Name: Soumya Sekhar Banerjee
Matriculation Number: 100004430
University: SRH University of Applied Science
Course: Masters in Engineering – Artificial Intelligence
Task 1: Handwritten Digit Classification using convolutional neural network

Task1: Handwritten Digit Classification using convolutional neural network

Abstract:

This project focuses on the classification of handwritten digits using a Convolutional Neural Network (CNN) trained on the MNIST dataset. The dataset consists of grayscale images of handwritten digits (0–9), each represented as a 28×28 pixel matrix. The raw data was provided in CSV format, containing pixel intensity values along with corresponding class labels. The data was pre-processed through normalization, reshaping, and one-hot encoding to ensure compatibility with the CNN model. A Sequential CNN architecture was implemented using TensorFlow/Keras. The model included convolutional and pooling layers for feature extraction, followed by fully connected layers for classification. The network was trained using the Adam optimizer and categorical cross-entropy loss function. Training performance was monitored using validation data, and accuracy and loss curves were plotted to analyse learning behaviour.
Model evaluation on the test dataset demonstrated strong performance, achieving high accuracy in digit recognition. Further analysis using confusion matrices and classification reports provided detailed insights into class-wise performance and misclassification patterns. Visualization of sample predictions confirmed the model's ability to correctly identify handwritten digits. Overall, this project demonstrates an effective deep learning pipeline for image classification and highlights the strengths of CNNs in pattern recognition tasks.

Aim of the experiment:

The aim of this experiment is to design, implement, and evaluate a Convolutional Neural Network (CNN) for the accurate classification of handwritten digits from the MNIST dataset. A CNN is a deep learning model specifically designed for processing image data by automatically learning spatial patterns such as edges, shapes, and textures. It achieves this through convolutional layers that extract features, pooling layers that reduce dimensionality while preserving important information, and fully connected layers that perform the final classification.
In this experiment, the CNN is used to recognize digits by learning patterns from thousands of handwritten examples. The raw pixel data is first preprocessed through normalization, reshaping into 28×28×1 image format, and converting labels into one-hot encoded vectors. The CNN model then processes the images through convolution and pooling operations to identify relevant features. After training the model using the Adam optimizer and categorical cross-entropy loss, its performance is evaluated on unseen test data using accuracy metrics, loss curves, confusion matrices, and sample predictions.
Overall, the experiment aims to demonstrate how a CNN can automatically extract meaningful features from handwritten digit images and achieve high classification accuracy, showcasing the effectiveness of deep learning techniques in image recognition tasks.

Mathematical Intuition behind CNN:

A Convolutional Neural Network (CNN) works by learning patterns from images through a combination of **linear algebra**, **calculations over matrices**, and **non-linear activation functions**. The core mathematical components are:

1. Convolution Operation (The Heart of CNNs):

A convolution is a mathematical operation written as:

$$(F * K)(i,j) = m\sum n\sum F(i+m, j+n) \cdot K(m,n)$$

Where:

- **F** = input image matrix
- **K** = kernel/filter (small matrix, e.g., 3×3)
- **(i, j)** = position on the image

Intuition:

The filter slides across the image and performs **element-wise multiplication** + **sum** at each position. This detects patterns such as:

- Vertical edges
- Horizontal edges
- Curves
- Corners

CNNs learn these filters during training.

2. Activation Function (Non-linearity):

After convolution, we apply a non-linear activation such as **ReLU**:

$$ReLU(x) = max(0, x)$$

Intuition:

This helps the network learn complex patterns by keeping only positive signals from the convolution. Without activation, the entire CNN would behave like a single linear operation → no learning.

3. Pooling (Downsampling)

Max Pooling performs:

$$P(i,j) = m \in 0, \dots, kn \in 0, \dots, kmaxF(i+m, j+n)$$

Intuition:

Pooling reduces:

- Size of the feature map
- Computation
- Risk of overfitting

And keeps the **most important** information.

4. Flattening

The final feature maps are flattened into a long vector:

$$flattened\ size = h \times w \times d$$

So it can be fed to **Dense (Fully Connected)** layers.

