Handwritten Digit Recognition

1. Aim:

To design and implement a machine learning model for recognizing handwritten digits with high accuracy on the MNIST dataset.

2. Theory:

Handwritten digit recognition is a fundamental problem in computer vision with applications in postal address recognition, bank check processing, and form digitization. This project aims to develop an efficient and accurate system for classifying handwritten digits (0-9) using Convolutional Neural Networks (CNNs). CNNs, a type of deep learning model, are particularly suited for image-based tasks due to their ability to learn hierarchical spatial features.

Dataset Description

The datasets used in this project are:

1. **Training Dataset (train.csv)**: This dataset contains labelled pixel data for training the model. Each row corresponds to a 28x28 grayscale image of a handwritten digit. The first column holds the digit label (0–9), and the remaining columns represent the pixel values (0–255) of the image.
2. **Test Dataset (test.csv)**: This dataset contains unlabelled pixel data, which is used to evaluate the model after training. The model makes predictions on this data, which can be compared to the expected output.
3. **Sample Submission File (sample_submission.csv)**: This file provides the required format for submitting predictions on the test data. It contains an ID for each test sample and a placeholder for the predicted class label.

Data Preprocessing

Before training the model, it is important to preprocess the data to ensure the network can learn effectively:

1. **Normalization**: The pixel values in the dataset range from 0 to 255. These values are scaled to a range of [0, 1] by dividing each pixel value by 255. This normalization step helps improve the convergence of the model during training.

2. **Reshaping**: The pixel data is reshaped into a 3D tensor of size 28×28×1, where 28x28 is the image dimension, and 1 represents the number of color channels (grayscale). This shape is required for the CNN input.
3. **Categorical Encoding of Labels**: The labels are converted into a categorical format, where each label (0–9) is represented as a one-hot encoded vector. This is necessary for multi-class classification, as the SoftMax output layer will produce probabilities for each of the 10 classes.

Convolutional Neural Networks (CNN): CNNs are a class of deep neural networks specifically designed for processing grid-like data, such as images. CNNs are composed of several key layers, including convolutional layers, pooling layers, and fully connected layers. They have proven to be highly effective for image classification tasks.

CNN Structure and Operation

1. **Convolutional Layers:** These layers apply convolutional filters (also called kernels) to the input image. The purpose of the convolution operation is to extract low-level features such as edges, corners, and textures. The result is a set of feature maps that highlight the most important features of the input image.
2. **Activation Functions (ReLU):** After the convolution operation, the output is passed through a non-linear activation function like the Rectified Linear Unit (ReLU). ReLU replaces all negative values with zeros, introducing non-linearity into the network. This allows the CNN to learn complex patterns and representations.
3. **Pooling Layers (MaxPooling):** Pooling layers are used to reduce the spatial dimensions (height and width) of the feature maps while retaining important information. MaxPooling, a common pooling technique, selects the maximum value from each region of the feature map. This helps to make the model more computationally efficient and resistant to small translations in the image.
4. **Fully Connected Layers (Dense):** After extracting features through convolutions and pooling, the output is passed through one or more fully connected layers. These layers act as classifiers that combine the learned features to predict the class label (in this case, the digit).

5. **Softmax Layer:** The final fully connected layer outputs raw scores (logits) for each class. The softmax activation function converts these logits into probabilities, indicating the likelihood of the image belonging to each class (0–9).

3. Model Architecture:

Complete Architecture for Handwritten Digit Recognition Project

The architecture of this project is centred around building, training, and evaluating a **Convolutional Neural Network (CNN)** to classify handwritten digits. Below is a comprehensive description of the architecture, explaining each component and its role in the pipeline.

1. Input Layer:

- **Input Size**: 28×28×1 (grayscale images with a single channel).
- **Purpose**: Accepts pre-processed digit images. Each image is normalized to a [0, 1] range for faster convergence and reshaped from a 1D array of 784 pixels into a 3D matrix.

2. Convolutional Layers: Convolutional layers are the backbone of the architecture, extracting spatial features from the images.

