HAND WRITTEN DIGIT RECOGINITION WITH CNN

BY:
DAYANA DEVI K

NAME : DAYANA DEVI K

DEPARTMENT : B TECH INFORMATION TECHNOLOGY

COLLEGE NAME : MEENAKSHI SUNDARARAJAN ENGINEERING COLLEGE

GMAIL ID : dayanadevikuppan4@gmail.com

NM ID: 23C5C111FF87AFF81544AEE85B078B29

Zone III : Chennai-III

AGENDA

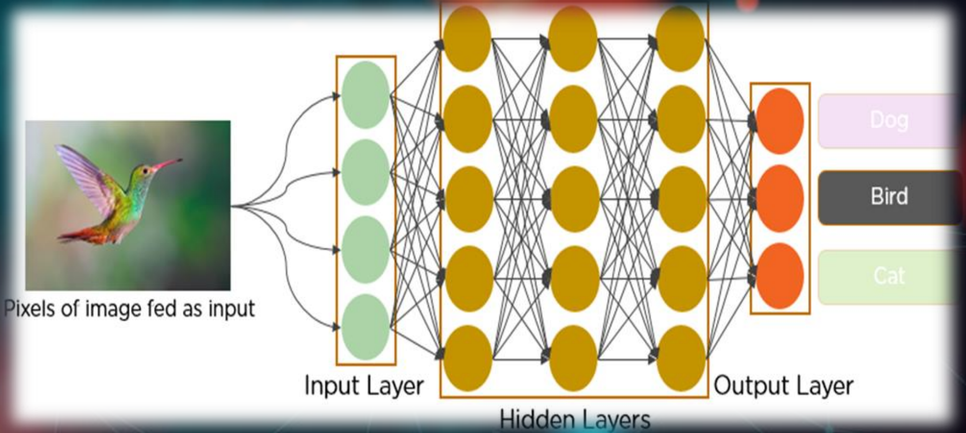
1. PROBLEM STATEMENT
2. CNN AND ITS LAYERS (algorithm)
3. PROJECT OVERVIEW
4. WHO ARE THE END USERS
5. SOLUTION AND ITS VALUE PROPOSITION
6. THE WOW IN MY SOLUTION
7. MODELLING
8. CODE IMPLEMENTATION
9. RESULTS

PROBLEM STATEMENT

- To Develop a deep learning model to accurately classify handwritten digits from the MNIST dataset using convolutional neural networks (CNNs). The model should be trained to recognize digits ranging from 0 to 9 and achieve high accuracy in classifying unseen handwritten digits.
- In today's world, there's a big need for computers to understand handwritten numbers. Think about when we write a check or fill out a form – wouldn't it be helpful if a computer could easily read what we wrote?
- The goal is to make this system really good at recognizing all sorts of handwriting, so it can be useful in things like sorting mail, processing checks, and filling out forms automatically.

Convolutional Neural Network (Algorithm used)

- CNN is a deep learning technique to classify the input automatically (well, after you provide the right data).
- Over the years, CNN has found a good grip over classifying images for computer visions and now it is being used in healthcare domains too.
- This indicates that CNN is a reliable deep learning algorithm for an automated end-to-end prediction.
- CNN essentially extracts ‘useful’ features from the given input automatically making it super easy for us!



THREE LAYERS OF CNN

Convolutional Layer

- This layer extracts high-level input features from input data and passes those features to the next layer in the form of feature maps.

Pooling Layer

- It is used to reduce the dimensions of data by applying pooling on the feature map to generate new feature maps with reduced dimensions. PL takes either maximum or average in the old feature map within a given stride.

Fully-Connected Layer

- Finally, the task of classification is done by the FC layer. Probability scores are calculated for each class label by a popular activation function called the softmax function.

PROJECT OVERVIEW

1.DATA PREPARATION

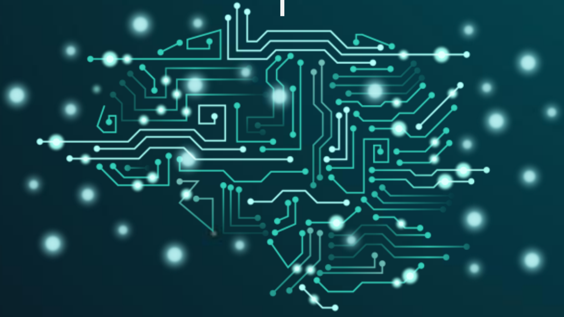
2.MODEL ARCHITECTURE

3.MODEL COMPILATION

4.TRAIN THE MODEL

5.EVALUATE THE MODEL

6.MAKE PREDICTIONS



WHO ARE END USERS ?

