```
[ ] Start coding or generate with AI.
```

implementaion Of CNN&ANN model

1. importing the required libraries

```
[5] #importing the required libraries
    from tensorflow.keras.datasets import mnist
    from tensorflow.keras.models import Sequential
    from tensorflow.keras.layers import Conv2D
    from tensorflow.keras.layers import MaxPool2D
    from tensorflow.keras.layers import Flatten
    from tensorflow.keras.layers import Dropout
    from tensorflow.keras.layers import Dense
```

2. loading data

```
[7] (X_train,y_train) , (X_test,y_test)=mnist.load_data()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11490434/11490434 [==============================] - 0s 0us/step
```

3.Preprocessing the Data

```
[3] X_train = X_train / 255
    X_test = X_test / 255
    X_train_flattened = X_train.reshape(len(X_train), 28*28)
    X_test_flattened = X_test.reshape(len(X_test), 28*28)
```

```
[8] from tensorflow import keras
```

4.build a simple ANN model

```
    model = keras.Sequential([
        keras.layers.Flatten(input_shape=(28, 28)),
        keras.layers.Dense(100, activation='relu'),
        keras.layers.Dense(10, activation='sigmoid')
    ])
```

```
[10] model.compile(optimizer='adam',
                   loss='sparse_categorical_crossentropy',
                   metrics=['accuracy'])
     model.fit(X_train, y_train, epochs=10)
```

```
Epoch 1/10
1875/1875 [==============================] - 7s 3ms/step - loss: 0.2784 - accuracy: 0.9216
Epoch 2/10
1875/1875 [==============================] - 5s 3ms/step - loss: 0.1242 - accuracy: 0.9635
Epoch 3/10
1875/1875 [==============================] - 6s 3ms/step - loss: 0.0848 - accuracy: 0.9744
Epoch 4/10
1875/1875 [==============================] - 5s 3ms/step - loss: 0.0644 - accuracy: 0.9808
Epoch 5/10
1875/1875 [==============================] - 6s 3ms/step - loss: 0.0500 - accuracy: 0.9849
Epoch 6/10
1875/1875 [==============================] - 5s 3ms/step - loss: 0.0419 - accuracy: 0.9870
Epoch 7/10
1875/1875 [==============================] - 6s 3ms/step - loss: 0.0337 - accuracy: 0.9892
Epoch 8/10
1875/1875 [==============================] - 5s 3ms/step - loss: 0.0273 - accuracy: 0.9914
Epoch 9/10
1875/1875 [==============================] - 6s 3ms/step - loss: 0.0231 - accuracy: 0.9929
Epoch 10/10
1875/1875 [==============================] - 5s 3ms/step - loss: 0.0189 - accuracy: 0.9943
<keras.src.callbacks.History at 0x78558c772b00>
```

5. Evaluate on ANN model on the test dataset

```
[11] model.evaluate(X_test,y_test)
```

```
313/313 [==============================] - 1s 2ms/step - loss: 0.0886 - accuracy: 0.9749
[0.08858191967010498, 0.9749000072479248]
```

6.Build a simple CNN

[19]
```python
from keras.datasets import mnist
from keras.models import Sequential
from keras.layers import Dense, Dropout, Conv2D, MaxPool2D
```