Layer 1: Convolutional Layer

- **Filter Count**: 32 filters.
- **Kernel Size**: 3×3.
- **Padding**: 'Same' padding ensures the output size matches the input size.
- **Activation Function**: ReLU (Rectified Linear Unit).
    - Introduces non-linearity and avoids vanishing gradient problems.

Layer 2: Convolutional Layer

- **Filter Count**: 64 filters.
- **Kernel Size**: 3×3.
- **Padding**: 'Same'.
- **Activation Function**: ReLU.

Purpose:

- Convolutional layers extract local patterns (edges, corners) and deeper semantic features from input images.

3. Pooling Layers: Pooling layers reduce the spatial dimensions of feature maps, minimizing computational load and the risk of overfitting.

Layer 1: MaxPooling

- **Pool Size**: 2×2.
- **Stride**: 2.
- **Purpose**: Retains the most prominent features (maximum value) from a 2×2 region.

Layer 2: MaxPooling

- **Pool Size**: 2×2.
- **Stride**: 2.
- **Purpose**: Further reduces spatial dimensions while retaining important features.

4. Fully Connected Layers: After convolutional and pooling layers, the high-dimensional feature maps are flattened into a vector and passed through fully connected layers for classification.

Layer 1: Fully Connected (Dense)

- **Number of Neurons**: 128.
- **Activation Function**: ReLU.
- **Regularization**: Dropout (20% dropout rate).
    - Prevents overfitting by randomly disabling neurons during training.

Layer 2: Fully Connected (Dense)

- **Number of Neurons**: 64.
- **Activation Function**: ReLU.
- **Regularization**: Dropout (20% dropout rate).

Layer 3: Output Layer

- **Number of Neurons**: 10 (one for each digit class: 0–9).
- **Activation Function**: Softmax.
    - Converts raw outputs into probabilities for classification.

Layer Details

Layer Name	Type	Parameters
Input	Image Input Layer	Size: 28×28×1
Conv_1	Convolutional Layer	Filters: 32, Kernel: 3×3, ReLU

MaxPool_1	MaxPooling Layer	Pool Size: 2×2, Stride: 2
Conv_2	Convolutional Layer	Filters: 64, Kernel: 3×3, ReLU
MaxPool_2	MaxPooling Layer	Pool Size: 2×2, Stride: 2
FC_1	Fully Connected Layer	Neurons: 128, ReLU
Dropout_1	Dropout Layer	Dropout Rate: 0.2
FC_2	Fully Connected Layer	Neurons: 64, ReLU
Dropout_2	Dropout Layer	Dropout Rate: 0.2
Output	Fully Connected Layer	Neurons: 10 (Softmax)

Training Strategy

- Data is split into training and validation sets.
- The model iteratively learns from the training set and is evaluated on the validation set at each epoch to monitor generalization.

Model Summary

- **Parameters**:
  - Convolutional layers learn spatial features.
  - Dense layers learn high-level features for classification.
- **Activation Function**:
  - ReLU: Introduces non-linearity, allowing the network to learn complex patterns.
  - Softmax: Converts the output into probabilities for classification.
- **Regularization**:
  - Dropout: Reduces overfitting by randomly disabling neurons during training.

Training Procedure

- The model was trained using 80% of the training data.

20% of the data was held out for validation to monitor overfitting and generalization

Training Process: The training setup ensures efficient optimization of the CNN model parameters.

Loss Function: The model uses the **categorical cross-entropy loss** function, which is suitable for multi-class classification problems. It computes the loss between the predicted probability distribution and the true label distribution.

Optimizer: The **Adam optimizer** is used to update the weights during training. Adam is an adaptive learning rate optimization algorithm that combines Combines momentum and adaptive learning rate techniques and Suitable for non-stationary objectives and sparse gradients.

