**Practical 9**

**Implementation of convolutional neural network to predict numbers from number images.**

**9**

**Aim:** Implementation of convolutional neural network to predict numbers from number images

**Description:**

Learns Features: It uses convolutional layers to automatically learn features from images, like edges, shapes, and textures.

Stacked Layers: These convolutional layers are stacked together, allowing the network to learn increasingly complex features.

Classification or Detection: Finally, the network uses fully-connected layers to classify images (e.g., identifying digits) or detect objects within them.

Grid-like Approach: CNNs process images in a grid-like fashion using filters (kernels) that slide across the image. This helps capture spatial information and identify features regardless of their position.

Parameter Efficiency: Compared to fully-connected neural networks, CNNs have fewer parameters due to shared weights within filters. This reduces training complexity and helps prevent overfitting.

Wide Applications: Beyond image recognition, CNNs are used for tasks like video analysis, natural language processing (analyzing text structure), and even medical image analysis.

**Code:**

from keras.datasets import mnist

from keras.utils import to\_categorical

from keras.models import Sequential

from keras.layers import Dense, Conv2D, Flatten

import matplotlib.pyplot as plt

# Download MNIST data and split into train and test sets

(X\_train, Y\_train), (X\_test, Y\_test) = mnist.load\_data()

# Plot the first image in the dataset

plt.imshow(X\_train[0])

plt.show()

print(X\_train[0].shape)

# Reshape data for CNN (add channel dimension)

X\_train = X\_train.reshape(60000, 28, 28, 1)

X\_test = X\_test.reshape(10000, 28, 28, 1)

# One-hot encode labels

Y\_train = to\_categorical(Y\_train)

Y\_test = to\_categorical(Y\_test)

# Print an example of one-hot encoded label

print(Y\_train[0])

# Define the model architecture

model = Sequential()

# Learn image features with convolutional layers

model.add(Conv2D(64, kernel\_size=3, activation='relu', input\_shape=(28, 28, 1)))

model.add(Conv2D(32, kernel\_size=3, activation='relu'))

model.add(Flatten())

# Add a dense layer with softmax activation for 10-class classification

model.add(Dense(10, activation='softmax'))

# Compile the model for training

model.compile(optimizer='adam', loss='categorical\_crossentropy', metrics=['accuracy'])

# Train the model with validation data

model.fit(X\_train, Y\_train, validation\_data=(X\_test, Y\_test), epochs=3)

# Make predictions on the first 4 test images

predictions = model.predict(X\_test[:4])

print(predictions) # Predicted probabilities for each class

# Print the actual labels for the first 4 test images

print(Y\_test[:4]) # One-hot encoded labels

**Output:**

**A screenshot of a computer

Description automatically generated**

**A screenshot of a computer code

Description automatically generated**

**Learning:**

* Imports: Grabs libraries for data, model building, and visualization.
* Data Prep: Loads MNIST dataset, reshapes for CNNs, and one-hot encodes labels.
* CNN Model: Builds a sequential model with convolutional layers for feature extraction and a dense layer for classification.
* Compilation: Sets up the model for training with optimizer, loss function, and accuracy metric.
* Training: Trains the model on data with validation for performance monitoring.