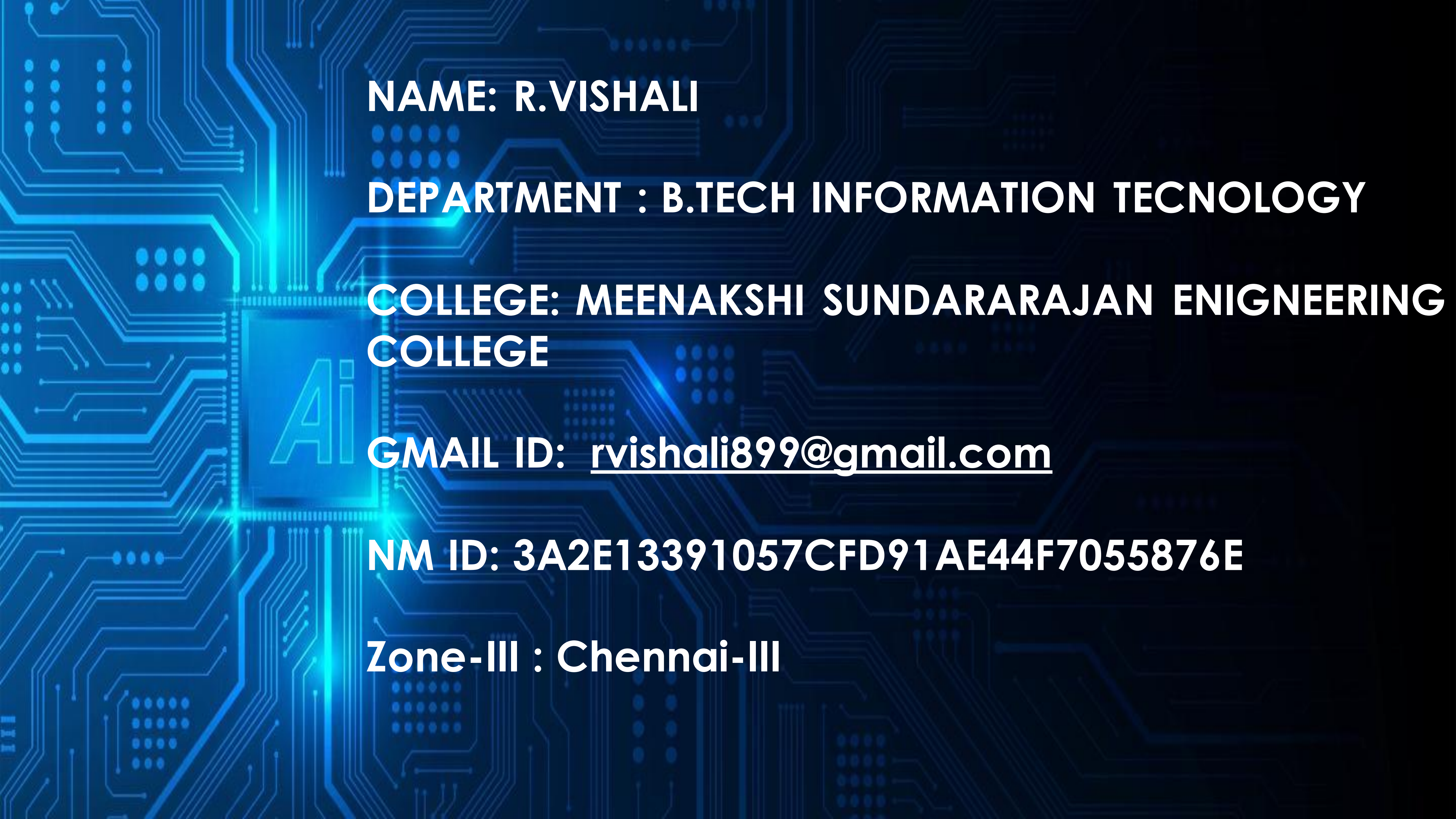
HAND WRITTEN DIGIT RECOGINITION WITH RNN