[25]
```python
len(X_train)
```

```
60000
```

[26]
```python
len(X_test)
```

```
10000
```

[27]
```python
X_train[0].shape
```

```
(28, 28)
```

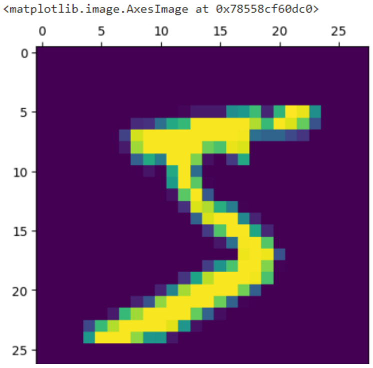
[28]
```python
X_train[0]
```

```
        0.        , 0.        , 0.        , 0.        , 0.18039216,
        0.50980392, 0.71764706, 0.99215686, 0.99215686, 0.81176471,
        0.00784314, 0.        , 0.        , 0.        , 0.        ,
        0.        , 0.        , 0.        ],
       [0.        , 0.        , 0.        , 0.        , 0.        ,
        0.        , 0.        , 0.        , 0.        , 0.        ,
        0.        , 0.        , 0.15294118, 0.58039216, 0.89803922,
        0.99215686, 0.99215686, 0.99215686, 0.98039216, 0.71372549,
        0.        , 0.        , 0.        , 0.        , 0.        ,
        0.        , 0.        , 0.        ],
       [0.        , 0.        , 0.        , 0.        , 0.        ,
        0.        , 0.        , 0.        , 0.        , 0.        ,
        0.09411765, 0.44705882, 0.86666667, 0.99215686, 0.99215686,
        0.99215686, 0.99215686, 0.78823529, 0.30588235, 0.        ,
        0.        , 0.        , 0.        , 0.        , 0.        ,
        0.        , 0.        , 0.        ],
       [0.        , 0.        , 0.        , 0.        , 0.        ,
        0.        , 0.        , 0.        , 0.09019608, 0.25882353,
        0.83529412, 0.99215686, 0.99215686, 0.99215686, 0.99215686,
        0.77647059, 0.31764706, 0.00784314, 0.        , 0.        ,
        0.        , 0.        , 0.        , 0.        , 0.        ,
        0.        , 0.        , 0.        ],
       [0.        , 0.        , 0.        , 0.        , 0.        ,
        0.        , 0.07058824, 0.67058824, 0.85882353, 0.99215686,
        0.99215686, 0.99215686, 0.99215686, 0.76470588, 0.31372549,
        0.03529412, 0.        , 0.        , 0.        , 0.        ,
        0.        , 0.        , 0.        , 0.        , 0.        ,
        0.        , 0.        , 0.        ],
       [0.        , 0.        , 0.        , 0.        , 0.21568627,
        0.6745098 , 0.88627451, 0.99215686, 0.99215686, 0.99215686,
        0.99215686, 0.95686275, 0.52156863, 0.04313725, 0.        ,
        0.        , 0.        , 0.        , 0.        , 0.        ,
        0.        , 0.        , 0.        , 0.        , 0.        ,
        0.        , 0.        , 0.        ],
       [0.        , 0.        , 0.        , 0.        , 0.53333333,
        0.99215686, 0.99215686, 0.99215686, 0.83137255, 0.52941176,
        0.51764706, 0.0627451 , 0.        , 0.        , 0.        ,
        0.        , 0.        , 0.        , 0.        , 0.        ,
        0.        , 0.        , 0.        , 0.        , 0.        ,
        0.        , 0.        , 0.        ],
       [0.        , 0.        , 0.        , 0.        , 0.        ,
        0.        , 0.        , 0.        , 0.        , 0.        ,
        0.        , 0.        , 0.        , 0.        , 0.        ,
        0.        , 0.        , 0.        , 0.        , 0.        ,
        0.        , 0.        , 0.        , 0.        , 0.        ,
        0.        , 0.        , 0.        ],
       [0.        , 0.        , 0.        , 0.        , 0.        ,
        0.        , 0.        , 0.        , 0.        , 0.        ,
        0.        , 0.        , 0.        , 0.        , 0.        ,
        0.        , 0.        , 0.        , 0.        , 0.        ,
        0.        , 0.        , 0.        , 0.        , 0.        ,
        0.        , 0.        , 0.        ],
       [0.        , 0.        , 0.        , 0.        , 0.        ,
        0.        , 0.        , 0.        , 0.        , 0.        ,
        0.        , 0.        , 0.        , 0.        , 0.        ,
        0.        , 0.        , 0.        , 0.        , 0.        ,
        0.        , 0.        , 0.        , 0.        , 0.        ,
        0.        , 0.        , 0.        ]])
```

[29]
```python
import tensorflow as tf
from tensorflow import keras
import matplotlib.pyplot as plt
%matplotlib inline
import numpy as np
```

[30]
```python
plt.matshow(X_train[0])
```

```
<matplotlib.image.AxesImage at 0x78558cf60dc0>
```