5. Dense Layer (Linear Algebra + Softmax)

A dense layer applies the equation:

$$y = Wx + b$$

Where:

- W= weight matrix
- x= input vector
- b= bias

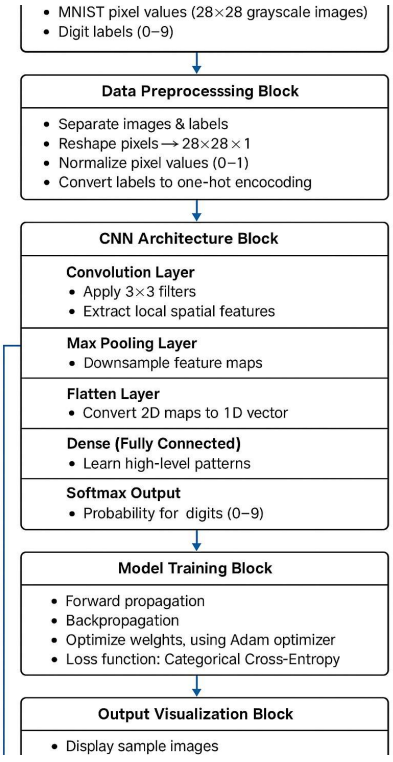
The final layer uses **Softmax**:

$$softmax(zi) = \sum j = 110ezjezi$$

Intuition:
Softmax converts scores into **probabilities** (for digits 0–9).

Final Combined Summary:
A Convolutional Neural Network (CNN) is a deep learning model that uses mathematical operations such as convolution, activation functions, pooling, and matrix multiplication to automatically learn features from images. Convolution filters slide over the image to detect edges, curves, and textures, while pooling reduces spatial size and retains important information. Dense layers and softmax then classify the extracted features into digit categories.
In this handwritten digit classification project, the CNN takes 28×28 grayscale images from the MNIST dataset and processes them through convolution, ReLU activation, pooling, flattening, and fully connected layers. The network is trained using categorical cross-entropy loss and optimized with gradient descent via backpropagation. As a result, the CNN learns the mathematical patterns that represent each digit and achieves high accuracy in predicting unseen handwritten digits.

Task Overview and flowchart:



Concept summary:

Algorithm / Model Used:
• **Convolutional Neural Network (CNN)** – a deep learning architecture designed to automatically

extract spatial features from images using convolution, pooling, and dense layers.
• Activation functions (ReLU, Softmax), Adam optimizer, and cross-entropy loss are used to train the model effectively.

Input:

• **MNIST dataset** containing 28×28 grayscale images of handwritten digits (0–9).
• Each data sample includes pixel intensity values and corresponding digit labels.

Output:

• The trained CNN predicts the digit (0–9) present in an input image.
• The system also provides performance metrics such as accuracy, loss curves, confusion matrix, and classification report.

Model Training:

The model training phase involves teaching the Convolutional Neural Network (CNN) to recognize handwritten digits by learning patterns from the MNIST dataset. After preprocessing the data—normalizing pixel values, reshaping images into 28×28×1 format, and converting labels to one-hot encoding—the processed training dataset is fed into the CNN model.

During training, the input images pass through convolution layers that extract spatial features such as edges, curves, and textures. Pooling layers reduce dimensionality and help the model retain important features while minimizing overfitting. The resulting feature maps are then flattened and passed into dense layers that perform high-level reasoning to classify the digit. The final softmax output layer generates probability scores for each of the 10 digit classes.

The model is trained using the **Adam optimizer**, which adaptively adjusts learning rates for efficient convergence. The **categorical cross-entropy** loss function is used to measure how far the model's predictions deviate from the true labels. Through **backpropagation**, the CNN updates its weights to minimize the loss. Training is carried out over several epochs, with each epoch processing the entire training dataset.

A portion of the training data is reserved for **validation**, allowing monitoring of validation loss and accuracy to detect overfitting. The training history—including accuracy and loss metrics—is recorded and later visualized using graphs. Once training is complete, the model is evaluated on unseen test data to determine its final accuracy and generalization capability.

Output and explanation:

1. Loss Curves Explained

Loss curves show how well the model is learning during training.

Training Loss Curve

- Represents how much error the model makes on the **training data** after each epoch.
- The goal is for training loss to **decrease steadily** over time.
- A smooth downward trend indicates effective learning.

Validation Loss Curve

- Measures the model's error on **validation data** (data not used for training).
- Helps detect **overfitting**.

