MNIST Dataset (ANN)

Dataset: MNIST

Problem: Multi-Class Classification

```
import tensorflow
from tensorflow import keras
from tensorflow.keras import Sequential
from tensorflow.keras.layers import Dense,Flatten
```

```
(X_train,y_train),(X_test,y_test) = keras.datasets.mnist.load_data()
```

Training set: 60,000 images (X_train, y_train).

Testing set: 10,000 images (X_test, y_test).

- The split is pre-defined: 60,000 images for training and 10,000 for testing.

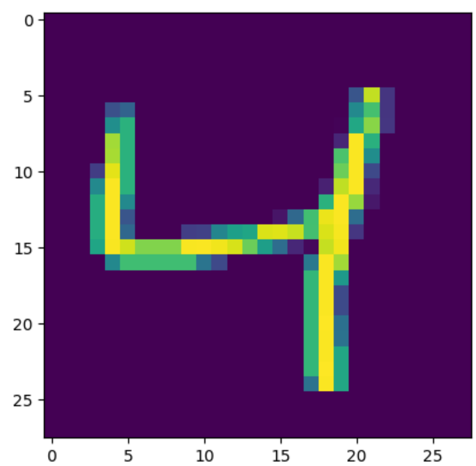
```
X_test.shape

(10000, 28, 28)
```

```
X_train.shape

(60000, 28, 28)
```

```
X_train[0].shape

(28, 28)
```

- Shape of each item is 28 × 28

```
import matplotlib.pyplot as plt
plt.imshow(X_train[2])
```



💡 **Each pixel value is an integer from 0 (black) → 255 (white).**

- Scale these values by dividing each value by 255
- So, 0 will be = 0 & 255 will be= 1

```
X_train = X_train/255
X_test = X_test/255
```

Make an ANN

- First we have to **flatten** the data from **28 × 28 → 784**

`from tensorflow.keras.layers import` `Flatten`

```
model = Sequential()

model.add(Flatten(input_shape=(28,28)))
model.add(Dense(128,activation='relu'))
model.add(Dense(32,activation='relu'))
model.add(Dense(10,activation='softmax')) #softmax for multi-class classifica
tion
```

```
model.summary()
```

Model: "sequential"

Layer (type)	Output Shape	Param #
flatten (Flatten)	(None, 784)	0
dense (Dense)	(None, 128)	100,480
dense_1 (Dense)	(None, 32)	4,128
dense_2 (Dense)	(None, 10)	330

Total params: 104,938 (409.91 KB)

Trainable params: 104,938 (409.91 KB)

Non-trainable params: 0 (0.00 B)

Compile the model

```
model.compile(loss='sparse_categorical_crossentropy',optimizer='Adam',metr
```

```
ics=['accuracy'])
```

- In `sparse` `_categorical_crossentropy`, you don't need to one hot encode the labels.
    - In `categorical_crossentropy` → You have to OHE the labels

Fit

```
history = model.fit(X_train,y_train,epochs=25,validation_split=0.2)
```

👆 Took 1 min 20 sec to train

```
Epoch 1/25
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 5s 2ms/step - accuracy: 0.8559 - loss: 0.5040 - val_accuracy: 0.9578 - val_loss: 0.1430
Epoch 2/25
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.9627 - loss: 0.1269 - val_accuracy: 0.9681 - val_loss: 0.1107
Epoch 3/25
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.9760 - loss: 0.0805 - val_accuracy: 0.9692 - val_loss: 0.1012
Epoch 4/25
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.9802 - loss: 0.0636 - val_accuracy: 0.9711 - val_loss: 0.0948
Epoch 5/25
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.9858 - loss: 0.0464 - val_accuracy: 0.9703 - val_loss: 0.1049
Epoch 6/25
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 3s 2ms/step - accuracy: 0.9882 - loss: 0.0359 - val_accuracy: 0.9750 - val_loss: 0.0894
Epoch 7/25
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 3s 2ms/step - accuracy: 0.9913 - loss: 0.0284 - val_accuracy: 0.9701 - val_loss: 0.1125
Epoch 8/25
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 3s 2ms/step - accuracy: 0.9921 - loss: 0.0232 - val_accuracy: 0.9722 - val_loss: 0.1055
Epoch 9/25
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 3s 2ms/step - accuracy: 0.9924 - loss: 0.0216 - val_accuracy: 0.9750 - val_loss: 0.1035
Epoch 10/25
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 3s 2ms/step - accuracy: 0.9952 - loss: 0.0151 - val_accuracy: 0.9735 - val_loss: 0.1175
Epoch 11/25
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 3s 2ms/step - accuracy: 0.9949 - loss: 0.0165 - val_accuracy: 0.9747 - val_loss: 0.1083
Epoch 12/25
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 3s 2ms/step - accuracy: 0.9954 - loss: 0.0144 - val_accuracy: 0.9758 - val_loss: 0.1185
Epoch 13/25
...
Epoch 24/25
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 3s 2ms/step - accuracy: 0.9970 - loss: 0.0100 - val_accuracy: 0.9772 - val_loss: 0.1528
Epoch 25/25
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 3s 2ms/step - accuracy: 0.9979 - loss: 0.0069 - val_accuracy: 0.9754 - val_loss: 0.1636
Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...
```

