Date of Submission July 3, 2025.

# Image Classification using Convolutional Neural Networks

**AMRIT KANDEL[1], (THA078BEI004), PRASISH TIMALSINA[2], (THA078BEI026)**
[1]Department of Electronics and Computer Engineering, Thapathali Campus (e-mail: amrit.tha078bei004@tcioe.edu.np)
[2]Department of Electronics and Computer Engineering, Thapathali Campus (e-mail: prasish.tha078bei026@tcioe.edu.np)

**ABSTRACT** This project about the implementation of a Convolutional Neural Network (CNN) based on the LeNet architecture to recognize handwritten digits in the MNIST dataset. The MNIST dataset consists of grayscale images of digits (0 to 9) and is widely used to evaluate image classification models. The network architecture consists of convolutional, pooling, and fully connected layers, optimized by the Adam optimizer and trained using sparse categorical crossentropy. The model achieved improved accuracy over standard ANN. It was evaluated against performance metrics such as accuracy, precision, recall, confusion matrix, and training curves.

**INDEX TERMS** Convolutional Neural Network (CNN), Image Classification, MNIST Dataset, Performance Metrics, Confusion Matrix, Precision, Recall,

## I. INTRODUCTION

### A. CONVOLUTIONAL NEURAL NETWORKS (CNN)

Convolutional Neural Network (CNN) is a more advanced version of artificial neural networks (ANNs), designed primarily for extracting features from grid-structured matrix data. This is extremely relevant to visual data such as images or videos, where patterns within the data are crucial. CNNs find wide usage in computer vision applications because they are effective in processing visual data. CNN consists of multiple layers like input layer, Convolutional layer, pooling layer, and fully connected layers.

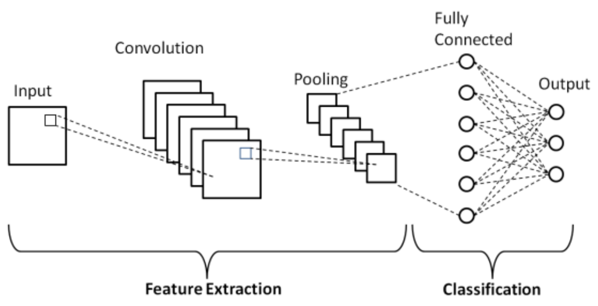


**FIGURE 1. Block Diagram of Convolutional Neural Network**

Different layers of a CNN are discussed below.

#### 1) Input Layer

The input layer is the start of a CNN, and the image is passed into the network. The image would often be a grid of numbers, and these are the pixel values. An RGB image would have three channels—red, green, and blue—so the input will be height, width, and depth (3 for RGB). The input layer does not perform any computation; it only holds the raw data to be calculated by the following layers.

#### 2) Convolutional Layer

The convolutional layer is the core of a CNN. It utilizes small filters (also known as kernels) that slide across the input image in an attempt to discover patterns. Each filter is like a window that scans a small patch of the image at once. Since the filter translates, it scales its values against the pixel values underneath and adds them together to create a new value. In this way, features like edges, lines, or shapes are learned by the network. The output of this layer is called a feature map, which is where particular patterns are found in the image.

#### 3) Activation Layer

After the convolutional layer, the CNN generally adds a ReLU layer, or Rectified Linear Unit. The reason for this layer is to allow the network to learn complex patterns by introducing non-linearity. In simple terms, it replaces all the negative values inside the feature map with zeros but does not change positive values. This speeds up the model and allows it to learn better as data in real-world data tends to be disorganized and non-linear.

#### 4) Pooling Layer

The pooling layer comes after the activation layer and helps reduce the size of the feature maps without the most important

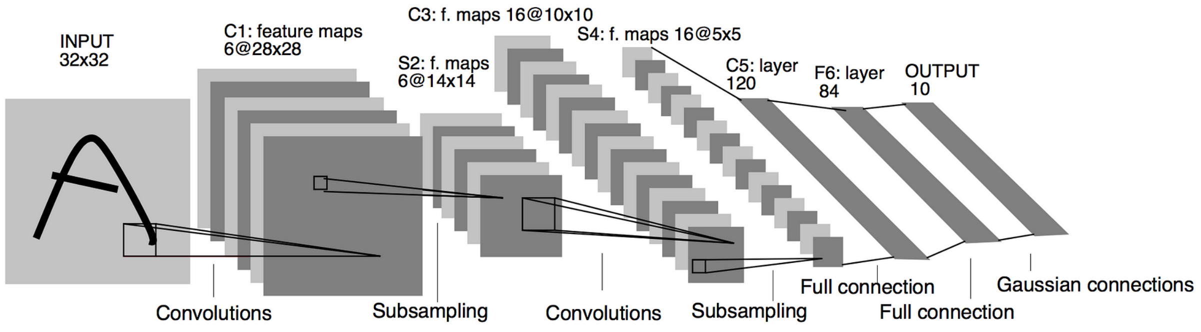


**FIGURE 2.** **LENET Architecture**

information. The aim is to accelerate the computations and prevent overfitting. The most common type is max pooling, where the largest value in a small area of the feature map is chosen. As an example, if the area is 2×2, only the maximum value is maintained. This enables the network to focus on the most dominant features and ignore insignificant changes like noise.