END USERS

INDIVIDUALS

needing to input handwritten digits into digital systems.

INSTITUTIONS

for grading and analyzing student work.

BUISNESS

across various industries for automating data entry.

GOVERNMENT

agencies for processing forms and documents.

SOLUTION AND ITS VALUE PROPOSITION

The system accurately identifies handwritten digits, providing reliable results for various applications.

Accuracy

01

It streamlines digit recognition processes, saving time and effort compared to manual entry or processing methods.

Efficiency

02

By automating the recognition process, it reduces the need for manual intervention, improving workflow efficiency.

Automation

03

YOUR SOLUTION AND ITS VALUE PROPOSITION (CONTD)

The system can handle large volumes of handwritten digits, making it suitable for both small-scale and large-scale applications.

Scalability

04

It can be integrated into diverse systems and applications across different industries, offering flexibility and adaptability.

Versatility

05

By reducing the need for manual labor and minimizing errors, it contributes to cost savings over time.

Cost-effectiveness

06

THE WOW IN MY SOLUTION

WOW FACTORS	
40%	100%
●●○○○	●●●●●
INITIAL MODEL TRAINING	**PROVIDE VALUES**
Impress with early results of your CNN model trained on a subset of the MNIST dataset, demonstrating its ability to learn and make predictions even in the early stages	Its user-friendly interface, scalability, and cost-effectiveness, our solution delivers unparalleled value by streamlining processes and optimizing workflow.

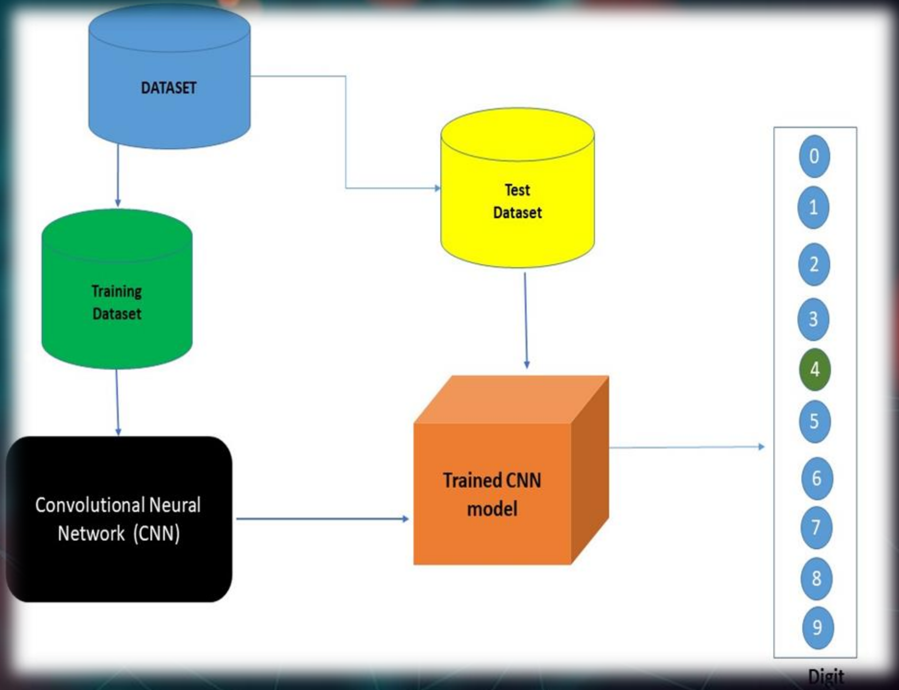
MODELLING

1 .Data Preparation:

- The MNIST dataset consists of grayscale images of handwritten digits from 0 to 9.
- Each image is 28 Load the MNIST dataset, which consists of grayscale images of handwritten digits and their corresponding labels.
- Normalize the pixel values to the range [0, 1]. Reshape the input data to have a single channel. x28 pixels.

