14.08.25

4.

4. Build a simple feed forward neural network to recognize handwritten character.

Aim:

To design and implement a simple feed-forward neural network (FFNN) using python to recognize handwritten characters from the MNIST dataset

Objectives:

1. To load and preprocess the MNIST handwritten Characters dataset
2. To build a feed-forward neural network with input, hidden, and output layer
3. To train the model using back propagation and gradients descent
4. To evaluate the trained model on a test dataset and report accuracy
5. To visualize some sample Predictions for verification.

Pseudocode:

Start.

Import necessary Libraries

Step 1: Load data

Load MNIST dataset (training_images, training_labels, testing_test_labels)

Step 2: Preprocess data

Step 2:

Preprocess data

Normalize image pixel values to range [0,1]

One-hot Encode labels for output layer

Compatibility

Step 3: Build model

Initialize a sequential feed-forward model

Add flatten layer to convert 28×28 to 784 vector

And Dense hidden layer with ReLU activation.

And Dense output layer with softmax activation.

Step 4:

Compile model

Choose optimizer = 'adam'

loss function = 'categorical_crossentropy'

metrics = 'accuracy'

Step 5: Train model

fit the model on training data for defined epochs, and batch size

Step 6:- Evaluate model

Predict Labels for sample test images

Plot images with predicted and actual labels

Output: Epoch 1/5

1688/1688 — accuracy: 0.8114 - loss: 0.4395
- val: accuracy: 0.9632 - val - loss: 0.1179.

Epoch 3/5

1688/1688 - accuracy: 0.9777 - loss: 0.0710
- val - accuracy: 0.9702 - val - loss: 0.0996

Epoch 5/5

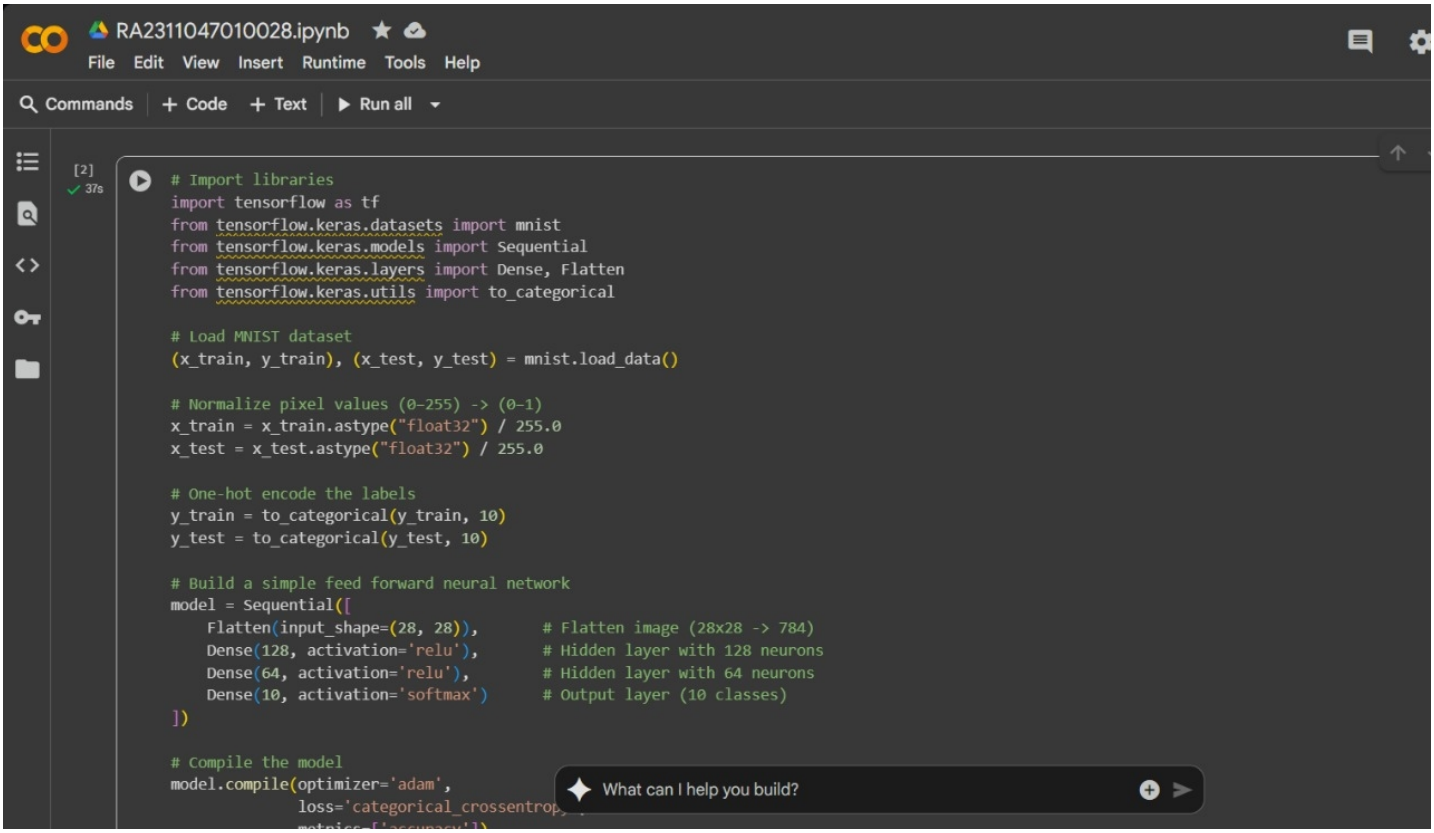
1688/1688 - accuracy: 0.9855 - loss 0.0440.
val - accuracy: 0.9765 :- val - loss: 0.0808