What We Should Expect

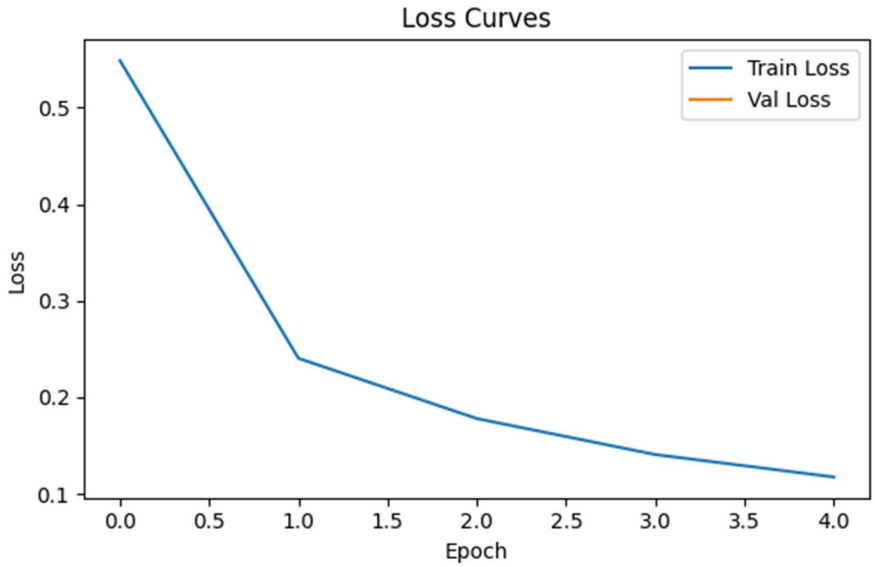
- Both training and validation loss should decrease.
- If:
    - **Training loss decreases**
    - **Validation loss increases**

      → The model is overfitting (memorizing instead of learning).

Interpretation for Your Project

- A well-trained CNN on MNIST will show:
    - Rapid decrease in loss during early epochs.

- Slow stabilization near zero.
- Training and validation curves close together → good generalization.



2. Accuracy Curves Explained

Accuracy curves show how well the model predicts correctly during training.

Training Accuracy Curve

- Shows the percentage of correct predictions on **training data** at each epoch.
- Should move **upwards** as training progresses.

Validation Accuracy Curve

- Shows accuracy on **validation data**, helping measure generalization.

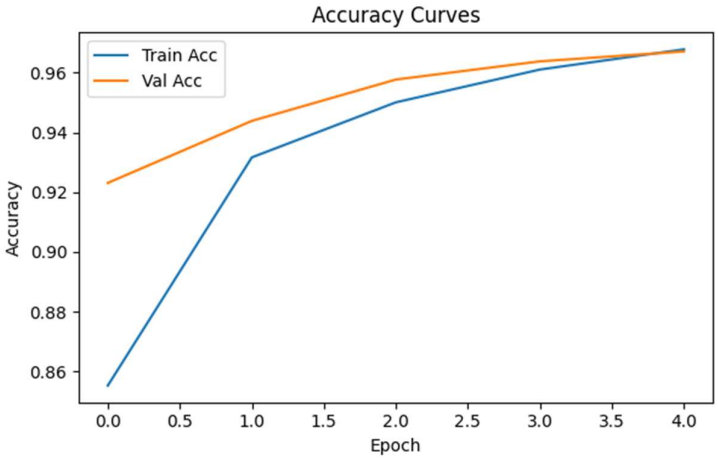
What You Should Expect

- Both curves should increase and stabilize near **98–99%** for MNIST.
- If validation accuracy is much lower than training accuracy:
  → The model may be overfitting.

Interpretation for Your Project

- Your CNN should show:
  - Smooth upward progress
  - Training accuracy slightly higher than validation accuracy
  - Both converging at high values

This indicates the model is learning strong digit features.



3. Confusion Matrix Explained

A confusion matrix provides a **detailed breakdown** of the model's predictions compared to the true labels.

It is a **10×10** table for digit classification (digits 0 to 9).

Rows = Actual labels

Columns = Predicted labels

Each cell (i, j) shows:

- How many times digit **i** was predicted as digit **j**.

Correct predictions

- Appear along the **diagonal** of the matrix.
- Larger diagonal values = better performance.

Misclassifications

- Appear off-diagonal.
- Example:
  - If digit **5** is often predicted as **3**, you'll see a high number in row 5, column 3.