2. Model Architecture :

- We design a CNN architecture for the digit recognition task.
- This typically involves stacking convolutional layers followed by pooling layers to extract relevant features from the images, flattening the output, and then passing it through fully connected layers for classification

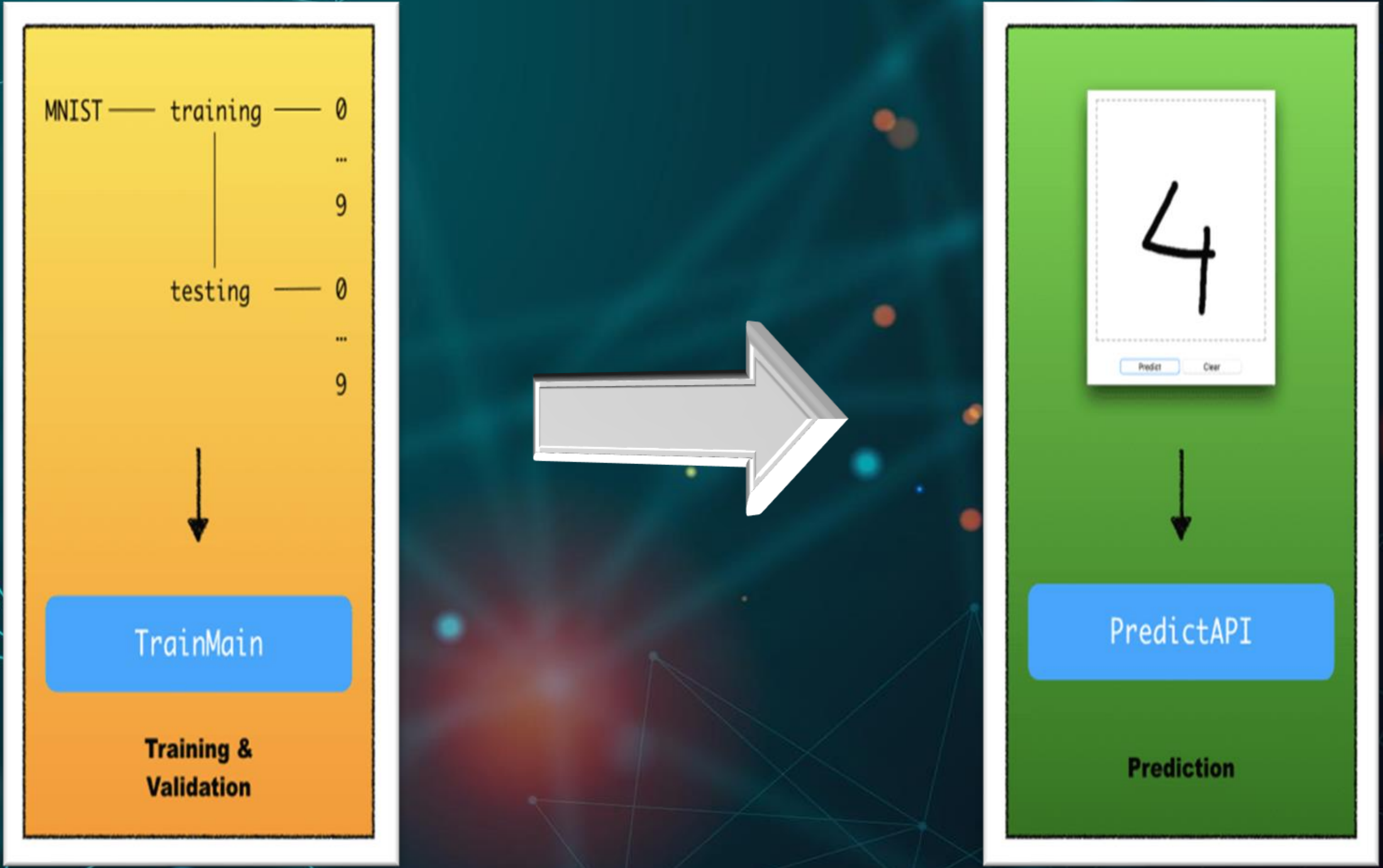


3. Compile Model: Once the architecture is defined, we compile the model with appropriate loss function (usually categorical cross-entropy for multi-class classification), optimizer (commonly Adam), and evaluation metric (accuracy).