NAME: R.VISHALI

DEPARTMENT : B.TECH INFORMATION TECNOLOGY

COLLEGE: MEENAKSHI SUNDARARAJAN ENIGNEERING COLLEGE

GMAIL ID: rvishali899@gmail.com

NM ID: 3A2E13391057CFD91AE44F7055876E

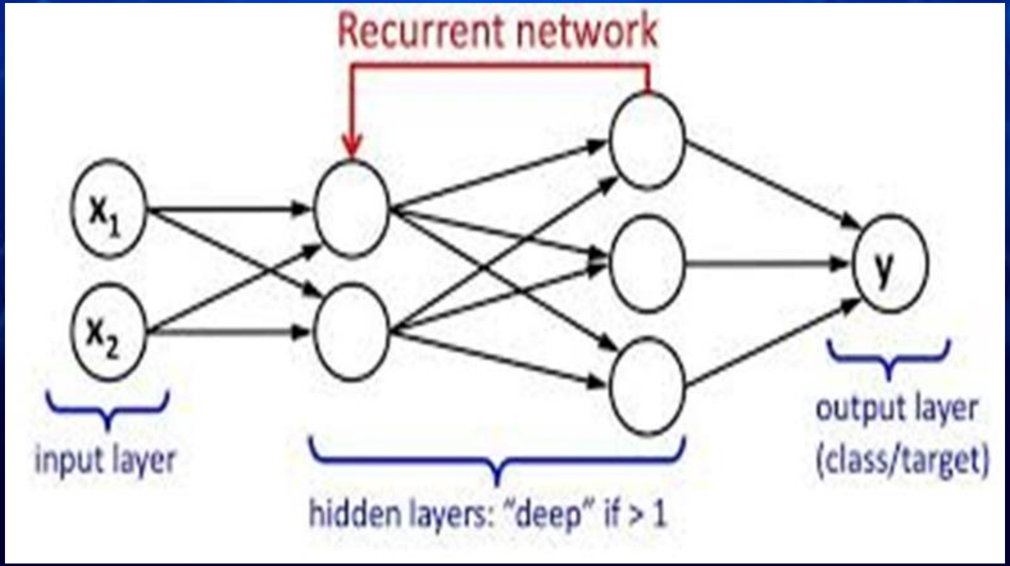
Zone-III : Chennai-III

AGENDA

- ❑ PROBLEM STATEMENT
- ❑ WHAT ARE RNN?
- ❑ WHY RNN?
- ❑ HOW DOES RNN WORK?
- ❑ PROJECT OVERVIEW
- ❑ WHO ARE END USERS?
- ❑ SOLUTION AND ITS VALUE PROPOSITION
- ❑ THE WOW IN MY SOLUTION
- ❑ MODELLING
- ❑ CODE IMPLEMENTATION
- ❑ RESULT

What are Recurrent Neural Networks (RNN)?

The output of a particular layer and feeding this back to the input in order to predict the output of theRecurrent Neural Networks (RNNs) are a type of artificial neural network designed to process sequences of data. They work especially well for jobs requiring sequences, such as time series data, voice, natural language, and other activities.
RNN works on the principle of saving e layer



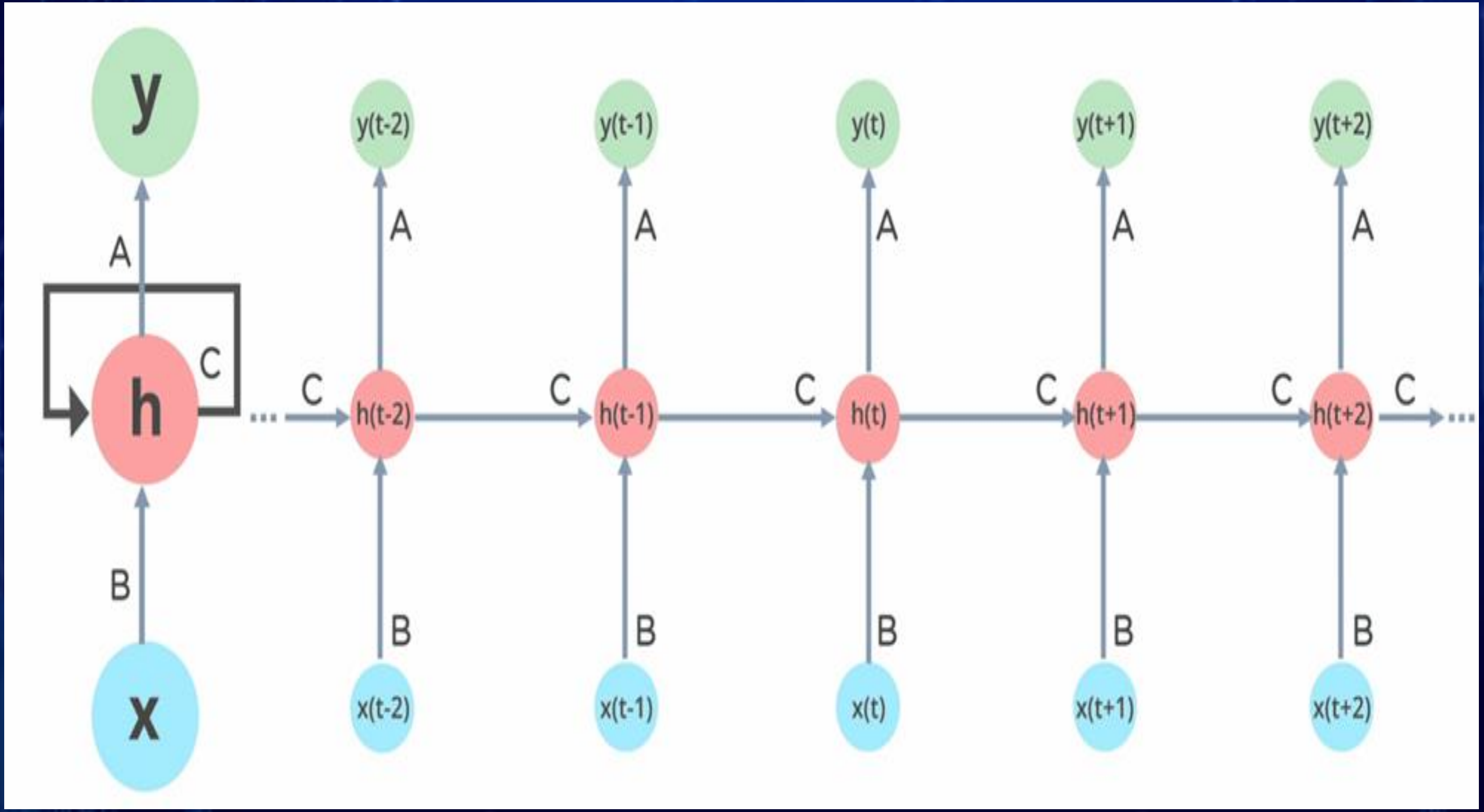
Why Recurrent Neural Networks?