[31]
```
y_train[0]
```
```
5
```

[32]
```
X_train = X_train / 255
X_test = X_test / 255
```

[33]
```
X_train[0]
```
```
array([[0.00000000e+00, 0.00000000e+00, 0.00000000e+00, 0.00000000e+00,
        0.00000000e+00, 0.00000000e+00, 0.00000000e+00, 0.00000000e+00],
       [0.00000000e+00, 0.00000000e+00, 0.00000000e+00, 0.00000000e+00,
        0.00000000e+00, 0.00000000e+00, 0.00000000e+00, 0.00000000e+00,
        0.00000000e+00, 0.00000000e+00, 3.69088812e-04, 1.75317186e-03,
        3.39869281e-03, 3.89081123e-03, 3.89081123e-03, 3.89081123e-03,
        3.89081123e-03, 3.09111880e-03, 1.19953864e-03, 0.00000000e+00,
        0.00000000e+00, 0.00000000e+00, 0.00000000e+00, 0.00000000e+00,
        0.00000000e+00, 0.00000000e+00, 0.00000000e+00, 0.00000000e+00],
       [0.00000000e+00, 0.00000000e+00, 0.00000000e+00, 0.00000000e+00,
        0.00000000e+00, 0.00000000e+00, 0.00000000e+00, 0.00000000e+00,
        3.53710111e-04, 1.01499423e-03, 3.27566321e-03, 3.89081123e-03,
        3.89081123e-03, 3.89081123e-03, 3.89081123e-03, 3.04498270e-03,
        1.24567474e-03, 3.07574010e-05, 0.00000000e+00, 0.00000000e+00,
        0.00000000e+00, 0.00000000e+00, 0.00000000e+00, 0.00000000e+00,
        0.00000000e+00, 0.00000000e+00, 0.00000000e+00, 0.00000000e+00],
       [0.00000000e+00, 0.00000000e+00, 0.00000000e+00, 0.00000000e+00,
        0.00000000e+00, 0.00000000e+00, 2.76816609e-04, 2.62975779e-03,
        3.36793541e-03, 3.89081123e-03, 3.89081123e-03, 3.89081123e-03,
        3.89081123e-03, 2.99884660e-03, 1.23029604e-03, 1.38408304e-04,
        0.00000000e+00, 0.00000000e+00, 0.00000000e+00, 0.00000000e+00,
        0.00000000e+00, 0.00000000e+00, 0.00000000e+00, 0.00000000e+00,
        0.00000000e+00, 0.00000000e+00, 0.00000000e+00, 0.00000000e+00],
       [0.00000000e+00, 0.00000000e+00, 0.00000000e+00, 0.00000000e+00,
        8.45828527e-04, 2.64513649e-03, 3.47558631e-03, 3.89081123e-03,
        3.89081123e-03, 3.89081123e-03, 3.89081123e-03, 3.75240292e-03,
        2.04536717e-03, 1.69165705e-04, 0.00000000e+00, 0.00000000e+00,
        0.00000000e+00, 0.00000000e+00, 0.00000000e+00, 0.00000000e+00,
        0.00000000e+00, 0.00000000e+00, 0.00000000e+00, 0.00000000e+00,
        0.00000000e+00, 0.00000000e+00, 0.00000000e+00, 0.00000000e+00],
       [0.00000000e+00, 0.00000000e+00, 0.00000000e+00, 0.00000000e+00,
        2.09150327e-03, 3.89081123e-03, 3.89081123e-03, 3.89081123e-03,
        3.26028451e-03, 2.07612457e-03, 2.02998847e-03, 2.46059208e-04,
        0.00000000e+00, 0.00000000e+00, 0.00000000e+00, 0.00000000e+00,
        0.00000000e+00, 0.00000000e+00, 0.00000000e+00, 0.00000000e+00,
        0.00000000e+00, 0.00000000e+00, 0.00000000e+00, 0.00000000e+00,
        0.00000000e+00, 0.00000000e+00, 0.00000000e+00, 0.00000000e+00],
       [0.00000000e+00, 0.00000000e+00, 0.00000000e+00, 0.00000000e+00,
        0.00000000e+00, 0.00000000e+00, 0.00000000e+00, 0.00000000e+00,
        0.00000000e+00, 0.00000000e+00, 0.00000000e+00, 0.00000000e+00,
        0.00000000e+00, 0.00000000e+00, 0.00000000e+00, 0.00000000e+00,
        0.00000000e+00, 0.00000000e+00, 0.00000000e+00, 0.00000000e+00,
        0.00000000e+00, 0.00000000e+00, 0.00000000e+00, 0.00000000e+00,
        0.00000000e+00, 0.00000000e+00, 0.00000000e+00, 0.00000000e+00],
       [0.00000000e+00, 0.00000000e+00, 0.00000000e+00, 0.00000000e+00,
        0.00000000e+00, 0.00000000e+00, 0.00000000e+00, 0.00000000e+00,
        0.00000000e+00, 0.00000000e+00, 0.00000000e+00, 0.00000000e+00,
        0.00000000e+00, 0.00000000e+00, 0.00000000e+00, 0.00000000e+00,
        0.00000000e+00, 0.00000000e+00, 0.00000000e+00, 0.00000000e+00,
        0.00000000e+00, 0.00000000e+00, 0.00000000e+00, 0.00000000e+00,
        0.00000000e+00, 0.00000000e+00, 0.00000000e+00, 0.00000000e+00,
        0.00000000e+00, 0.00000000e+00, 0.00000000e+00, 0.00000000e+00]])
```