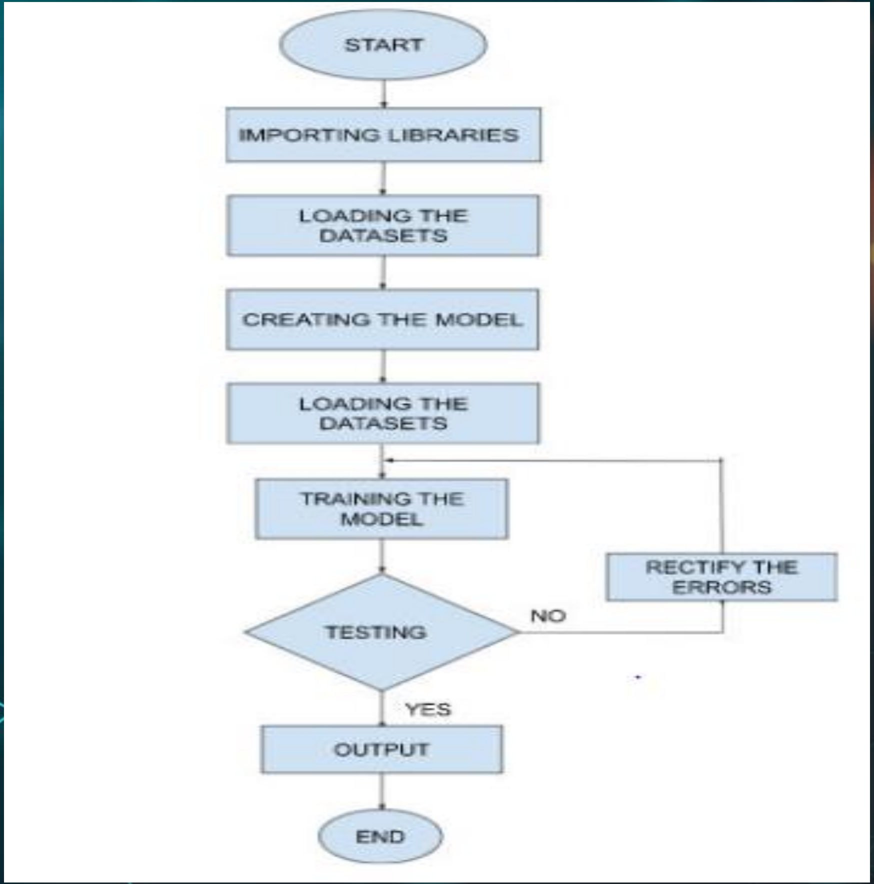
4. Training: We train the model on the training data, adjusting the weights using backpropagation and optimization algorithms to minimize the loss function. This step involves iterating through the training data for multiple epochs.

5. Evaluation: After training, we evaluate the model's performance on a separate test dataset to assess its accuracy and generalization ability.

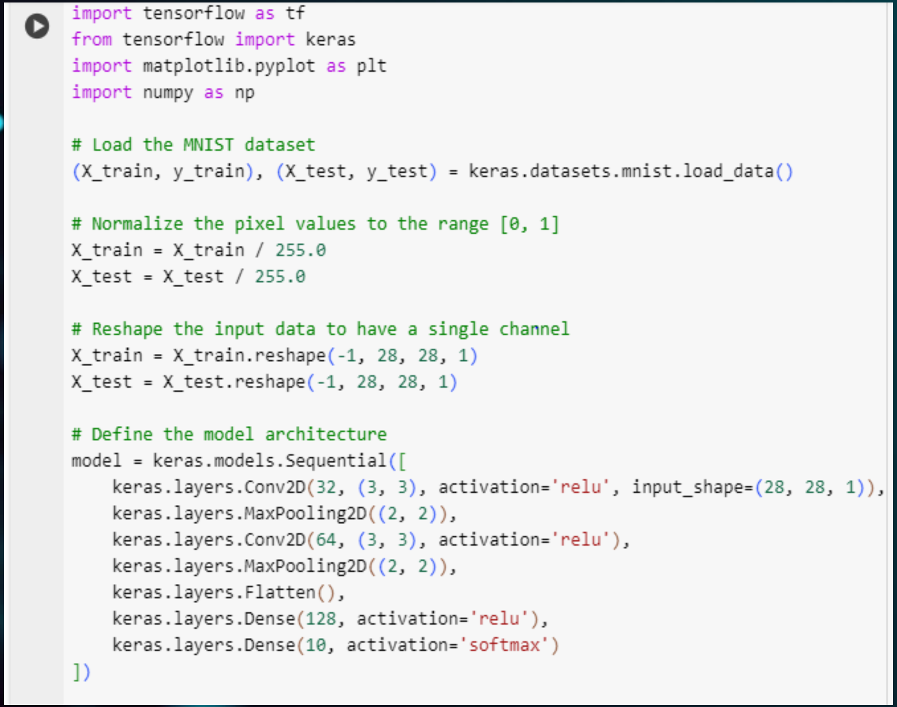
6. Prediction: Finally, we can use the trained model to make predictions on new, unseen handwritten digit images. The model predicts the digit represented by the input image.