RNN were created because there were a few issues in the feed-forward neural network:

•Cannot handle sequential data
•Considers only the current input
•Cannot memorize previous inputs
The solution to these issues is the RNN. An RNN can handle sequential data, accepting the current input data, and previously received inputs. RNNs can memorize previous inputs due to their internal memory

How Does Recurrent Neural Networks Work?

In Recurrent Neural networks, the information cycles through a loop to the middle hidden layer.



<u>These are the following process taken in the above diagram</u>

The input layer ‘x’ takes in the input to the neural network and processes it and passes it onto the middle layer.

The middle layer ‘h’ can consist of multiple hidden layers, each with its own activation functions and weights and biases.

If you have a neural network where the various parameters of different hidden layers are not affected by the previous layer, ie: the neural network does not have memory, then you can use a recurrent neural network.

The Recurrent Neural Network will standardize the different activation functions and weights and biases so that each hidden layer has the same parameters. Then, instead of creating multiple hidden layers, it will create one and loop over it as many times as required.

PROBLEM STATEMENT

- The task is to create an RNN model using TensorFlow and Keras for handwritten digit classification from the MNIST dataset. The pixel values are first normalized, and then a Sequential model with a Simple RNN layer is defined to process the sequential data of images.

- The model is compiled with the Adam optimizer and sparse categorical cross-entropy loss. After reshaping the input data to fit the RNN input shape, the model is trained for 5 epochs with a batch size of 64, with 10% of the training data used for validation. The trained model's accuracy is evaluated on the test set.

- Finally, a random test image is chosen for prediction, and the model's predicted label is visualized alongside the true label. The script provides insights into training progress, prints test accuracy, and showcases the model's prediction capability.

PROJECT OVERVIEW

DATA PREPARATION

↓

MODEL ARCHITECTURE

↓

MODEL COMPILATION

↓

TRAIN THE MODEL

↓

EVALUATE THE MODEL

↓

MAKE PREDICTIONS

WHO ARE END USERS?

INDIVIDUALS	• These are everyday individuals who interact with the application for various purposes, such as digitizing handwritten documents, recognizing handwritten numbers in forms, or using handwriting recognition as an input method for digital devices.
RESERCHERS	• These are individuals who may use the system for experimental purposes, studying its performance, improving algorithms, or integrating it into more complex systems.
BUSINESS	• Companies or institutions may use handwritten digit recognition systems for tasks like automatic sorting of mail, processing checks or invoices, or organizing handwritten data in databases.
GOVERNMENT AGENCIES	• Government departments may utilize handwritten digit recognition systems for tasks like processing handwritten forms, digitizing historical records, or verifying handwritten signatures.

SOLUTION AND ITS VALUE PROPOSITION

1. **Accurate Recognition**: RNNs are well-suited for sequential data like handwriting because they can capture temporal dependencies. This enables the model to accurately recognize handwritten digits even when they are written in a continuous stroke.

2. **Flexibility**: RNNs can handle inputs of varying lengths, making them adaptable to different styles of handwriting and different writing speeds. This flexibility allows the system to be used across a wide range of applications and scenarios.

3. **User-friendly Interface**: By integrating the RNN-based recognition system into user-friendly applications or devices, such as mobile apps or touchscreen interfaces, the solution can offer a seamless and intuitive user experience. This can increase user adoption and satisfaction.

4.Automation and Efficiency: By automating the process of digit recognition, the solution can save time and effort for users who would otherwise have to manually transcribe handwritten digits. This can lead to increased productivity and cost savings in various industries, such as banking, logistics, and healthcare

5.Scalability: RNN-based digit recognition solutions can be scaled to handle large volumes of handwritten data, making them suitable for both small-scale applications and enterprise-level deployments. This scalability ensures that the solution remains effective as the volume of data increases over time.

6.Customization and Adaptability: The RNN architecture allows for easy customization and fine-tuning to specific use cases or datasets. This adaptability enables the solution to achieve high levels of accuracy even in challenging environments or with limited training data.

THE WOW IN MY SOLUTION

- **Efficient Model Design**: The model architecture is straightforward yet effective. By using just a single-layer Simple RNN followed by a Dense layer, it achieves reasonable performance on the MNIST dataset. This simplicity highlights the power of deep learning frameworks like TensorFlow and Keras, which enable rapid prototyping and experimentation.

