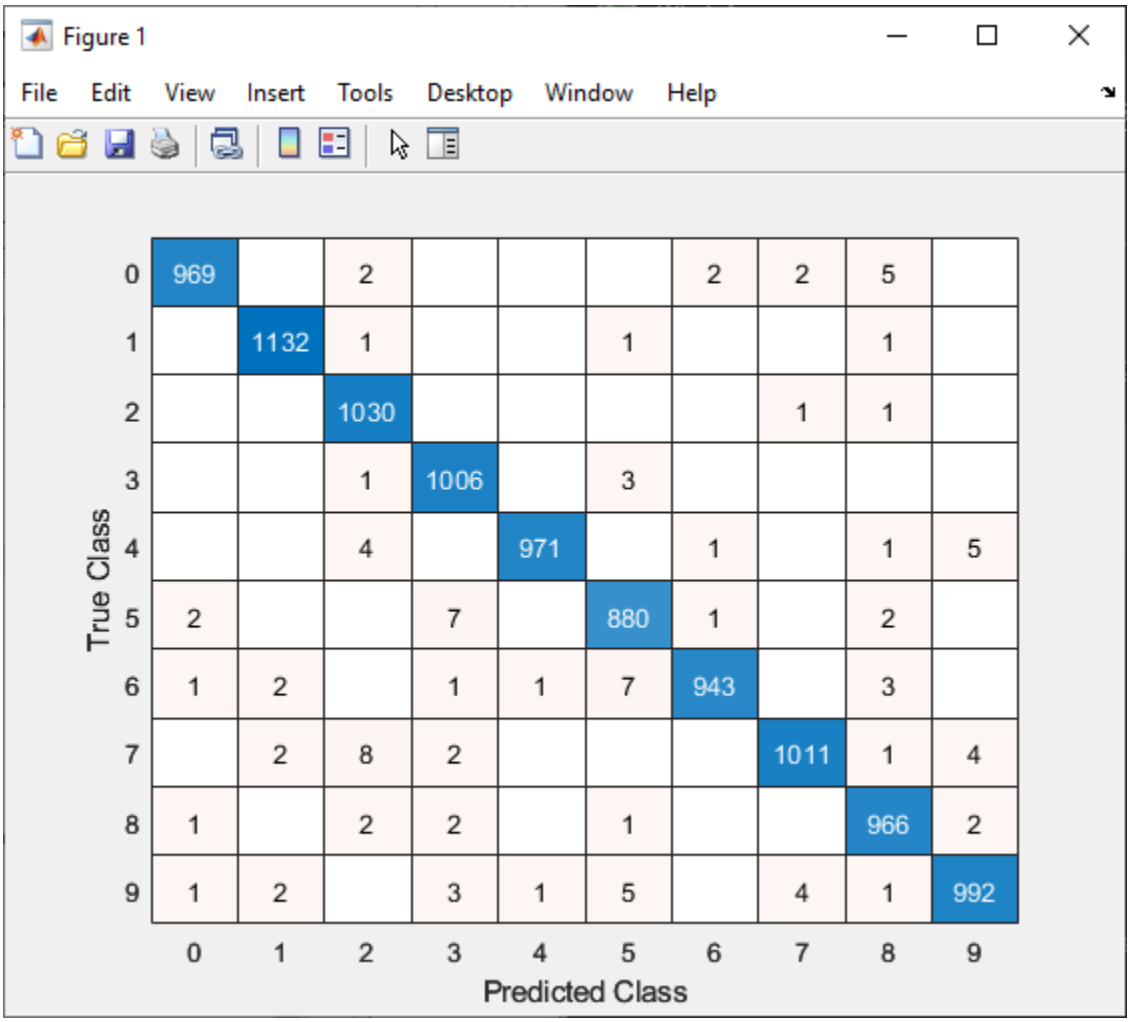
| 10 | 4680 | 00:00:56 | 99.22% | 0.0148 | 0.0100 |

|===
====================|

Training finished: Max epochs completed.