Predict

```
y_prob = model.predict(X_test)
```

```
y_prob
0.0s
ay([[1.0056319e-21, 7.7211427e-19, 2.7944839e-18, ..., 1.0000000e+00,
     1.9599515e-16, 2.3558664e-12],
    [4.4258521e-27, 2.2742756e-18, 1.0000000e+00, ..., 3.8952863e-24,
     2.9865945e-18, 4.6176798e-37],
    [5.0889438e-17, 9.9999988e-01, 9.2213494e-11, ..., 2.8155130e-08,
     5.2915265e-08, 2.0391710e-15],
```

- It's very small number

Convert the above values in numbers 0 to 9:

```
y_pred = y_prob.argmax(axis=1)
```

```
array([7, 2, 1, ..., 4, 5, 6], dtype=int64)
```

- `argmax` is a function that finds the index of the biggest number in a list.
- For one row (one image's probabilities), like `[0.05, 0.01, 0.03, 0.02, 0.10, 0.05, 0.04, 0.60, 0.07, 0.03]`:
- The biggest number is `0.60`.
- Its position (index) is `7` (starting from 0: `0, 1, 2, ..., 7`).
- **So, `argmax()` on that list returns `7`.**

What does axis=1 mean?

- Since y_prob is a 2D array (a table), it has two directions (axes)
- `axis=1` tells argmax to look across each **row** (**horizontally**) and find the index of the **biggest number in that row**.

```
y_prob = [
  [0.05, 0.01, 0.03, 0.02, 0.10, 0.05, 0.04, 0.60, 0.07, 0.03],  # Row 1 → max is 0.
```

```
60 at index 7
  [0.80, 0.05, 0.02, 0.01, 0.03, 0.04, 0.02, 0.01, 0.02, 0.02],  # Row 2 → max is 0.80 at index 0
  [0.10, 0.15, 0.70, 0.02, 0.01, 0.01, 0.00, 0.00, 0.01, 0.00]   # Row 3 → max is 0.70 at index 2
]
```

```
[0.05, 0.02, 0.10, 0.70, 0.03, 0.01, 0.02, 0.04, 0.02, 0.01]
```

- There are **10 numbers** because there are 10 possible digits (0–9).
- Each number matches a digit:
  - Index 0 (0.05): 5% chance it's a 0.
  - Index 1 (0.02): 2% chance it's a 1.
  - Index 2 (0.10): 10% chance it's a 2.
  - Index 3 (0.70): 70% chance it's a 3.
  - And so on, up to Index 9 (0.01): 1% chance it's a 9.
- The numbers add up to 1 (100%) because the model splits its confidence across all 10 options.

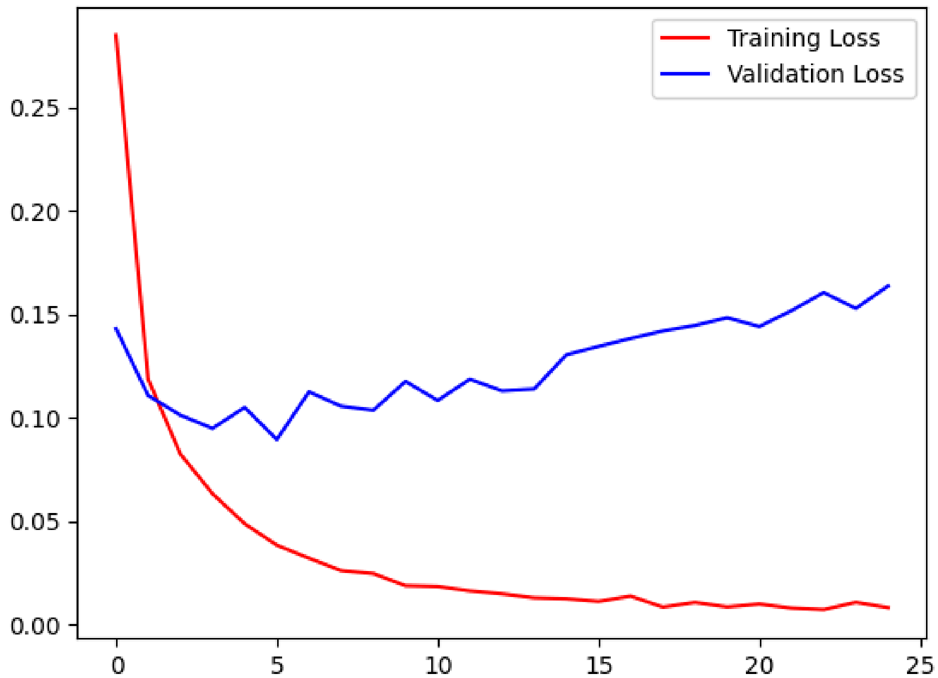
Calculate the Accuracy Score

```
from sklearn.metrics import accuracy_score
accuracy_score(y_test,y_pred)
```