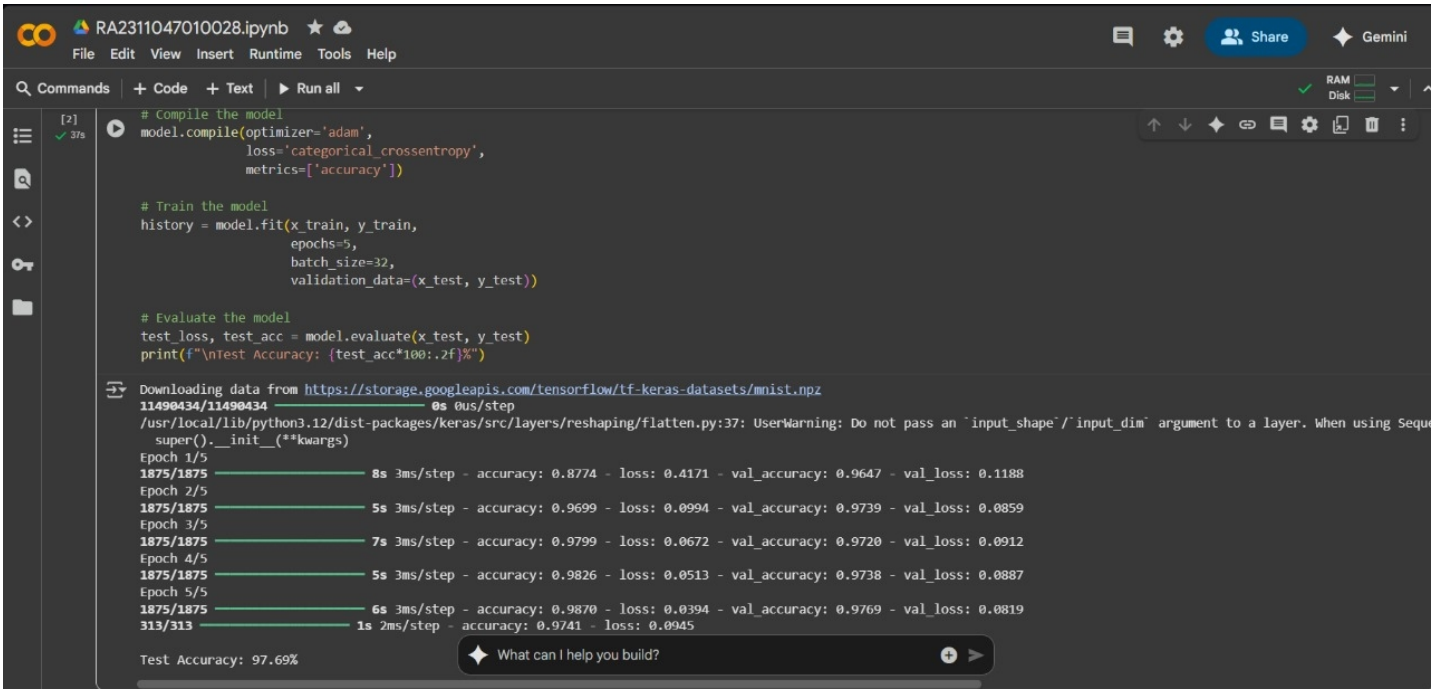
Test Accuracy: 0.9749.