Accuracy: 98.97%

The confusion Matrix is:



Appendix A

MLP classifier

```
% Step 1: Load the dataset
train_data = readmatrix('mnist_train.csv', 'NumHeaderLines', 1);
test_data = readmatrix('mnist_test.csv', 'NumHeaderLines', 1);

% Extract features (input) and labels (output) for training and testing
X_train = train_data(:, 2:end)';
y_train = train_data(:, 1)';
X_test = test_data(:, 2:end)';
y_test = test_data(:, 1)';

% Step 2: Preprocess the data
X_train = double(X_train) / 255; % Normalize the pixel values to [0, 1]
X_test = double(X_test) / 255;

% Step 3: Define the MLP architecture
input_size = size(X_train, 1);
hidden_size = 100; % Number of neurons in the hidden layer
output_size = 10; % Number of output classes

% Initialize the weights and biases for each layer randomly
weights1 = randn(hidden_size, input_size) * 0.01;
bias1 = zeros(hidden_size, 1);
weights2 = randn(output_size, hidden_size) * 0.01;
bias2 = zeros(output_size, 1);

% Step 4: Implement the forward propagation
% Activation function: ReLU
relu = @(x) max(0, x);

% Step 5: Implement the backward propagation (backpropagation)
% Gradient descent optimizer
learning_rate = 0.01;

% Step 6: Train the MLP
num_epochs = 50; % Increased number of epochs
batch_size = 100;
num_batches = ceil(size(X_train, 2) / batch_size);

for epoch = 1:num_epochs
    % Shuffle the training data
    indices = randperm(size(X_train, 2));
    X_train = X_train(:, indices);
    y_train = y_train(:, indices);

    for batch = 1:num_batches
        % Get the current batch
        start_idx = (batch - 1) * batch_size + 1;
        end_idx = min(start_idx + batch_size - 1, size(X_train, 2));
        X_batch = X_train(:, start_idx:end_idx);
        y_batch = y_train(:, start_idx:end_idx);
```

```
        % Perform forward propagation
        Z1 = weights1 * X_batch + bias1;
        A1 = relu(Z1);
        Z2 = weights2 * A1 + bias2;
        A2 = softmax(Z2);

        % Perform backward propagation
        m = size(X_batch, 2);
        dZ2 = A2 - onehot(y_batch, output_size);
        dW2 = (1 / m) * dZ2 * A1';
        db2 = (1 / m) * sum(dZ2, 2);
        dZ1 = weights2' * dZ2 .* (Z1 > 0);
        dW1 = (1 / m) * dZ1 * X_batch';
        db1 = (1 / m) * sum(dZ1, 2);

        % Update the weights and biases
        weights2 = weights2 - learning_rate * dW2;
        bias2 = bias2 - learning_rate * db2;
        weights1 = weights1 - learning_rate * dW1;
        bias1 = bias1 - learning_rate * db1;
    end

    % Display the current epoch
    fprintf('Epoch %d/%d\n', epoch, num_epochs);
end

% Step 7: Evaluate the MLP
% Perform forward propagation on the testing dataset
Z1_test = weights1 * X_test + bias1;
A1_test = relu(Z1_test);
Z2_test = weights2 * A1_test + bias2;
A2_test = softmax(Z2_test);

% Calculate the predicted labels
[~, pred_labels] = max(A2_test);

% Calculate the accuracy
accuracy = sum(pred_labels == y_test) / length(y_test) * 100;
fprintf('Accuracy: %.2f%%\n', accuracy);

% Step 8: Calculate the confusion matrix
confusion_mat = zeros(output_size, output_size);
for i = 1:length(y_test)
    true_label = y_test(i) + 1;
    predicted_label = pred_labels(i);
    confusion_mat(true_label, predicted_label) = confusion_mat(true_label,
predicted_label) + 1;
end

% Display the confusion matrix
disp('Confusion Matrix:');
disp(confusion_mat);

% Step 9: Plot the confusion matrix
figure;
```

```
heatmap(confusion_mat, 'ColorbarVisible', 'off', 'Colormap', parula, 'FontSize', 8,
'XLabel', 'Predicted Label', 'YLabel', 'True Label', 'Title', 'Confusion Matrix');

% Helper function: Convert labels to one-hot encoding
function out = onehot(labels, num_classes)
    m = size(labels, 2);
    out = zeros(num_classes, m);
    for i = 1:m
        out(labels(i)+1, i) = 1;
    end
end
```