[34]
```
X_train_flattened = X_train.reshape(len(X_train), 28*28)
X_test_flattened = X_test.reshape(len(X_test), 28*28)
```

[35]
```
X_train_flattened.shape
```
```
(60000, 784)
```

Very simple neural network with no hidden layers

[36]
```
model = keras.Sequential([
    keras.layers.Dense(10, input_shape=(784,), activation='sigmoid')
])

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

model.fit(X_train_flattened, y_train, epochs=5)
```
```
Epoch 1/5
1875/1875 [==============================] - 4s 2ms/step - loss: 2.1876 - accuracy: 0.5972
Epoch 2/5
1875/1875 [==============================] - 3s 2ms/step - loss: 1.9765 - accuracy: 0.7033
Epoch 3/5
1875/1875 [==============================] - 5s 3ms/step - loss: 1.7902 - accuracy: 0.7385
Epoch 4/5
1875/1875 [==============================] - 4s 2ms/step - loss: 1.6263 - accuracy: 0.7613
Epoch 5/5
1875/1875 [==============================] - 3s 2ms/step - loss: 1.4836 - accuracy: 0.7752
<keras.src.callbacks.History at 0x785563b98490>
```

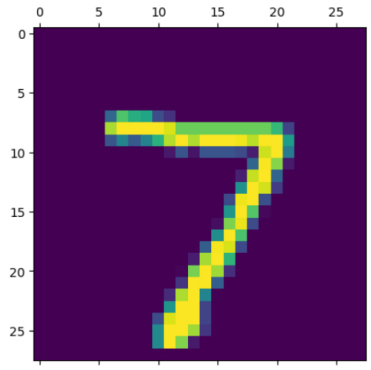
[37]
```
model.evaluate(X_test_flattened, y_test)
```
```
313/313 [==============================] - 1s 1ms/step - loss: 1.4038 - accuracy: 0.8034
[1.4038022756576538, 0.8033998800682068]
```

[38]
```
y_predicted = model.predict(X_test_flattened)
y_predicted[0]
```

```
313/313 [==============================] - 0s 1ms/step
array([0.31793118, 0.4008361 , 0.3571871 , 0.40740892, 0.47357666,
       0.38162598, 0.28726527, 0.80860776, 0.37566516, 0.58500075],
      dtype=float32)
```

[39]
```
plt.matshow(X_test[0])
```

```
<matplotlib.image.AxesImage at 0x785563acb490>
```



[40]
```
np.argmax(y_predicted[0])
```

```
7
```

[41]
```
y_predicted_labels = [np.argmax(i) for i in y_predicted]
y_predicted_labels[:5]
```

