14/8/25

week-4

Aim: To build a simple feed forward Neural Network to recognize handwritten character, using MNIST Dataset.

Objective :-

1) To understand the architecture of a feed forward Neural Network.
2) To Apply supervised learning for image classification.
3) To preprocess MNIST Dataset for Neural Network training.
4) To evaluate the models accuracy and analyze the performance.

Procedure :-

1) Import libraries (Tensorflow, keras, matplotlib)
2) Load MNIST Dataset.
3) Normalize the data (pixel values)
4) one-hot encode labels (0-9 into vector length of 10)
5) Define model architecture:
   → Flatten layer (convert 28x28 images into 784 inputs)
   → Dense hidden layer with Relu activation.
   → Dense output layer with softmax activation
6) Compile model with optimizer, loss function, metrics.
7) Train the model on training Data.
8) Evaluate performance on test data.
9) plot accuracy graph for train and validation sets.

pseudo code :-

START
Import Tensorflow, keras, matplotlib
Load MNIST dataset
Normalize input images to range [0, 1]
Convert labels to one-hot encoding
Initialize sequential model
Add flatten layer for 28x28 input

Add dense hidden layer with ReLU activation
Add dense output layer with softmax activation
compile model with Adam optimizer, categorical cross entropy loss, accuracy metrics
Train model for 5 epochs
Evaluate model on test data
plot training vs validation accuracy
predict labels for first 5 test samples
Display predicted labels with images
END

Observations:-

*) The model achieved ~97-98% accuracy on the MNIST test dataset with just 5 epochs of training.

*) Accuracy improved steadily with each epoch, indicating effective learning.

*) Predictions on unseen data matched the actual digits in most cases.

*) Errors occured mainly on digits that were poorly written.

Conclusion:-

A simple feed forward Neural Network with one hidden layer can accurately classify handwritten digits from the MNIST dataset.

The performance shows that FFNN's are effective for basic image recognization tasks after Normalization and one-hot encoding.

However, for more complex image recognition problems, deeper architectures like convolutional neural networks may be more suitable.

Output :-

Epoch 1/5
val_accuracy : 0.9596 - val_loss : 0.1386

Epoch 2/5
val_accuracy : 0.9706 - val_loss : 0.0983

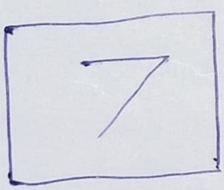
Epoch 3/5
val_accuracy : 0.9716 - val_loss : 0.0882

Epoch 4/5
val_accuracy : 0.9759 - val_loss : 0.0763

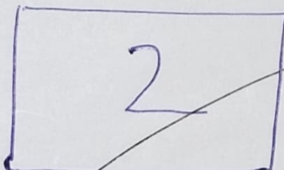
Epoch 5/5
val_accuracy : 0.9748 - val_loss : 0.0815

Test accuracy : 0.9748

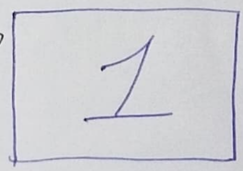
predicted : 7



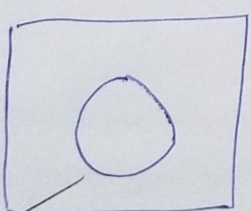
predicted : 2



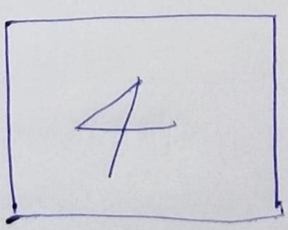
predicted : 1



predicted : 0



predicted : 4



python.ipynb

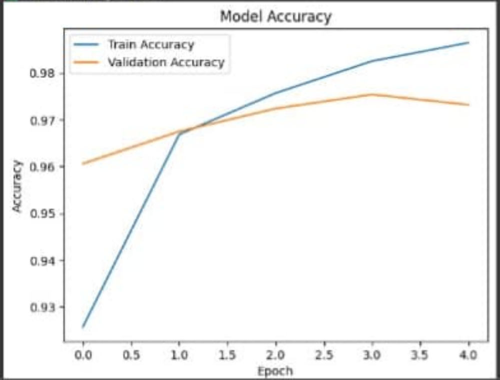
File Edit View Insert Runtime Tools Help

Commands + Code + Text Run all

```python
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
import matplotlib.pyplot as plt
(x_train, y_train), (x_test, y_test) = mnist.load_data()
x_train = x_train / 255.0
x_test = x_test / 255.0
y_train = to_categorical(y_train, 10)
y_test = to_categorical(y_test, 10)
model = Sequential()
model.add(Flatten(input_shape=(28, 28)))
model.add(Dense(128, activation='relu'))
model.add(Dense(10, activation='softmax'))
model.compile(optimizer='adam',
              loss='categorical_crossentropy',
              metrics=['accuracy'])
history = model.fit(x_train, y_train,
                    epochs=5,
                    batch_size=32,
                    validation_data=(x_test, y_test))
test_loss, test_acc = model.evaluate(x_test, y_test)
print(f"\n✅ Test Accuracy: {test_acc:.4f}")
plt.plot(history.history['accuracy'], label='Train Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.title('Model Accuracy')
plt.legend()
plt.show()
predictions = model.predict(x_test[:5])
for i in range(5):
    plt.imshow(x_test[i], cmap='gray')
    plt.title(f"Predicted: {predictions[i].argmax()}")
    plt.axis('off')
    plt.show()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11490434/11490434 ━━━━━━━━━━━━━━━━━━━━ 1s 0us/step
/usr/local/lib/python3.11/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
  super().__init__(**kwargs)
Epoch 1/5
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 11s 5ms/step - accuracy: 0.8780 - loss: 0.4321 - val_accuracy: 0.9606 - val_loss: 0.1322
Epoch 2/5
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 9s 5ms/step - accuracy: 0.9647 - loss: 0.1210 - val_accuracy: 0.9675 - val_loss: 0.1034
Epoch 3/5
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 10s 5ms/step - accuracy: 0.9753 - loss: 0.0772 - val_accuracy: 0.9724 - val_loss: 0.0881
Epoch 4/5
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 11s 5ms/step - accuracy: 0.9833 - loss: 0.0553 - val_accuracy: 0.9754 - val_loss: 0.0769
Epoch 5/5
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 9s 5ms/step - accuracy: 0.9881 - loss: 0.0419 - val_accuracy: 0.9732 - val_loss: 0.0875
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.9684 - loss: 0.1032

✅ Test Accuracy: 0.9732
```



{} Variables
Terminal

```
plt.show()
```