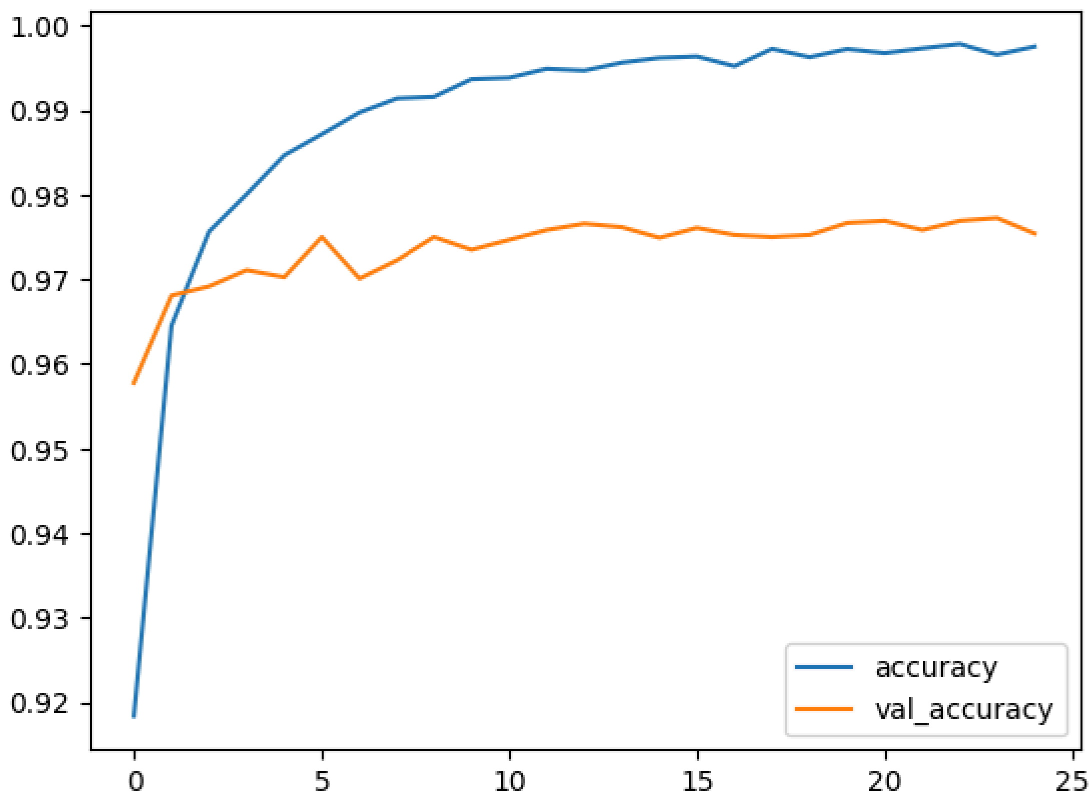
Output: 0.9751

Plot Graph

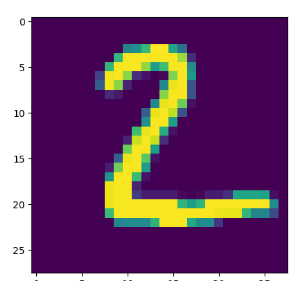
```
plt.plot(history.history['loss'], color='red', label='Training Loss')
plt.plot(history.history['val_loss'], color='blue', label='Validation Loss')
plt.legend()
```




```
plt.plot(history.history['accuracy'], label= 'accuracy')
plt.plot(history.history['val_accuracy'], label= 'val_accuracy')
plt.legend()
```




```
plt.imshow(X_test[1])
```



```
model.predict(X_test[1].reshape(1,28,28)).argmax(axis=1)

Output:
array([2], dtype=int64)
```