MNIST
training
0
…
9
testing
0
…
9
TrainMain
Training &
Validation
Predict
Clear
PredictAPI
Prediction

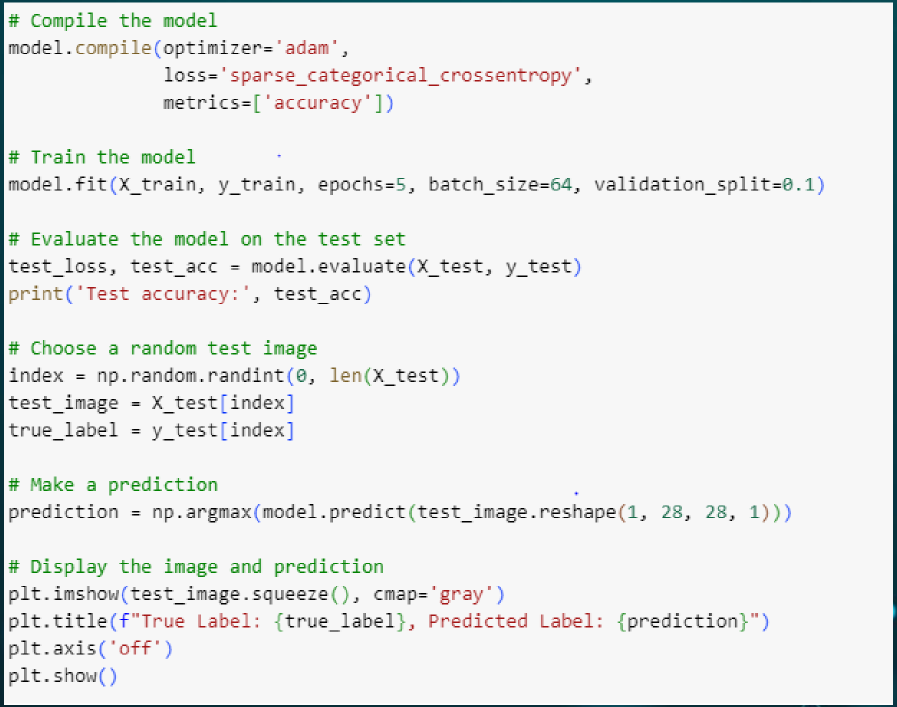
FLOW CHART


START
IMPORTING LIBRARIES
LOADING THE DATASETS
CREATING THE MODEL
LOADING THE DATASETS
TRAINING THE MODEL
TESTING
NO
RECTIFY THE ERRORS
YES
OUTPUT
END

CODE IMPLEMENTATION

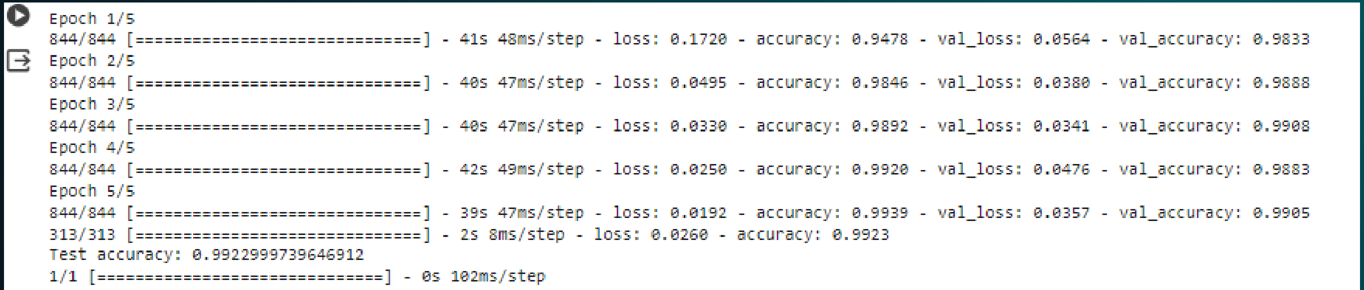


```python
import tensorflow as tf
from tensorflow import keras
import matplotlib.pyplot as plt
import numpy as np

# Load the MNIST dataset
(X_train, y_train), (X_test, y_test) = keras.datasets.mnist.load_data()

# Normalize the pixel values to the range [0, 1]
X_train = X_train / 255.0
X_test = X_test / 255.0

# Reshape the input data to have a single channel
X_train = X_train.reshape(-1, 28, 28, 1)
X_test = X_test.reshape(-1, 28, 28, 1)

# Define the model architecture
model = keras.models.Sequential([
    keras.layers.Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)),
    keras.layers.MaxPooling2D((2, 2)),
    keras.layers.Conv2D(64, (3, 3), activation='relu'),
    keras.layers.MaxPooling2D((2, 2)),
    keras.layers.Flatten(),
    keras.layers.Dense(128, activation='relu'),
    keras.layers.Dense(10, activation='softmax')
])
```