12/8/25

Result:

The feed - forward neural network was implemented successfully using Tensorflow on the MINIST dataset achieved around 97+. accuracy in recognizing handwritten


RA2311047010028.ipynb
File Edit View Insert Runtime Tools Help
Commands + Code + Text Run all
[2]
37s
# Import libraries
import tensorflow as tf
from tensorflow.keras.datasets import mnist
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Flatten
from tensorflow.keras.utils import to_categorical
# Load MNIST dataset
(x_train, y_train), (x_test, y_test) = mnist.load_data()
# Normalize pixel values (0-255) -> (0-1)
x_train = x_train.astype("float32") / 255.0
x_test = x_test.astype("float32") / 255.0
# One-hot encode the labels
y_train = to_categorical(y_train, 10)
y_test = to_categorical(y_test, 10)
# Build a simple feed forward neural network
model = Sequential([
Flatten(input_shape=(28, 28)), # Flatten image (28x28 -> 784)
Dense(128, activation='relu'), # Hidden layer with 128 neurons
Dense(64, activation='relu'), # Hidden layer with 64 neurons
Dense(10, activation='softmax') # Output layer (10 classes)
])
# Compile the model
model.compile(optimizer='adam',
loss='categorical_crossentrop
What can I help you build?


RA2311047010028.ipynb
File Edit View Insert Runtime Tools Help
Share
Gemini
Commands
+ Code
+ Text
Run all
RAM
Disk
[2]
37s
# Compile the model
model.compile(optimizer='adam',
loss='categorical_crossentropy',
metrics=['accuracy'])
# Train the model
history = model.fit(x_train, y_train,
epochs=5,
batch_size=32,
validation_data=(x_test, y_test))
# Evaluate the model
test_loss, test_acc = model.evaluate(x_test, y_test)
print(f"\nTest Accuracy: {test_acc*100:.2f}%")
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11490434/11490434 0s 0us/step
/usr/local/lib/python3.12/dist-packages/keras/src/layers/reshaping/flatten.py:37: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequ
super().__init__(**kwargs)
Epoch 1/5
1875/1875 8s 3ms/step - accuracy: 0.8774 - loss: 0.4171 - val_accuracy: 0.9647 - val_loss: 0.1188
Epoch 2/5
1875/1875 5s 3ms/step - accuracy: 0.9699 - loss: 0.0994 - val_accuracy: 0.9739 - val_loss: 0.0859
Epoch 3/5
1875/1875 7s 3ms/step - accuracy: 0.9799 - loss: 0.0672 - val_accuracy: 0.9720 - val_loss: 0.0912
Epoch 4/5
1875/1875 5s 3ms/step - accuracy: 0.9826 - loss: 0.0513 - val_accuracy: 0.9738 - val_loss: 0.0887
Epoch 5/5
1875/1875 6s 3ms/step - accuracy: 0.9870 - loss: 0.0394 - val_accuracy: 0.9769 - val_loss: 0.0819
313/313 1s 2ms/step - accuracy: 0.9741 - loss: 0.0945
Test Accuracy: 97.69%
What can I help you build?