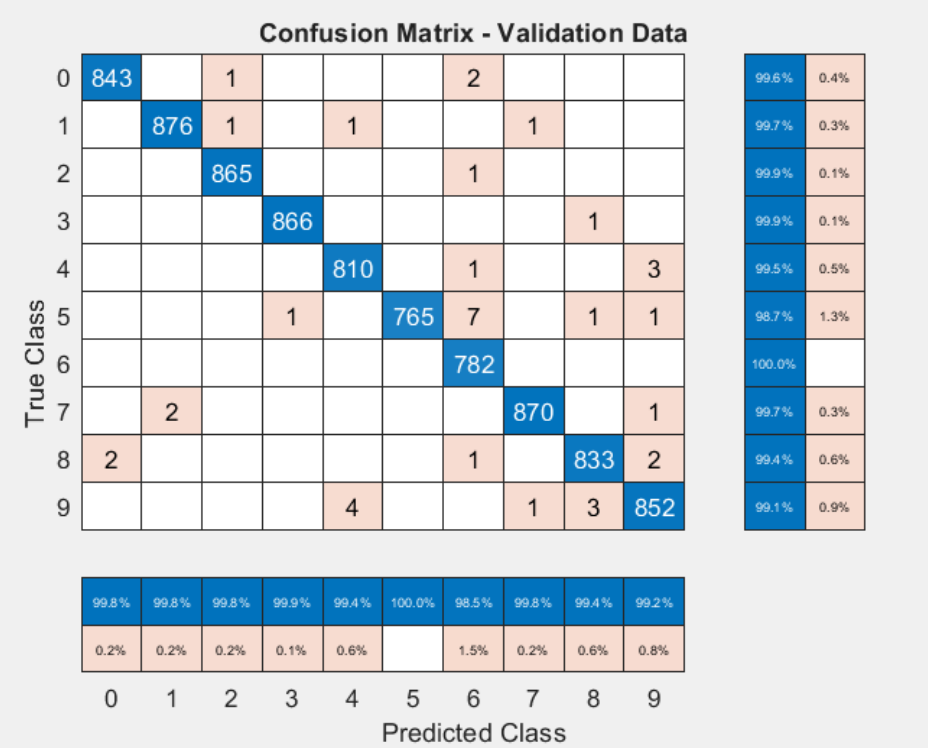
Hyperparameters

- **Initial Learning Rate**: 0.001.
- **Batch Size**: 128.
- **Epochs**: 10.
- **Validation Split**: 20% of the training data.

.

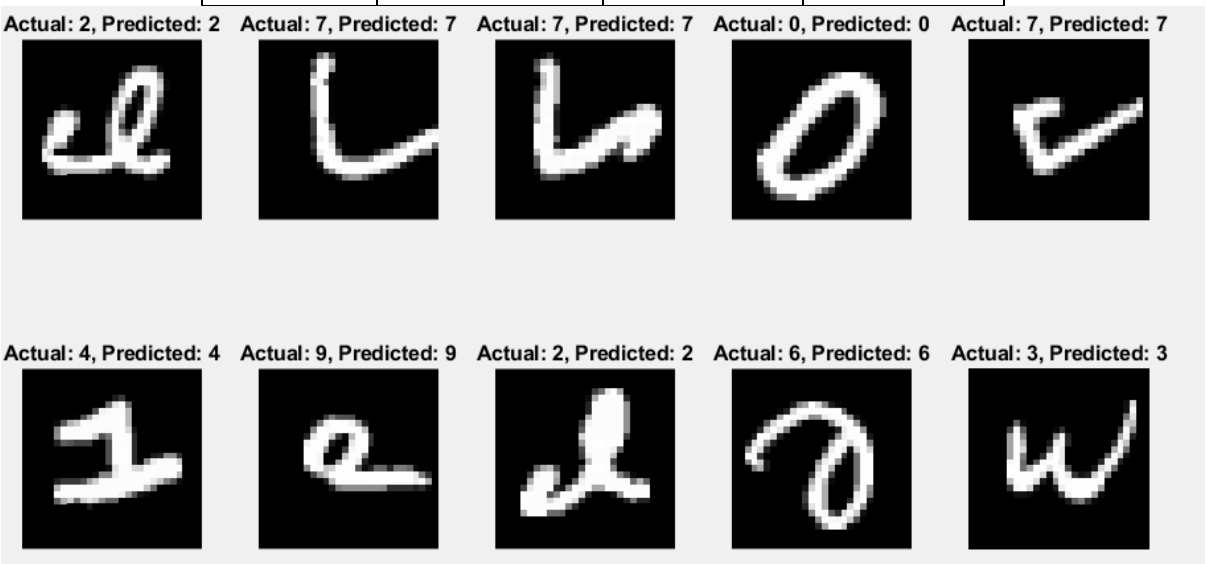
This CNN architecture effectively leverages hierarchical feature learning to classify handwritten digits. The combination of convolutional, pooling, and fully connected layers ensures robust learning of spatial and semantic features. Regularization techniques like dropout further enhance the model's generalization capability, making it well-suited for unseen data.