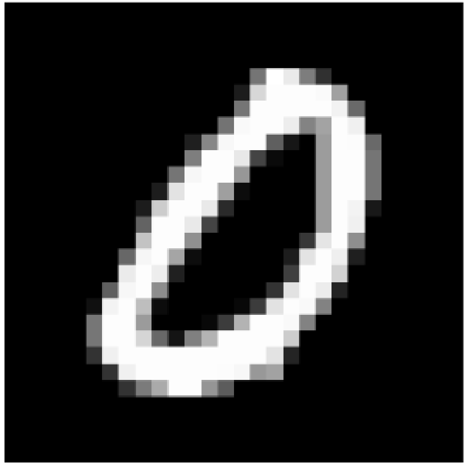
```python
# Compile the model
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

# Train the model
model.fit(X_train, y_train, epochs=5, batch_size=64, validation_split=0.1)

# Evaluate the model on the test set
test_loss, test_acc = model.evaluate(X_test, y_test)
print('Test accuracy:', test_acc)

# Choose a random test image
index = np.random.randint(0, len(X_test))
test_image = X_test[index]
true_label = y_test[index]

# Make a prediction
prediction = np.argmax(model.predict(test_image.reshape(1, 28, 28, 1)))

# Display the image and prediction
plt.imshow(test_image.squeeze(), cmap='gray')
plt.title(f"True Label: {true_label}, Predicted Label: {prediction}")
plt.axis('off')
plt.show()
```

RESULTS



```
Epoch 1/5
844/844 [==============================] - 41s 48ms/step - loss: 0.1720 - accuracy: 0.9478 - val_loss: 0.0564 - val_accuracy: 0.9833
Epoch 2/5
844/844 [==============================] - 40s 47ms/step - loss: 0.0495 - accuracy: 0.9846 - val_loss: 0.0380 - val_accuracy: 0.9888
Epoch 3/5
844/844 [==============================] - 40s 47ms/step - loss: 0.0330 - accuracy: 0.9892 - val_loss: 0.0341 - val_accuracy: 0.9908
Epoch 4/5
844/844 [==============================] - 42s 49ms/step - loss: 0.0250 - accuracy: 0.9920 - val_loss: 0.0476 - val_accuracy: 0.9883
Epoch 5/5
844/844 [==============================] - 39s 47ms/step - loss: 0.0192 - accuracy: 0.9939 - val_loss: 0.0357 - val_accuracy: 0.9905
313/313 [==============================] - 2s 8ms/step - loss: 0.0260 - accuracy: 0.9923
Test accuracy: 0.9922999739646912
1/1 [==============================] - 0s 102ms/step
```