Appendix B

Knn classifier

```
% Load MNIST dataset from CSV files
train_data = csvread('mnist_train.csv', 1, 0);
test_data = csvread('mnist_test.csv', 1, 0);

% Use only a subset of the data to reduce memory usage
train_data = train_data(1:10000,:);
test_data = test_data(1:2000,:);

% Split dataset into features and labels
X_train = train_data(:,2:end);
y_train = train_data(:,1);
X_test = test_data(:,2:end);
y_test = test_data(:,1);

% Normalize input data
X_train = X_train ./ 255;
X_test = X_test ./ 255;

% Train and test kNN classifier
k = 5;
y_pred = knn_classifier(X_train, y_train, X_test, k);

% Compute confusion matrix and accuracy
C = confusionmat(y_test, y_pred);
accuracy = sum(diag(C)) / sum(C(:));

% Display confusion matrix and accuracy
disp('Confusion Matrix:');
disp(C);
fprintf('Accuracy: %.2f%%\n', 100*accuracy);

function y_pred = knn_classifier(X_train, y_train, X_test, k)
    % Compute pairwise distances between test and training data
    dists = pdist2(X_test, X_train);

    % Find k nearest neighbors for each test sample
    [~, indices] = sort(dists, 2);
    k_nearest = indices(:,1:k);

    % Predict class labels based on majority vote of nearest neighbors
    y_pred = mode(y_train(k_nearest), 2);
end
```

Appendix C

CNN (convolutional Neural Network)

```
% Load the MNIST dataset from CSV files
train_data = readmatrix('mnist_train.csv', 'NumHeaderLines', 1);
test_data = readmatrix('mnist_test.csv', 'NumHeaderLines', 1);

% Remove the first column (labels)
X_train = train_data(:, 2:end);
X_test = test_data(:, 2:end);

% The first column corresponds to labels
y_train = categorical(train_data(:, 1));
y_test = categorical(test_data(:, 1));

% Reshape the data to [Height Width Channels Observations]
X_train = reshape(X_train', [28, 28, 1, size(X_train, 1)]);
X_test = reshape(X_test', [28, 28, 1, size(X_test, 1)]);

% Normalize the data
X_train = X_train / 255;
X_test = X_test / 255;

% Define the LeNet-5 architecture
layers = [
    imageInputLayer([28 28 1])

    convolution2dLayer(5, 20)
    reluLayer()
    maxPooling2dLayer(2, 'Stride', 2)

    convolution2dLayer(5, 50)
    reluLayer()
    maxPooling2dLayer(2, 'Stride', 2)

    fullyConnectedLayer(500)
    reluLayer()

    fullyConnectedLayer(10)
    softmaxLayer()
    classificationLayer()
];

% Specify the training options
options = trainingOptions('sgdm', ...
    'MaxEpochs', 10, ...
    'MiniBatchSize', 128);

% Train the CNN using the specified options
net = trainNetwork(X_train, y_train, layers, options);

% Evaluate the trained CNN on the test set
y_pred = classify(net, X_test);
accuracy = sum(y_pred == y_test) / numel(y_test);
```

```
disp(['Accuracy: ', num2str(accuracy * 100), '%'])

% Plot confusion matrix
figure
confusionchart(y_test, y_pred);
```