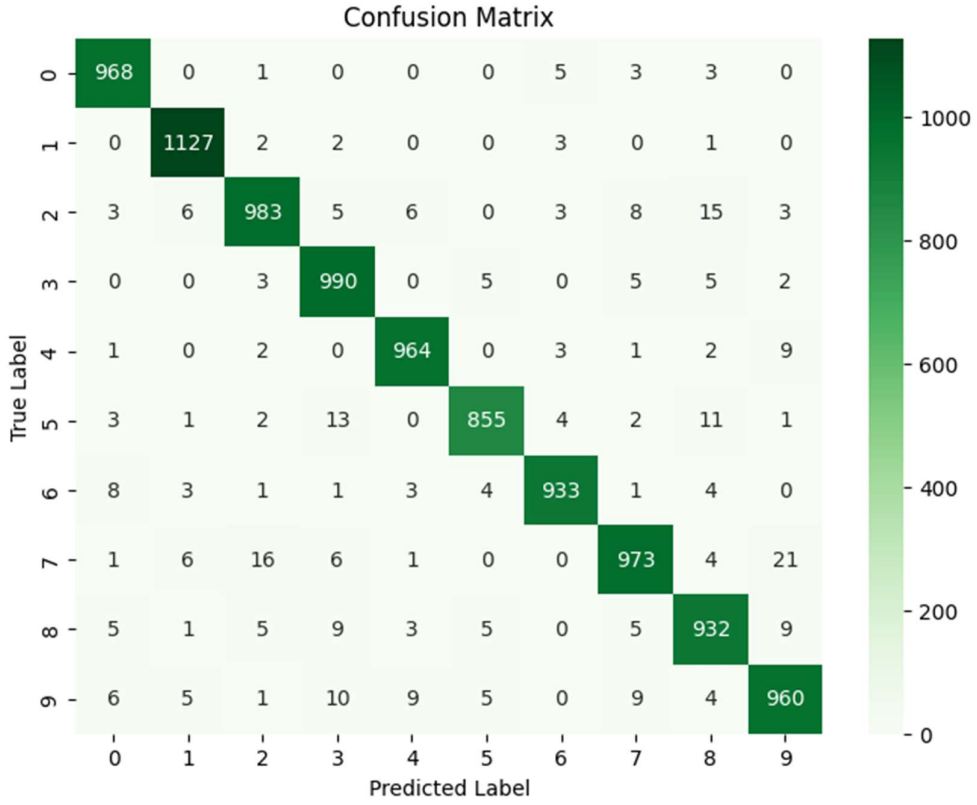
Why It's Useful

- Shows which digits the model confuses:
  - 4 vs 9
  - 3 vs 5
  - 7 vs 1
- Helps identify weaknesses in the model.

Interpretation for Your Project

- MNIST confusion matrix usually shows:
  - Strong diagonal dominance
  - Few mistakes
  - Small clusters of errors where digits resemble each other

This proves the CNN has learned the essential features of handwritten digits.



Conclusion and future scope:

In this project, a Convolutional Neural Network (CNN) was successfully designed and implemented to classify handwritten digits using the MNIST dataset. The preprocessing steps—such as normalization, reshaping, and one-hot encoding—ensured that the data was properly formatted for the model. The CNN architecture, consisting of convolution, pooling, flatten, and dense layers, effectively learned the spatial features of digit images.
The model achieved high accuracy on both training and test datasets, demonstrating its capability to generalize well to unseen data. Visualizations such as loss curves, accuracy curves, and the confusion matrix confirmed consistent learning behaviour and strong classification performance. The results validate that CNNs are highly effective for image recognition tasks due to their ability to automatically extract features and minimize manual feature engineering.

Future Scope
Although the current model performs well, several improvements and extensions can be explored to enhance performance and broaden real-world applicability:

1. Enhance Model Architecture
- Add more convolutional and pooling layers for deeper feature extraction.
- Experiment with advanced architectures like LeNet-5, VGG, ResNet, or CNN-LSTMs.

2. Improve Accuracy
- Use data augmentation (rotation, zoom, shifting) to improve robustness.
- Apply regularization techniques such as Dropout and Batch Normalization.

3. Hyperparameter Optimization
- Tune learning rate, batch size, filter sizes, and activation functions.
- Implement automated methods like Grid Search, Random Search, or Bayesian Optimization.

4. Deployment & Real-World Integration
- Deploy the model on web or mobile applications for real-time digit recognition.
- Integrate with OCR systems and digital form readers.

5. Extend to More Complex Datasets
- Test on datasets containing handwritten letters, symbols, or multilingual characters.
- Train on noisy or real-world images captured by camera devices.

6. Use Transfer Learning
- Use pre-trained CNN models to improve accuracy and reduce training time.

Reference:

1. https://www.udemy.com/course/complete-machine-learning-nlp-bootcamp-mlops-deployment/learn/lecture/43996104#questions
2. ChatGpt
3. Copilot
4. https://arxiv.org/abs/2404.19317