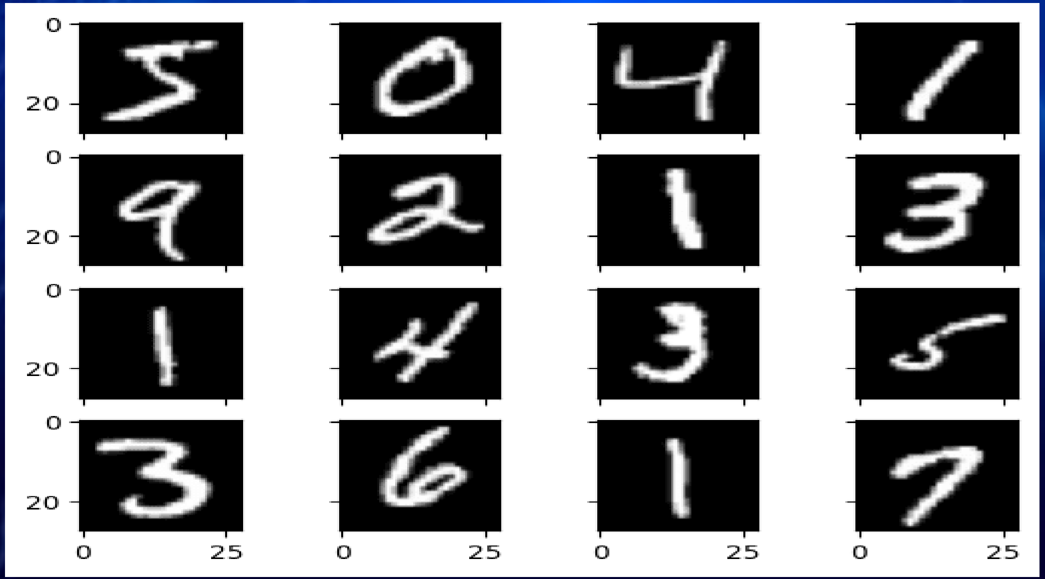
- **Clear Training Procedure**: The training procedure is clearly defined and implemented. The code specifies the number of epochs, batch size, and validation split, providing a structured approach to model training.

- **Visualization**: The code goes the extra mile by visualizing a random test image along with its true label and predicted label. This visual feedback helps in understanding the model's behavior and provides insights into its strengths and weaknesses.

MODELLING

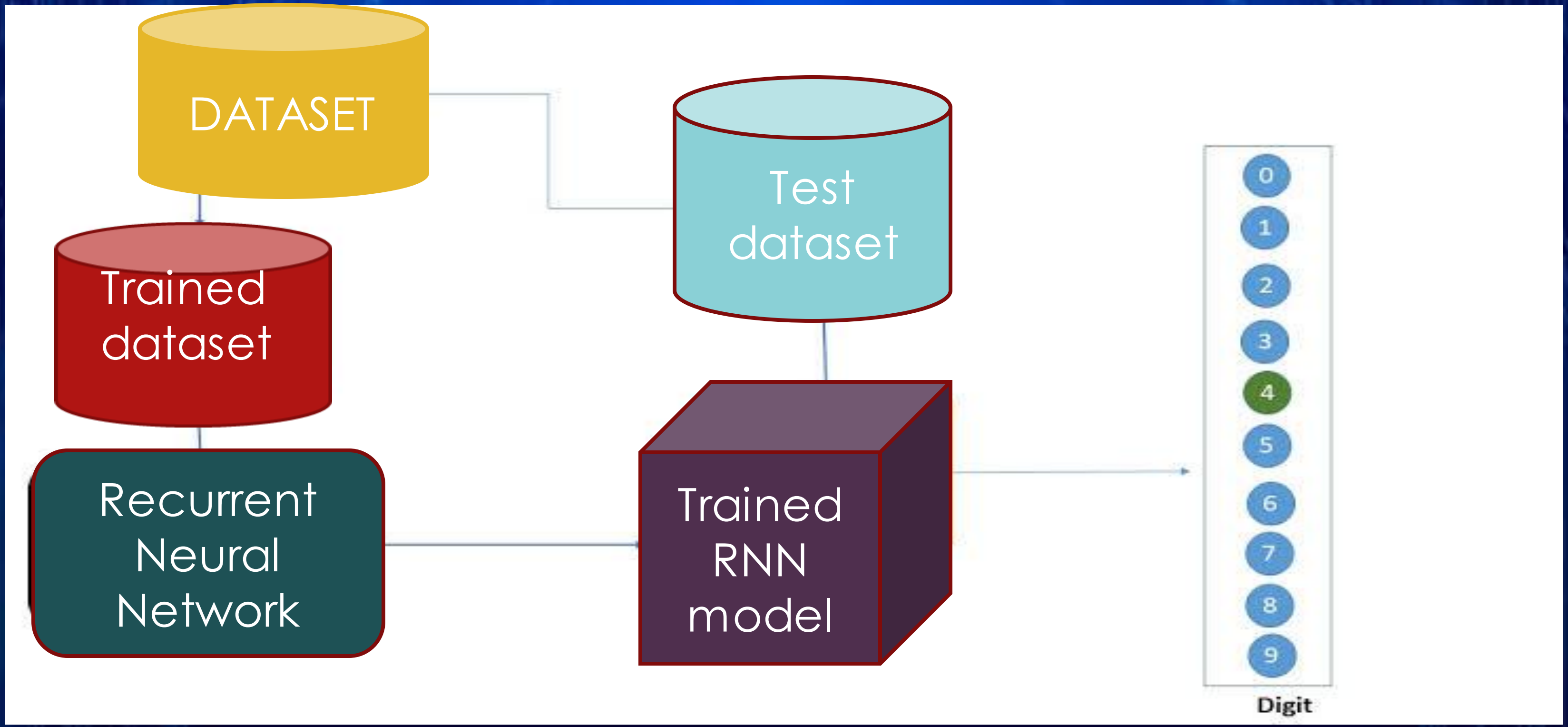
1.Data Preparation:

Utilize TensorFlow's dataset module to load the MNIST dataset, which is divided into training and test sets.
Normalize the pixel values of the images to the range [0, 1] by dividing by 255.



2.Model Architecture:

Construct a Sequential model in Keras.Incorporate a Simple RNN layer with 128 units to process the sequential nature of the input images. The input shape is set to (28, 28), reflecting the dimensions of each image.Add a Dense layer with 10 units and a softmax activation function, enabling the model to output probabilities for each digit class (0 through 9).