#### 5) Fully Connected Layer

There is one or more fully connected layers at the end of the CNN. These function essentially like a traditional neural network, with every node connected to every other node in the previous layer. They gather all the features learned by the previous layers and utilize them in order to generate the final prediction. For example, if the network is trying to recognize objects, the fully connected layer will help decide whether the image is of a cat, dog, car, or other object.

### B. LENET ARCHITECTURE

LeNet-5 is one of the earliest convolutional neural networks, developed by Yann LeCun in the 1990s to recognize handwritten digits. It is a simple yet powerful architecture that processes grayscale images (usually 32×32 pixels) and classifies them into 10 categories (digits 0–9).

The architecture consists of 7 layers (excluding input), including convolutional layers, pooling layers, and fully connected layers:

- **Input Layer** – Takes a 32 × 32 grayscale image.
- **Convolutional Layer 1 (C1)** – Applies 6 filters of size 5 × 5 with stride 1 and no padding. Output size: 28 × 28 × 6.
- **Pooling Layer 1 (S2)** – Applies average pooling (subsampling) with a 2 × 2 filter. Output size: 14 × 14 × 6.
- **Convolutional Layer 2 (C3)** – Applies 16 filters of size 5 × 5. Output size: 10 × 10 × 16.
- **Pooling Layer 2 (S4)** – Applies average pooling again with a 2 × 2 filter. Output size: 5 × 5 × 16.
- **Fully Connected Layer (C5)** – A fully connected layer with 120 neurons. The previous feature maps (from S4) are flattened and connected.
- **Fully Connected Layer (F6)** – A dense layer with 84 neurons. Prepares the features for the final classification.
- **Output Layer** – A fully connected softmax layer with 10 neurons, one for each digit class (0–9).

## II. DATASET DESCRIPTION

We used MNIST (Modified National Institute of Standards and Technology) Handwritten Digit dataset. It is a standard in the machine learning and computer vision community. It includes 70,000 grayscale images of handwritten digits of size 28x28 pixels. It is divided into 60,000 training images and 10,000 test images, with each image labeled with the appropriate digit it represents, from 0 through 9. MNIST has become a standard for evaluating classification algorithms due to its simplicity, cleanliness, and balanced classes. While fairly simple, it is nevertheless a basic dataset for evaluating new image classification models and techniques.

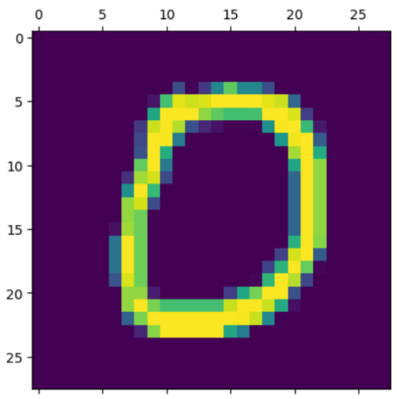
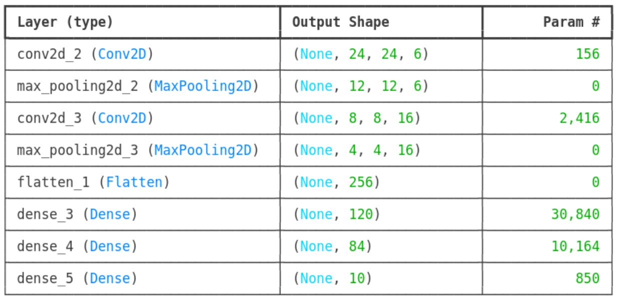


**FIGURE 3.** **MNIST dataset sample**

## III. EXPERIMENTATION MODEL

The experimental model used in this project is a Convolutional Neural Network (CNN) based on LeNet-5 architecture used on TensorFlow and Keras. The model was learned on classifying images on the MNIST dataset, which contains 28×28 gray-scale digit images of handwritten digits. In order to match the pre-shape of the anticipated input for the CNN, the images were reshaped to (28, 28, 1) and normalized by dividing pixel values by 255. The model architecture consists of two convolutional layers with 6 and 16 filters (5×5 kernel size), followed by max pooling layers for the reduction in dimensionality. The output from the convolutional layers was flattened and input into two fully connected dense layers of 120 and 84 neurons, followed by a final dense layer of 10 neurons with softmax activation to perform multi-class classification. The model was trained with the Adam optimizer and sparse categorical crossentropy as the loss function. It was then trained on the training set with 64 as the batch size for 10 epochs, and the test set was utilized as validation during training.