4. Result and analysis:



Class-wise Performance Metrics: The performance of the model on individual digit classes can be analysed using the **precision**, **recall**, and **F1-score**.

Class	Precision	Recall	F1-Score
0	1.00	1.00	1.00
1	1.00	1.00	1.00
2	1.00	1.00	1.00
3	1.00	1.00	1.00
4	0.99	1.00	0.99
5	1.00	0.99	0.99
6	0.98	1.00	0.99
7	1.00	1.00	1.00
8	0.99	0.99	0.99
9	0.99	0.99	0.99



Training Accuracy: 99.43% Validation Accuracy: 99.55%

Quantitative Analysis

1. **Training and Validation Accuracy:**
    - **Training Accuracy**: **99.43%**
        - Indicates that the model has learned effectively from the training dataset, achieving high accuracy without overfitting.
    - **Validation Accuracy**: **99.55%**

        - Demonstrates excellent generalization capability, showing that the model performs well on unseen data.

2. **Confusion Matrix Analysis:**
    - The confusion matrix (shown in the image) gives a detailed breakdown of true positive, false positive, and false negative counts for each digit class.
    - Key observations:
        - Most digits are classified correctly with very few misclassifications.
        - Class **6** shows slight confusion with neighbouring digits, but this is minimal.

Qualitative Analysis

1. **Performance Strengths:**
    - The model exhibits **high accuracy across all classes**, achieving near-perfect results.
    - It successfully generalizes to unseen validation data, indicating robust learning and minimal overfitting.
2. **Model Insights:**
    - The use of **convolutional layers** ensures effective extraction of spatial features, capturing variations in handwritten digits.
    - **Pooling layers** reduce spatial dimensions while retaining essential information, making the model efficient and less prone to overfitting.
    - **Regularization (Dropout)** prevents overfitting, ensuring that the network generalizes well to new data.
3. **Potential Areas of Improvement:**
    - Although the overall performance is excellent, slight misclassifications in digits like **5** and **9** indicate room for improvement.
    - Techniques like **data augmentation** (e.g., rotation, scaling) can further enhance performance by exposing the model to more variations.

