14/08/25

Exp-4 Build a simple feed-forward neural network to recognize handwritten characters"

Aim :

To design and implement a simple feed-forward neural network using Python to recognize handwritten characters from a dataset.

Description :

A feed forward neural network (FNN) is a type of artificial neural network where connections between nodes do not form a cycle. In this experiment, the FNN will be trained on a dataset of handwritten characters to classify them into respective categories. The model consists of an input layer, one or more hidden layers with activation functions, and an output layer using softmax for classification.

Precision & Recall :

- Precision = Correctly predicted positive observations / Total predicted positive observations.

- Recall = Correctly predicted positive observations/ All actual positive observations.

These metrics evaluate classification performance beyond accuracy, especially when class distribution is imbalanced.

Confusion Matrix :

Actual|Predicted	0	1.	2	3
0	95	0	1	0
1	0	93	2	1
2	1	1	96	0
3	0	2	1	94

Procedure :

1.) Load the handwritten character dataset (MNIST)

2.) Normalize pixel values between 0 and 1

3.) Flatten the images into 1D arrays.

4.) Create a feed-forward neural network with :

- Input layer
- Hidden Layer (ReLU)
- Output Layer (Softmax)

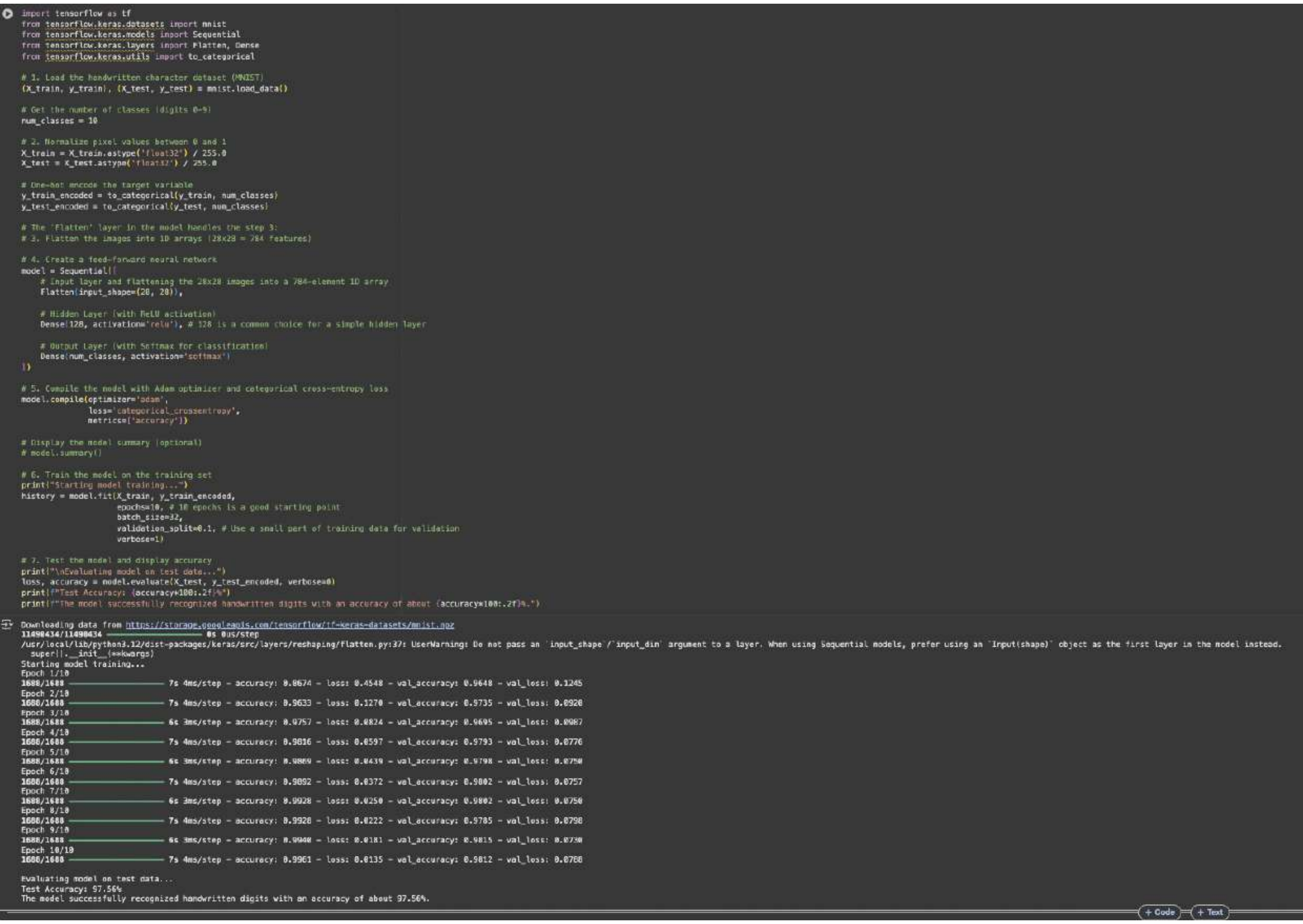
5.) Compile the model with Adam optimizer and categorical cross-entropy loss.