The model is optimized using the Adam optimizer. The loss function used is sparse_categorical_crossentropy, appropriate when the target labels are integers (not one-hot encoded). The summary of the model is given in Figure 4.

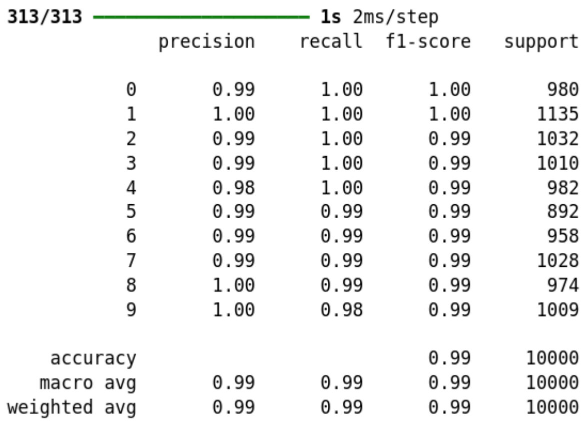


| Layer (type) | Output Shape | Param # |
|---|---|---|
| conv2d_2 (Conv2D) | (None, 24, 24, 6) | 156 |
| max_pooling2d_2 (MaxPooling2D) | (None, 12, 12, 6) | 0 |
| conv2d_3 (Conv2D) | (None, 8, 8, 16) | 2,416 |
| max_pooling2d_3 (MaxPooling2D) | (None, 4, 4, 16) | 0 |
| flatten_1 (Flatten) | (None, 256) | 0 |
| dense_3 (Dense) | (None, 120) | 30,840 |
| dense_4 (Dense) | (None, 84) | 10,164 |
| dense_5 (Dense) | (None, 10) | 850 |

**FIGURE 4. Model Summary**

## IV. RESULTS

In lab, we used different performance metrics like Accuracy, precision and recall and observed the result. In experiment, we were able to successfully obtain the correct label for given handwritten digit from the test set. When the model was trained for 10 epochs and batch size was taken 64, we obtained the accuracy of 0.9889 and 0.0350. The performance of the model can be summarized by using various performance metrics as in the figure 5.



```
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step
              precision    recall  f1-score   support

           0       0.99      1.00      1.00       980
           1       1.00      1.00      1.00      1135
           2       0.99      1.00      0.99      1032
           3       0.99      1.00      0.99      1010
           4       0.98      1.00      0.99       982
           5       0.99      0.99      0.99       892
           6       0.99      0.99      0.99       958
           7       0.99      0.99      0.99      1028
           8       1.00      0.99      0.99       974
           9       1.00      0.98      0.99      1009

    accuracy                           0.99     10000
   macro avg       0.99      0.99      0.99     10000
weighted avg       0.99      0.99      0.99     10000
```

**FIGURE 5. Result Summary**

The confusion matrix is given in figure 6.

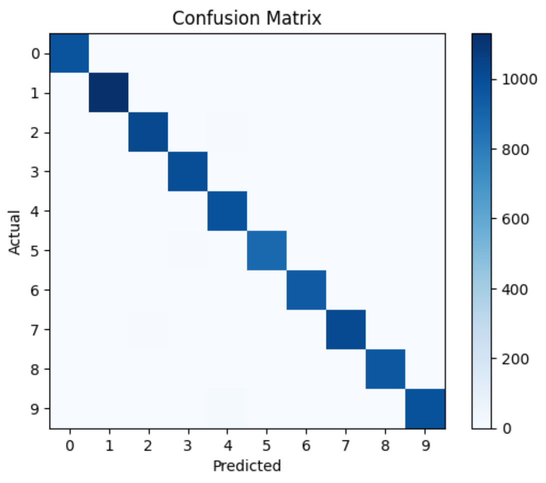


**FIGURE 6. Confusion Matrix**

The accuracy and loss curve are as given in figure 7.