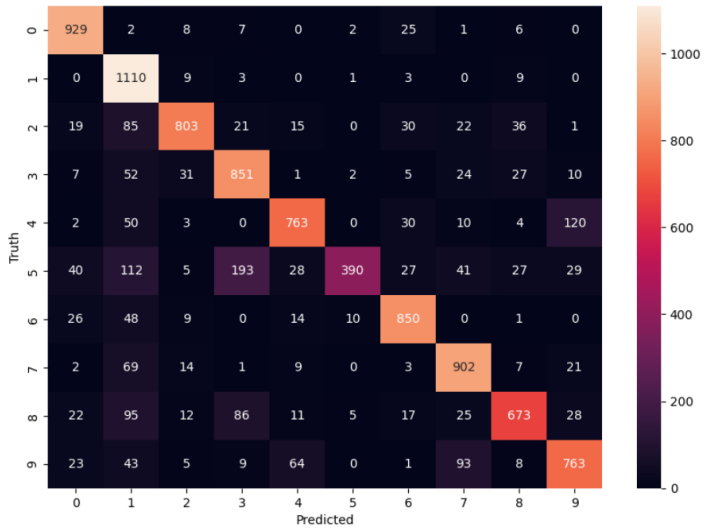
```
[7, 2, 1, 0, 4]
```

[42]
```
cm = tf.math.confusion_matrix(labels=y_test,predictions=y_predicted_labels)
cm
```

```
<tf.Tensor: shape=(10, 10), dtype=int32, numpy=
array([[ 929,    2,    8,    7,    0,    2,   25,    1,    6,    0],
       [   0, 1110,    9,    3,    0,    1,    3,    0,    9,    0],
       [  19,   85,  803,   21,   15,    0,   30,   22,   36,    1],
       [   7,   52,   31,  851,    1,    2,    5,   24,   27,   10],
       [   2,   50,    3,    0,  763,    0,   30,   10,    4,  120],
       [  40,  112,    5,  193,   28,  390,   27,   41,   27,   29],
       [  26,   48,    9,    0,   14,   10,  850,    0,    1,    0],
       [   2,   69,   14,    1,    9,    0,    3,  902,    7,   21],
       [  22,   95,   12,   86,   11,    5,   17,   25,  673,   28],
       [  23,   43,    5,    9,   64,    0,    1,   93,    8,  763]],
      dtype=int32)>
```

[43]
```
import seaborn as sn
plt.figure(figsize = (10,7))
sn.heatmap(cm, annot=True, fmt='d')
plt.xlabel('Predicted')
plt.ylabel('Truth')
```

```
Text(95.72222222222221, 0.5, 'Truth')
```



[44]
```
model = keras.Sequential([
    keras.layers.Dense(100, input_shape=(784,), activation='relu'),
```

```
    keras.layers.Dense(10, activation='sigmoid')
])

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

model.fit(X_train_flattened, y_train, epochs=5)
```

```
Epoch 1/5
1875/1875 [==============================] - 10s 5ms/step - loss: 1.2528 - accuracy: 0.6968
Epoch 2/5
1875/1875 [==============================] - 11s 6ms/step - loss: 0.5294 - accuracy: 0.8651
Epoch 3/5
1875/1875 [==============================] - 8s 4ms/step - loss: 0.4059 - accuracy: 0.8900
Epoch 4/5
1875/1875 [==============================] - 11s 6ms/step - loss: 0.3571 - accuracy: 0.9000
Epoch 5/5
1875/1875 [==============================] - 8s 4ms/step - loss: 0.3304 - accuracy: 0.9070
<keras.src.callbacks.History at 0x785562c964a0>
```

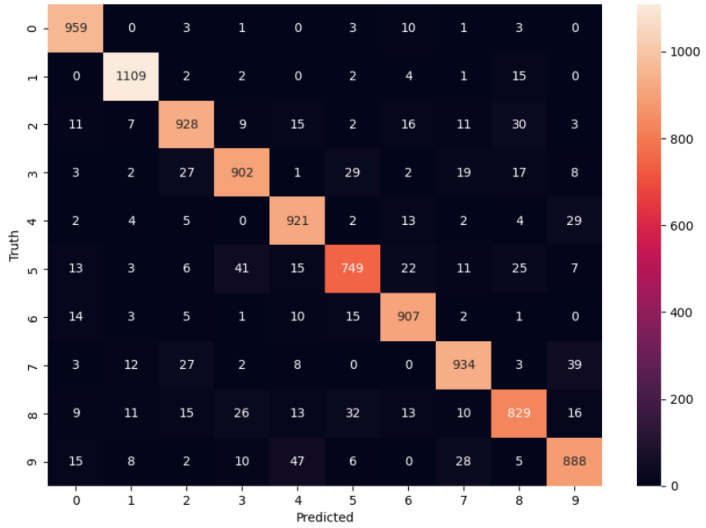
[45]
```
model.evaluate(X_test_flattened,y_test)
```

```
313/313 [==============================] - 1s 4ms/step - loss: 0.3096 - accuracy: 0.9126
[0.3096467852592468, 0.9125999808311462]
```