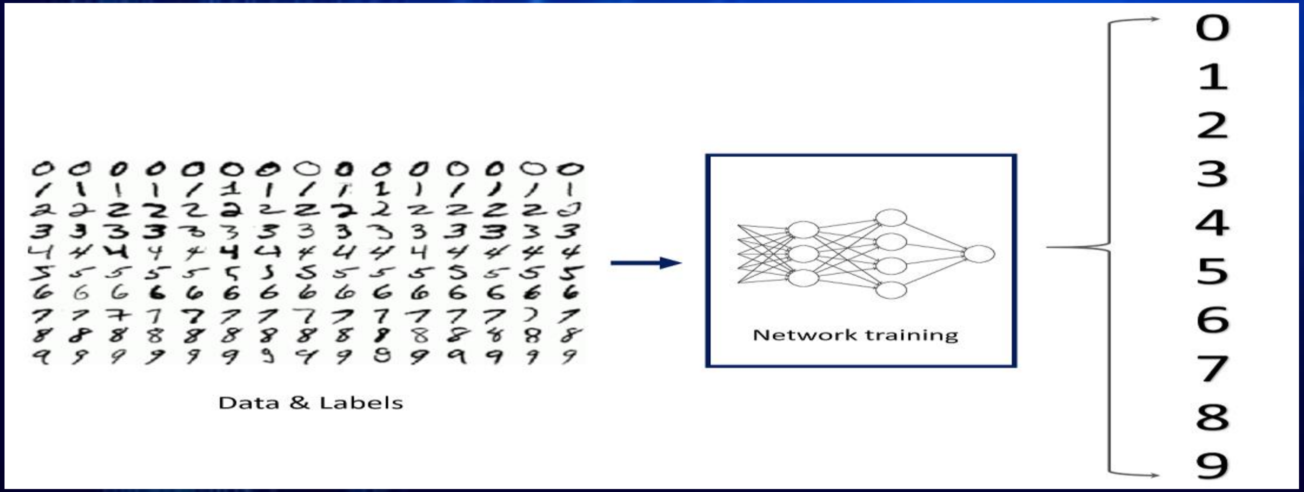


3.Model Compilation:

Compile the model using the Adam optimizer, which is well-suited for training deep neural networks.Define the loss function as sparse categorical cross-entropy, suitable for multiclass classification tasks.Monitor model performance during training using accuracy as the metric.

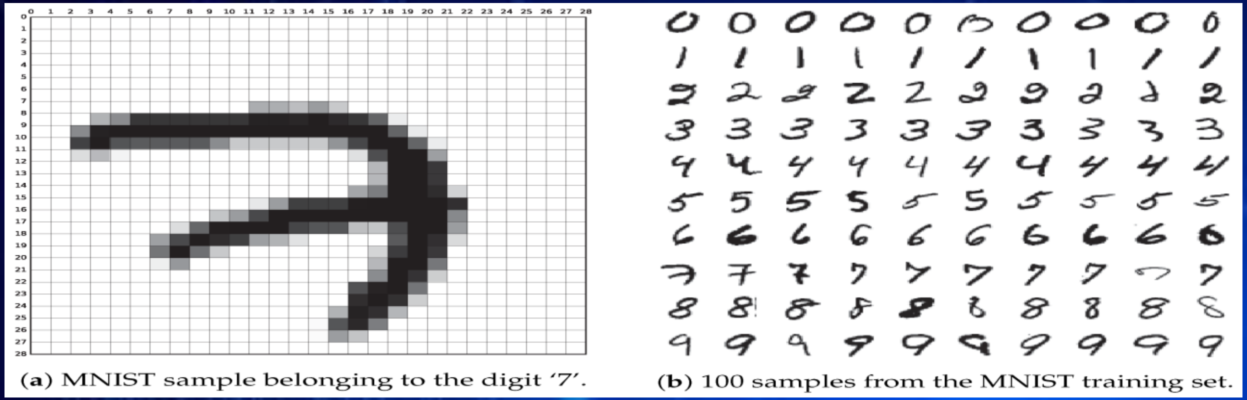
4.Model Training:

Train the model on the training data for 5 epochs with a batch size of 64. Additionally, allocate 10% of the training data for validation to monitor model performance during training.



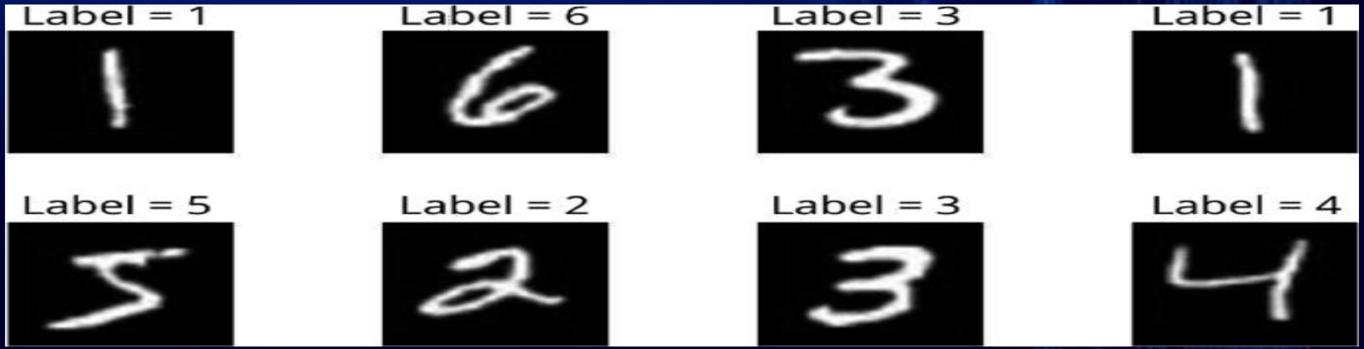
5.Model Evaluation:

Evaluate the trained model on the test set to assess its generalization performance. Print the test accuracy, indicating the model's ability to correctly classify unseen data

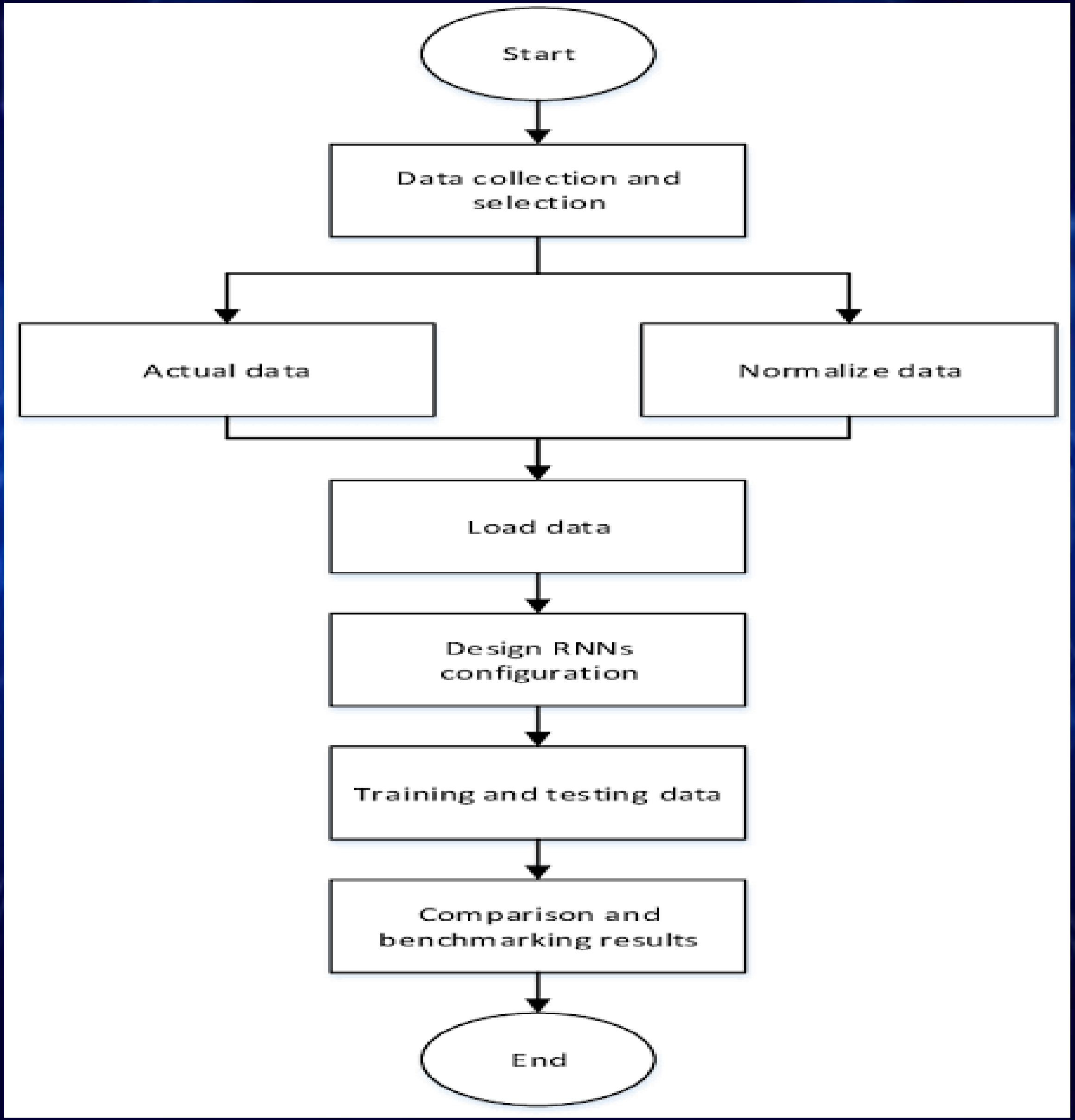

(a) MNIST sample belonging to the digit '7'. (b) 100 samples from the MNIST training set.

6.Prediction and Visualization:

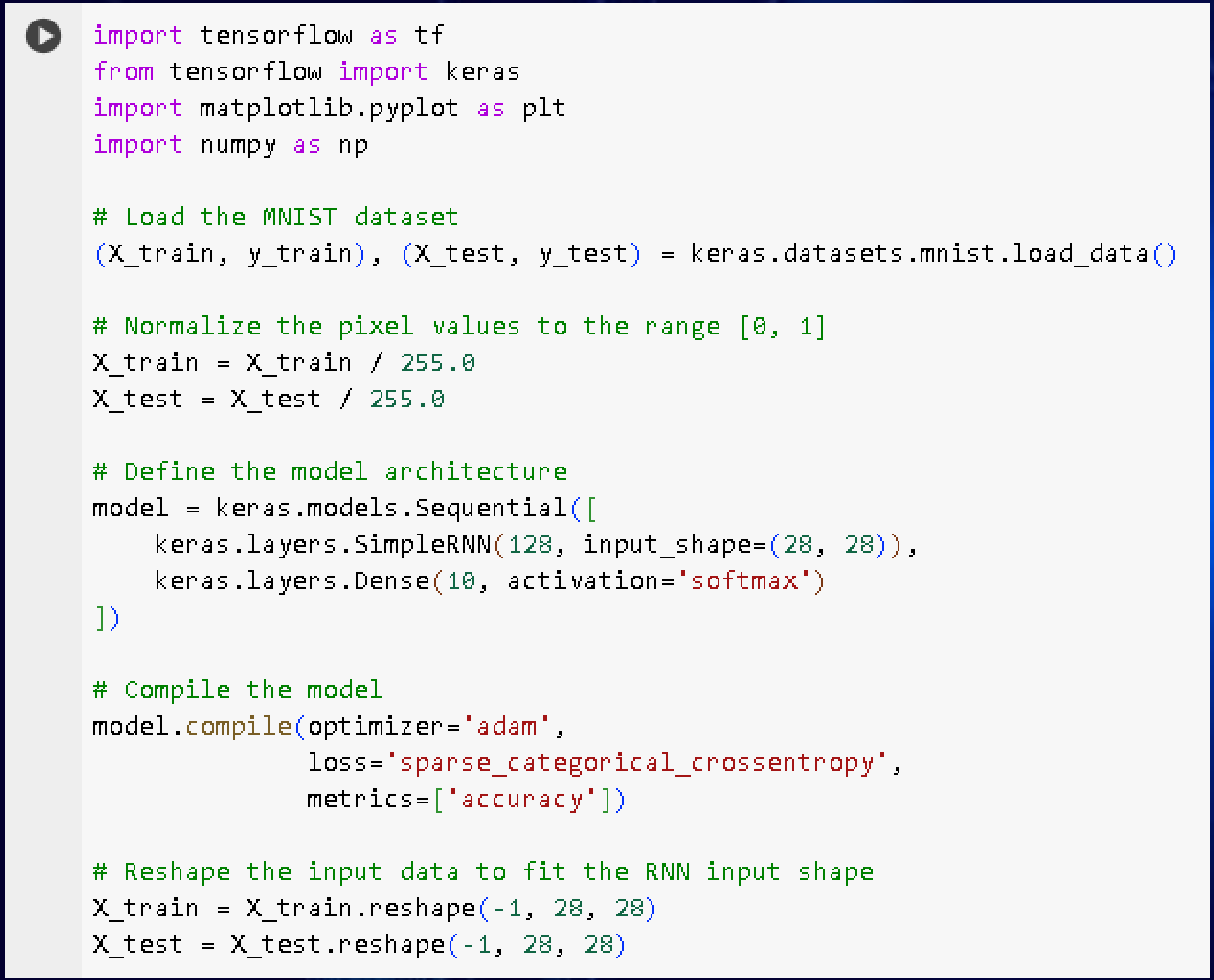
Select a random test image from the dataset.Utilize the trained model to predict the label of the selected image. Visualize the test image alongside its true label and the predicted label using Matplotlib.