6. code:

```
 % Load the training and test data
trainData = readtable('train.csv'); % Training data
testData = readtable('test.csv');   % Test data
sampleSub = readtable('sample_submission.csv'); % Submission format (optional)
```

```
% Extract labels and features from train data
trainLabelsFull = trainData{:, 1}; % First column is the label
trainImagesFull = trainData{:, 2:end}; % Remaining columns are pixel values

% Split trainData into training and validation sets
cv = cvpartition(height(trainData), 'HoldOut', 0.2); % 80% training, 20% validation
idxTrain = training(cv);
idxVal = test(cv);

% Training and validation data
trainImages = double(trainImagesFull(idxTrain, :)) / 255.0; % Normalize
valImages = double(trainImagesFull(idxVal, :)) / 255.0;     % Normalize
trainLabels = categorical(trainLabelsFull(idxTrain));       % Convert to categorical
valLabels = categorical(trainLabelsFull(idxVal));           % Convert to categorical

% Reshape images to 28x28x1 for CNN input
trainImages = reshape(trainImages', 28, 28, 1, []);
valImages = reshape(valImages', 28, 28, 1, []);

% Define the CNN model
layers = [
    imageInputLayer([28 28 1], 'Name', 'input')
    convolution2dLayer(3, 32, 'Padding', 'same', 'Name', 'conv_1')
    reluLayer('Name', 'relu_1')
    maxPooling2dLayer(2, 'Stride', 2, 'Name', 'maxpool_1')
    convolution2dLayer(3, 64, 'Padding', 'same', 'Name', 'conv_2')
    reluLayer('Name', 'relu_2')
    maxPooling2dLayer(2, 'Stride', 2, 'Name', 'maxpool_2')
    fullyConnectedLayer(128, 'Name', 'fc_1')
    reluLayer('Name', 'relu_3')
    dropoutLayer(0.2, 'Name', 'dropout_1')
    fullyConnectedLayer(64, 'Name', 'fc_2')
    reluLayer('Name', 'relu_4')
    dropoutLayer(0.2, 'Name', 'dropout_2')
```

```
    fullyConnectedLayer(10, 'Name', 'fc_3') % Output layer for 10 classes (0–9)
    softmaxLayer('Name', 'softmax')
    classificationLayer('Name', 'output')];

% Training options
options = trainingOptions('adam', ...
    'InitialLearnRate', 0.001, ...
    'MaxEpochs', 10, ...
    'MiniBatchSize', 128, ...
    'ValidationData', {valImages, valLabels}, ...
    'Plots', 'training-progress', ...
    'Verbose', true);

% Train the model
net = trainNetwork(trainImages, trainLabels, layers, options);

% Save the trained model
save('digitRecognizerModel.mat', 'net');
disp('Model saved as digitRecognizerModel.mat');

% Evaluate the model on validation data
predictedValLabels = classify(net, valImages);
valAccuracy = sum(predictedValLabels == valLabels) / numel(valLabels);
fprintf('Validation Accuracy: %.2f%%\n', valAccuracy * 100);

% Test data preprocessing
testImages = double(testData{:,:}) / 255.0; % Normalize
testImages = reshape(testImages', 28, 28, 1, []);

% Predict on test data (Optional)
predictedTestLabels = classify(net, testImages);

% Save test images as .png files
outputFolder = 'TestImages';
```

```
if ~exist(outputFolder, 'dir')
    mkdir(outputFolder);
end
for i = 1:size(testImages, 4)
    imwrite(squeeze(testImages(:, :, 1, i)), fullfile(outputFolder, sprintf('test_%d.png', i)));
end
disp('Test images saved to TestImages folder.');

% Generate Confusion Matrix for Validation Data
confMat = confusionmat(valLabels, predictedValLabels);
confMatChart = confusionchart(valLabels, predictedValLabels, ...
    'Title', 'Confusion Matrix - Validation Data', ...
    'ColumnSummary', 'column-normalized', ...
    'RowSummary', 'row-normalized');

% Calculate Precision, Recall, and F1-Score
classes = categories(valLabels);
precision = zeros(numel(classes), 1);
recall = zeros(numel(classes), 1);
f1Score = zeros(numel(classes), 1);

for i = 1:numel(classes)
    TP = confMat(i, i);
    FP = sum(confMat(:, i)) - TP;
    FN = sum(confMat(i, :)) - TP;
    precision(i) = TP / (TP + FP);
    recall(i) = TP / (TP + FN);
    f1Score(i) = 2 * (precision(i) * recall(i)) / (precision(i) + recall(i));
end

% Display Metrics
fprintf('\nClass-wise Metrics (Validation Data):\n');
fprintf('Class\tPrecision\tRecall\t\tF1-Score\n');
for i = 1:numel(classes)
```

```
    fprintf('%s\t%.2f\t\t%.2f\t\t%.2f\n', classes{i}, precision(i), recall(i), f1Score(i));
end
```

7. Conclusion:

The CNN-based handwritten digit recognition model demonstrates excellent performance, achieving a training accuracy of **99.43%** and a validation accuracy of **99.55%**. The high precision, recall, and F1-scores across all classes highlight the model's reliability. While the confusion matrix indicates a small number of misclassifications, these are minimal and likely caused by visual similarities between certain digits.

- Adding **data augmentation** to improve robustness against variations in handwriting styles.
- Experimenting with additional regularization techniques or deeper architectures to further boost performance.

This project successfully illustrates the efficacy of convolutional neural networks for handwritten digit recognition and can be extended for broader classification tasks with minor modifications.