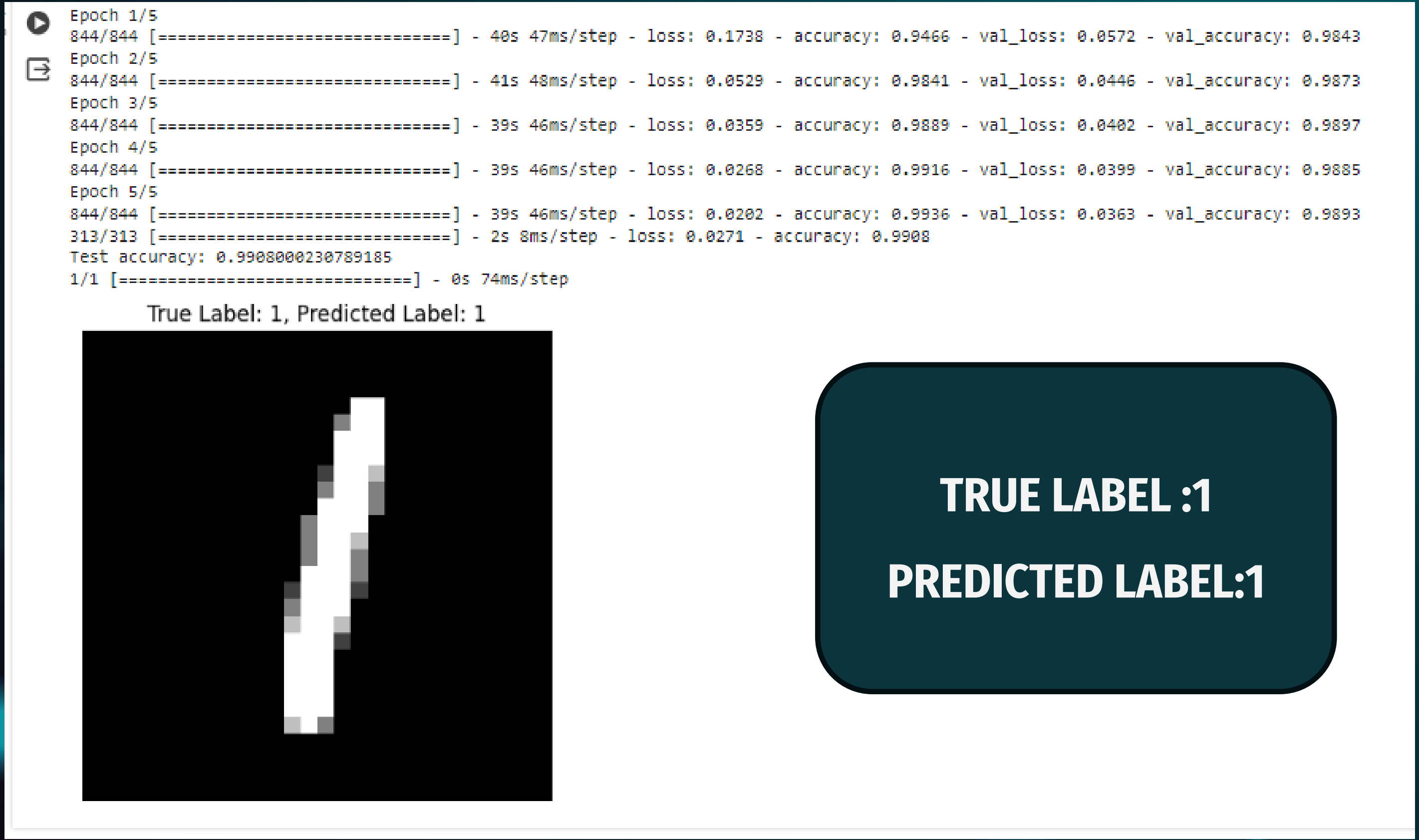
True Label: 0, Predicted Label: 0



TRUE LABEL :0

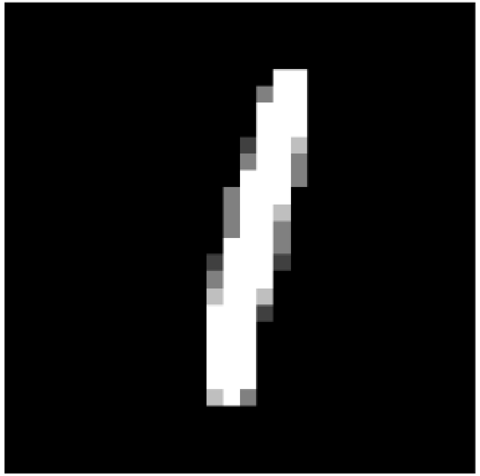
PREDICTED LABEL:0

RESULTS (CONTD)



```
Epoch 1/5
844/844 [==============================] - 40s 47ms/step - loss: 0.1738 - accuracy: 0.9466 - val_loss: 0.0572 - val_accuracy: 0.9843
Epoch 2/5
844/844 [==============================] - 41s 48ms/step - loss: 0.0529 - accuracy: 0.9841 - val_loss: 0.0446 - val_accuracy: 0.9873
Epoch 3/5
844/844 [==============================] - 39s 46ms/step - loss: 0.0359 - accuracy: 0.9889 - val_loss: 0.0402 - val_accuracy: 0.9897
Epoch 4/5
844/844 [==============================] - 39s 46ms/step - loss: 0.0268 - accuracy: 0.9916 - val_loss: 0.0399 - val_accuracy: 0.9885
Epoch 5/5
844/844 [==============================] - 39s 46ms/step - loss: 0.0202 - accuracy: 0.9936 - val_loss: 0.0363 - val_accuracy: 0.9893
313/313 [==============================] - 2s 8ms/step - loss: 0.0271 - accuracy: 0.9908
Test accuracy: 0.9908000230789185
1/1 [==============================] - 0s 74ms/step
```