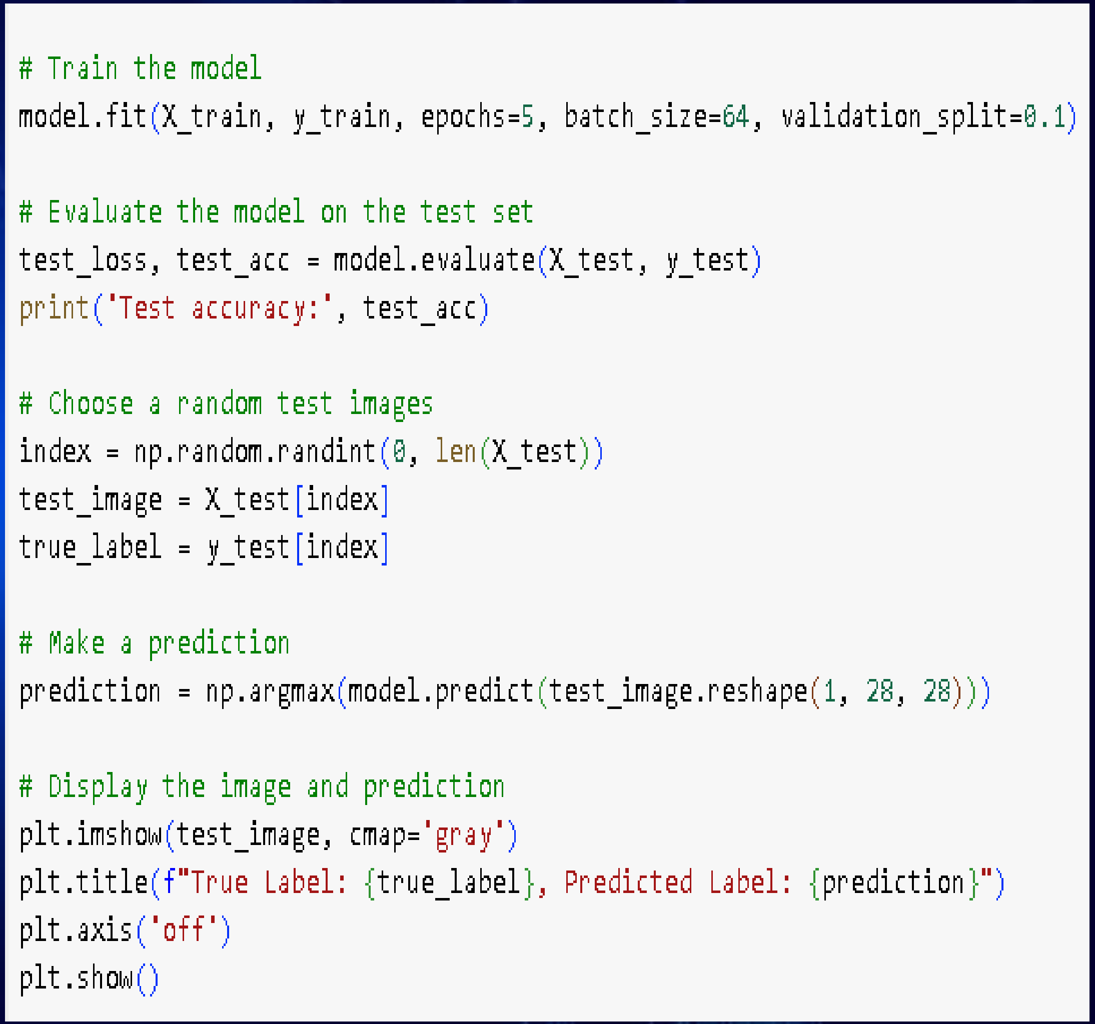
FLOW CHART


Start
Data collection and selection
Actual data
Normalize data
Load data
Design RNNs configuration
Training and testing data
Comparison and benchmarking results
End

CODE IMPLEMENTATION

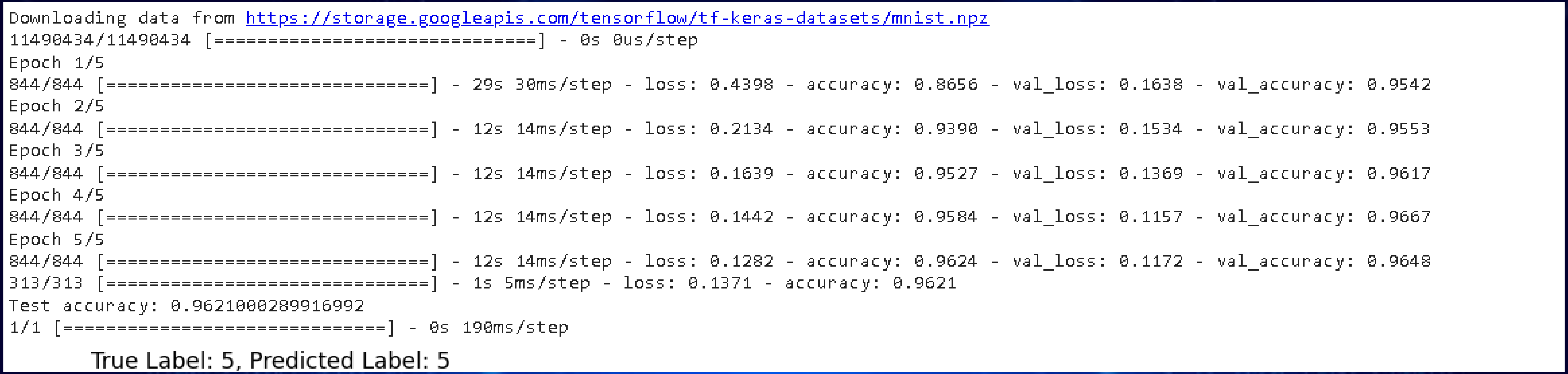


```python
import tensorflow as tf
from tensorflow import keras
import matplotlib.pyplot as plt
import numpy as np

# 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

# Define the model architecture
model = keras.models.Sequential([
    keras.layers.SimpleRNN(128, input_shape=(28, 28)),
    keras.layers.Dense(10, activation='softmax')
])

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

# Reshape the input data to fit the RNN input shape
X_train = X_train.reshape(-1, 28, 28)
X_test = X_test.reshape(-1, 28, 28)
```



```python
# 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 images
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)))

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

RESULT



```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11490434/11490434 [==============================] - 0s 0us/step
Epoch 1/5
844/844 [==============================] - 29s 30ms/step - loss: 0.4398 - accuracy: 0.8656 - val_loss: 0.1638 - val_accuracy: 0.9542
Epoch 2/5
844/844 [==============================] - 12s 14ms/step - loss: 0.2134 - accuracy: 0.9390 - val_loss: 0.1534 - val_accuracy: 0.9553
Epoch 3/5
844/844 [==============================] - 12s 14ms/step - loss: 0.1639 - accuracy: 0.9527 - val_loss: 0.1369 - val_accuracy: 0.9617
Epoch 4/5
844/844 [==============================] - 12s 14ms/step - loss: 0.1442 - accuracy: 0.9584 - val_loss: 0.1157 - val_accuracy: 0.9667
Epoch 5/5
844/844 [==============================] - 12s 14ms/step - loss: 0.1282 - accuracy: 0.9624 - val_loss: 0.1172 - val_accuracy: 0.9648
313/313 [==============================] - 1s 5ms/step - loss: 0.1371 - accuracy: 0.9621
Test accuracy: 0.9621000289916992
1/1 [==============================] - 0s 190ms/step
```

True Label: 5, Predicted Label: 5