6.) Train the model on the training set

7.) Test the model and display accuracy.

13/10/25

Result: The feed-forward neural network recognized handwritten digits with an accuracy of about 97%. This model which is effective is working successfully.



```
import tensorflow as tf
from tensorflow.keras.datasets import mnist
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Flatten, Dense
from tensorflow.keras.utils import to_categorical

# 1. Load the handwritten character dataset (MNIST)
(X_train, y_train), (X_test, y_test) = mnist.load_data()

# Get the number of classes (digits 0-9)
num_classes = 10

# 2. Normalize pixel values between 0 and 1
X_train = X_train.astype('float32') / 255.0
X_test = X_test.astype('float32') / 255.0

# One-hot encode the target variable
y_train_encoded = to_categorical(y_train, num_classes)
y_test_encoded = to_categorical(y_test, num_classes)

# The 'Flatten' layer in the model handles the step 3:
# 3. Flatten the images into 1D arrays (28x28 = 784 features)

# 4. Create a feed-forward neural network
model = Sequential([
    # Input layer and flattening the 28x28 images into a 784-element 1D array
    Flatten(input_shape=(28, 28)),

    # Hidden Layer (with ReLU activation)
    Dense(128, activation='relu'), # 128 is a common choice for a simple hidden layer

    # Output Layer (with Softmax for classification)
    Dense(num_classes, activation='softmax')
])

# 5. Compile the model with Adam optimizer and categorical cross-entropy loss
model.compile(optimizer='adam',
              loss='categorical_crossentropy',
              metrics=['accuracy'])

# Display the model summary (optional)
# model.summary()

# 6. Train the model on the training set
print("Starting model training...")
history = model.fit(X_train, y_train_encoded,
                    epochs=10, # 10 epochs is a good starting point
                    batch_size=32,
                    validation_split=0.1, # Use a small part of training data for validation
                    verbose=1)

# 7. Test the model and display accuracy
print("\nEvaluating model on test data...")
loss, accuracy = model.evaluate(X_test, y_test_encoded, verbose=0)
print(f"Test Accuracy: {accuracy*100:.2f}%")
print(f"The model successfully recognized handwritten digits with an accuracy of about {accuracy*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 Sequential models, prefer using an `Input(shape)` object as the first layer in the model instead.
  super().__init__(**kwargs)
Starting model training...
Epoch 1/10
1688/1688 ━━━━━━━━━━━━━━━━━━━━ 7s 4ms/step - accuracy: 0.8674 - loss: 0.4548 - val_accuracy: 0.9648 - val_loss: 0.1245
Epoch 2/10
1688/1688 ━━━━━━━━━━━━━━━━━━━━ 7s 4ms/step - accuracy: 0.9633 - loss: 0.1270 - val_accuracy: 0.9735 - val_loss: 0.0920
Epoch 3/10
1688/1688 ━━━━━━━━━━━━━━━━━━━━ 6s 3ms/step - accuracy: 0.9757 - loss: 0.0824 - val_accuracy: 0.9695 - val_loss: 0.0987
Epoch 4/10
1688/1688 ━━━━━━━━━━━━━━━━━━━━ 7s 4ms/step - accuracy: 0.9816 - loss: 0.0597 - val_accuracy: 0.9793 - val_loss: 0.0776
Epoch 5/10
1688/1688 ━━━━━━━━━━━━━━━━━━━━ 6s 3ms/step - accuracy: 0.9869 - loss: 0.0439 - val_accuracy: 0.9798 - val_loss: 0.0750
Epoch 6/10
1688/1688 ━━━━━━━━━━━━━━━━━━━━ 7s 4ms/step - accuracy: 0.9892 - loss: 0.0372 - val_accuracy: 0.9802 - val_loss: 0.0757
Epoch 7/10
1688/1688 ━━━━━━━━━━━━━━━━━━━━ 6s 3ms/step - accuracy: 0.9928 - loss: 0.0250 - val_accuracy: 0.9802 - val_loss: 0.0750
Epoch 8/10
1688/1688 ━━━━━━━━━━━━━━━━━━━━ 7s 4ms/step - accuracy: 0.9928 - loss: 0.0222 - val_accuracy: 0.9785 - val_loss: 0.0798
Epoch 9/10
1688/1688 ━━━━━━━━━━━━━━━━━━━━ 6s 3ms/step - accuracy: 0.9940 - loss: 0.0181 - val_accuracy: 0.9815 - val_loss: 0.0730
Epoch 10/10
1688/1688 ━━━━━━━━━━━━━━━━━━━━ 7s 4ms/step - accuracy: 0.9961 - loss: 0.0135 - val_accuracy: 0.9812 - val_loss: 0.0788

Evaluating model on test data...
Test Accuracy: 97.56%
The model successfully recognized handwritten digits with an accuracy of about 97.56%.
```