[46]
```
y_predicted = model.predict(X_test_flattened)
y_predicted_labels = [np.argmax(i) for i in y_predicted]
cm = tf.math.confusion_matrix(labels=y_test,predictions=y_predicted_labels)

plt.figure(figsize = (10,7))
sn.heatmap(cm, annot=True, fmt='d')
plt.xlabel('Predicted')
plt.ylabel('Truth')
```

```
313/313 [==============================] - 1s 2ms/step
Text(95.72222222222221, 0.5, 'Truth')
```



Using Flatten layer so that we don't have to call .reshape on input dataset

[47]
```
model = keras.Sequential([
    keras.layers.Flatten(input_shape=(28, 28)),
    keras.layers.Dense(100, activation='relu'),
    keras.layers.Dense(10, activation='sigmoid')
])

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

model.fit(X_train, y_train, epochs=10)
```

```
Epoch 1/10
1875/1875 [==============================] - 6s 3ms/step - loss: 1.2202 - accuracy: 0.7136
Epoch 2/10
1875/1875 [==============================] - 6s 3ms/step - loss: 0.5115 - accuracy: 0.8696
Epoch 3/10
1875/1875 [==============================] - 5s 3ms/step - loss: 0.3950 - accuracy: 0.8926
Epoch 4/10
1875/1875 [==============================] - 7s 4ms/step - loss: 0.3493 - accuracy: 0.9022
Epoch 5/10
1875/1875 [==============================] - 5s 3ms/step - loss: 0.3234 - accuracy: 0.9079
Epoch 6/10
1875/1875 [==============================] - 7s 4ms/step - loss: 0.3055 - accuracy: 0.9123
```

```
Epoch 7/10
1875/1875 [==============================] - 9s 5ms/step - loss: 0.2924 - accuracy: 0.9153
Epoch 8/10
1875/1875 [==============================] - 9s 5ms/step - loss: 0.2806 - accuracy: 0.9189
Epoch 9/10
1875/1875 [==============================] - 8s 5ms/step - loss: 0.2709 - accuracy: 0.9221
Epoch 10/10
1875/1875 [==============================] - 5s 3ms/step - loss: 0.2616 - accuracy: 0.9245
<keras.src.callbacks.History at 0x785572f085e0>
```

Now let us build a convolutional neural network to train our images

[48]
```
from tensorflow.keras import datasets, layers, models
cnn = models.Sequential([
    layers.Conv2D(filters=32, kernel_size=(3, 3), activation='relu', input_shape=(28,28,1)),
    layers.MaxPooling2D((2, 2)),

    layers.Conv2D(filters=64, kernel_size=(3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),

    layers.Flatten(),
    layers.Dense(64, activation='relu'),
    layers.Dense(10, activation='softmax')
])
```

[49]
```
cnn.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
```

[50]
```
X_train.shape
```

```
(60000, 28, 28)
```

```
cnn.fit(X_train, y_train, epochs=5)
```

```
Epoch 1/5
1875/1875 [==============================] - 47s 25ms/step - loss: 0.7859 - accuracy: 0.7370
Epoch 2/5
1875/1875 [==============================] - 43s 23ms/step - loss: 0.2747 - accuracy: 0.9171
Epoch 3/5
1875/1875 [==============================] - 48s 26ms/step - loss: 0.1762 - accuracy: 0.9477
Epoch 4/5
1875/1875 [==============================] - 43s 23ms/step - loss: 0.1301 - accuracy: 0.9606
Epoch 5/5
1875/1875 [==============================] - 44s 23ms/step - loss: 0.1087 - accuracy: 0.9674
<keras.src.callbacks.History at 0x785572e16e00>
```

```
cnn.evaluate(X_test,y_test)
```

```
313/313 [==============================] - 2s 7ms/step - loss: 0.0982 - accuracy: 0.9702
[0.0981912687420845, 0.9702000021934509]
```

While we have considered a simple dataset and still we can see that even after ANN was trained for 10 epochs and CNN for 5 epochs, the evaluation of CNN is better than ANN.