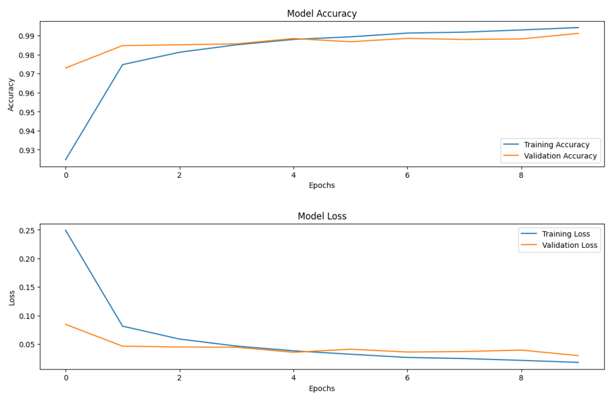


**FIGURE 7. Training curves**

## V. DISCUSSION

We used a CNN model based on the LeNet architecture to recognize handwritten digits from the MNIST dataset. The model for 10 epochs with a batch size of 64 and we were able to achieve a very good accuracy of 98.89% on the test set, which shows that the model learned and classified the digits well. The loss value was also very small, near 0.0350, which shows that the model did not make many mistakes. Training and validation accuracy improved progressively over time, which meant that the model was training appropriately. We could observe from the confusion matrix that most of the digits were correctly predicted, although there were a few small errors between digits. Precision, recall, and F1-score were also employed to check the performance of the model. This shows that our model worked well and the experiment was successful.

## VI. CONCLUSION

In this lab, we successfully built and train a Convolutional Neural Network model to recognize handwritten digits in the MNIST dataset. The model worked very well with high accuracy and low error on the test set. We learned from this lab working of CNN and how different layers like convolution, pooling, and fully connected layers work together for the classification of images.

## APPENDIX A

### *SOURCE CODE*

```
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Flatten
    , Conv2D, MaxPooling2D
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.datasets import mnist

model = Sequential([
    Conv2D(filters = 6, kernel_size = (5,5),
        strides = (1,1), padding='valid',
        activation='relu'),
    MaxPooling2D(pool_size = (2,2)),
    Conv2D(filters = 16, kernel_size = (5,5),
        strides = (1,1), padding = 'valid',
        activation='relu'),
    MaxPooling2D(pool_size = (2,2)),
    Flatten(),
    Dense(120, activation='relu'),
    Dense(84, activation='relu'),
    Dense(10, activation='softmax')
]
)

model.compile(
    optimizer = 'adam',
    loss = 'sparse_categorical_crossentropy',
    metrics = ['accuracy']
)

(X_train, y_train), (X_test, y_test) = mnist.
    load_data()
```

```
X_train.shape
y_train.shape

X_test.shape
y_test.shape

plt.matshow(X_train[0])

X_train.shape


X_train = X_train/255
X_test = X_test / 255
print(X_train.shape)

X_train_flattened = X_train.reshape(-1, 28, 28, 1)
X_test_flattened = X_test.reshape(-1, 28, 28, 1)

history = model.fit(x=X_train_flattened, y=y_train
    , batch_size=64, epochs=10, validation_data=(
    X_test_flattened, y_test))

model.evaluate(X_test_flattened, y_test)

model.summary()

# plt.plot(history.history['val_accuracy'])
# plt.plot(history.history['accuracy'])
# plt.plot(history.history['loss'])
# plt.plot(history.history['val_loss'])

plt.figure(figsize=(12, 8))

# Subplot for accuracy
plt.subplot(2, 1, 1)
plt.plot(history.history['accuracy'], label='
    Training Accuracy')
plt.plot(history.history['val_accuracy'], label='
    Validation Accuracy')
plt.title('Model Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.tight_layout(pad=3.0)  # Add padding between
    plots

# Subplot for loss
plt.subplot(2, 1, 2)
plt.plot(history.history['loss'], label='Training
    Loss')
plt.plot(history.history['val_loss'], label='
    Validation Loss')
plt.title('Model Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.tight_layout(pad=3.0)  # Ensure padding is
    applied


y_predicted = model.predict(X_test_flattened)

# Convert predicted probabilities to class labels
y_pred = np.argmax(y_predicted, axis=1)

## To get performance metrics
from sklearn.metrics import classification_report
print(classification_report(y_test, y_pred))

y_predicted = model.predict(X_test_flattened)

np.round(y_predicted[100],3)

# To get confusion matrix
```

```
from sklearn.metrics import confusion_matrix
import matplotlib.pyplot as plt
confusion_matrix = confusion_matrix(y_test, y_pred
    )
print(confusion_matrix)
plt.imshow(confusion_matrix, cmap='Blues',
    interpolation='nearest')
plt.colorbar()
plt.xticks(np.arange(10), np.arange(10))
plt.yticks(np.arange(10), np.arange(10))
plt.grid(False)
plt.xticks(np.arange(10), np.arange(10))
plt.yticks(np.arange(10), np.arange(10))
plt.title('Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
```