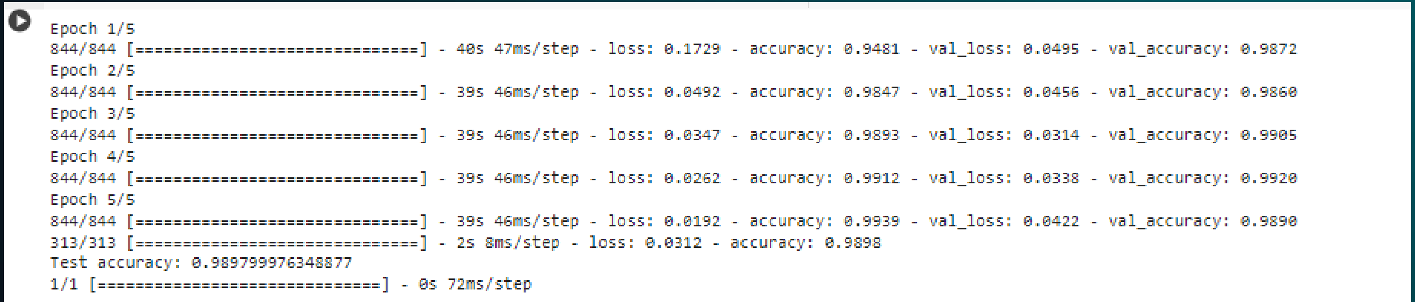
True Label: 1, Predicted Label: 1



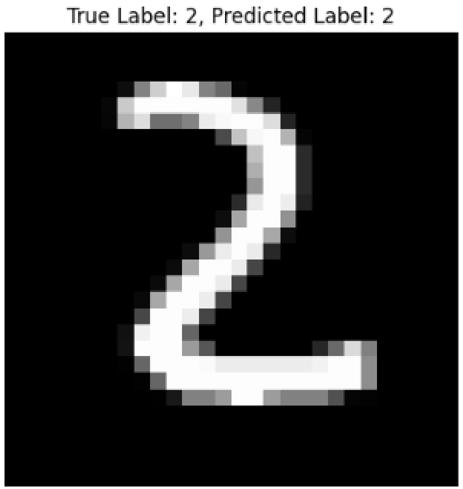
TRUE LABEL :1

PREDICTED LABEL:1

RESULTS (CONTD)



```
Epoch 1/5
844/844 [==============================] - 40s 47ms/step - loss: 0.1729 - accuracy: 0.9481 - val_loss: 0.0495 - val_accuracy: 0.9872
Epoch 2/5
844/844 [==============================] - 39s 46ms/step - loss: 0.0492 - accuracy: 0.9847 - val_loss: 0.0456 - val_accuracy: 0.9860
Epoch 3/5
844/844 [==============================] - 39s 46ms/step - loss: 0.0347 - accuracy: 0.9893 - val_loss: 0.0314 - val_accuracy: 0.9905
Epoch 4/5
844/844 [==============================] - 39s 46ms/step - loss: 0.0262 - accuracy: 0.9912 - val_loss: 0.0338 - val_accuracy: 0.9920
Epoch 5/5
844/844 [==============================] - 39s 46ms/step - loss: 0.0192 - accuracy: 0.9939 - val_loss: 0.0422 - val_accuracy: 0.9890
313/313 [==============================] - 2s 8ms/step - loss: 0.0312 - accuracy: 0.9898
Test accuracy: 0.989799976348877
1/1 [==============================] - 0s 72ms/step
```



TRUE LABEL :2

PREDICTED LABEL:2

SUMMARY OF RESULTS

Model Training And Model Evaluation : The model is trained for 5 epochs with a batch size of 64 and achieves a validation accuracy of around 99%. : After training, the model is evaluated on the test set. The test accuracy obtained is printed, indicating how well the model generalizes to unseen data.

Prediction: A random test image is selected, and the model predicts its label. The true label of the image is also displayed.

Result: The test accuracy obtained is typically high, reflecting the effectiveness of the CNN model in recognizing handwritten digits. The prediction on the random test image is also likely to be accurate, demonstrating the model's ability to make correct predictions.

CONCLUSION: The CNN model trained on the MNIST dataset achieves high accuracy in digit recognition. This suggests that CNNs are suitable for image classification tasks, especially for datasets like MNIST, which consist of grayscale images of handwritten digits.

THANK YOU

BY:
DAYANA DEVI K