1. Import dependencies:

In [ ]:
```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import cv2
#from google.colab.patches import cv2_imshow
from PIL import Image
import tensorflow as tf
tf.random.set_seed(3)
from tensorflow import keras
from keras.datasets import mnist
from tensorflow.math import confusion_matrix
```

2. Loading the MNIST data from keras:

A- Creating the variables of testing and training:

In [ ]:
```
(X_train , Y_train) , (X_test, Y_test) = mnist.load_data()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-dataset
s/mnist.npz
11490434/11490434 [==============================] - 6s 0us/step
```

B- Type of the datas:

In [ ]:
```
type(X_train)
```

Out[ ]:
```
numpy.ndarray
```

C- The dimensions of the data:

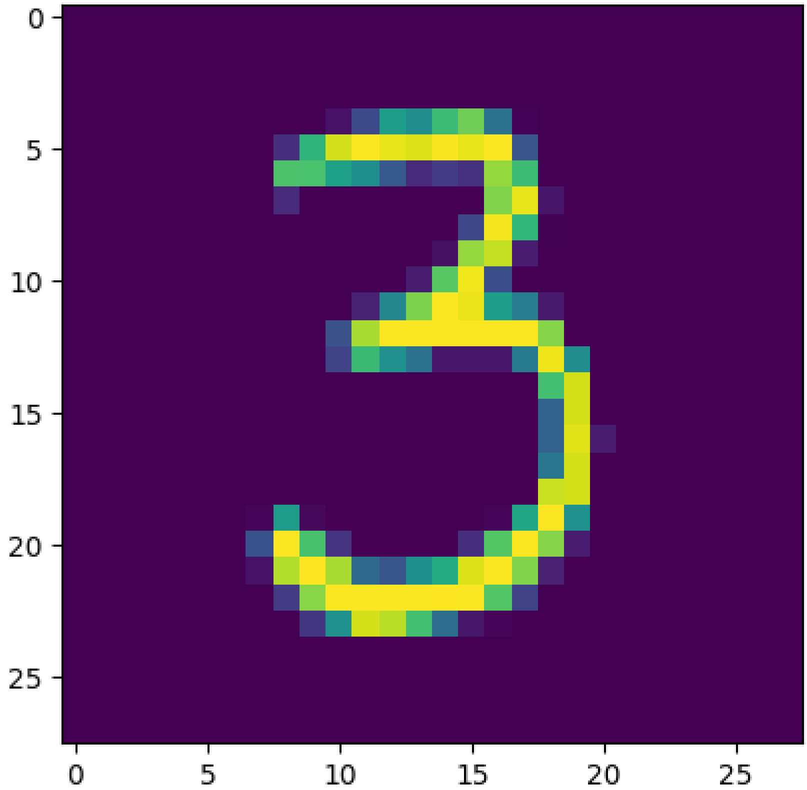
In [ ]:
```
print(X_train.shape) #6000 images with 28*28 px (it's a gray image)
print(Y_train.shape)
print(X_test.shape) #1000 images with 28*28 px (it's a gray image)
print(Y_test.shape)
```

```
(60000, 28, 28)
(60000,)
(10000, 28, 28)
(10000,)
```

2. Visualisation of our image:

A- Displaying an example:

In [ ]:
```
plt.imshow(X_train[50])
plt.show()
print(Y_train[50])
```



3

B- The value that the systeme will be predict:

```
In [ ]: print(np.unique(Y_train))
```

```
[0 1 2 3 4 5 6 7 8 9]
```

3. Normalisation (all value should be in range 0 & 1):

A- Scalling the value:

```
In [ ]: X_train = X_train / 255
        X_test = X_test / 255
```

4. Building the Neural Network:

A- Setting some layers of the Neural Network:

```
In [ ]: model = keras.Sequential([
            keras.layers.Flatten(input_shape = (28, 28)),
            keras.layers.Dense(50, activation = 'relu'),
            keras.layers.Dense(50, activation = 'relu'), #You can any value
            keras.layers.Dense(10, activation = 'sigmoid') #10 -> number of classes outp
        ])
```

B- Compiling the neural Network:

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

```
WARNING:tensorflow:From c:\Users\HP\AppData\Local\Programs\Python\Python39\lib\si
te-packages\keras\src\optimizers\__init__.py:309: The name tf.train.Optimizer is
deprecated. Please use tf.compat.v1.train.Optimizer instead.
```

C- Training our Neural network:

In [ ]:
```
model.fit(X_train, Y_train , epochs= 10)
```

```
Epoch 1/10
WARNING:tensorflow:From c:\Users\HP\AppData\Local\Programs\Python\Python39\lib\si
te-packages\keras\src\utils\tf_utils.py:492: The name tf.ragged.RaggedTensorValue
is deprecated. Please use tf.compat.v1.ragged.RaggedTensorValue instead.

WARNING:tensorflow:From c:\Users\HP\AppData\Local\Programs\Python\Python39\lib\si
te-packages\keras\src\engine\base_layer_utils.py:384: The name tf.executing_eager
ly_outside_functions is deprecated. Please use tf.compat.v1.executing_eagerly_out
side_functions instead.

1875/1875 [==============================] - 5s 2ms/step - loss: 0.2907 - accurac
y: 0.9160
Epoch 2/10
1875/1875 [==============================] - 4s 2ms/step - loss: 0.1307 - accurac
y: 0.9601
Epoch 3/10
1875/1875 [==============================] - 4s 2ms/step - loss: 0.0950 - accurac
y: 0.9708
Epoch 4/10
1875/1875 [==============================] - 4s 2ms/step - loss: 0.0763 - accurac
y: 0.9774
Epoch 5/10
1875/1875 [==============================] - 4s 2ms/step - loss: 0.0627 - accurac
y: 0.9804
Epoch 6/10
1875/1875 [==============================] - 4s 2ms/step - loss: 0.0543 - accurac
y: 0.9831
Epoch 7/10
1875/1875 [==============================] - 4s 2ms/step - loss: 0.0460 - accurac
y: 0.9855
Epoch 8/10
1875/1875 [==============================] - 4s 2ms/step - loss: 0.0404 - accurac
y: 0.9869
Epoch 9/10
1875/1875 [==============================] - 4s 2ms/step - loss: 0.0352 - accurac
y: 0.9884
Epoch 10/10
1875/1875 [==============================] - 4s 2ms/step - loss: 0.0318 - accurac
y: 0.9896
```

Out[ ]:
```
<keras.src.callbacks.History at 0x16523b6b820>
```

D- Accuracy of the test data:

In [ ]:
```
loss , accuracy = model.evaluate(X_test , Y_test)
print(loss)
print(accuracy)
```

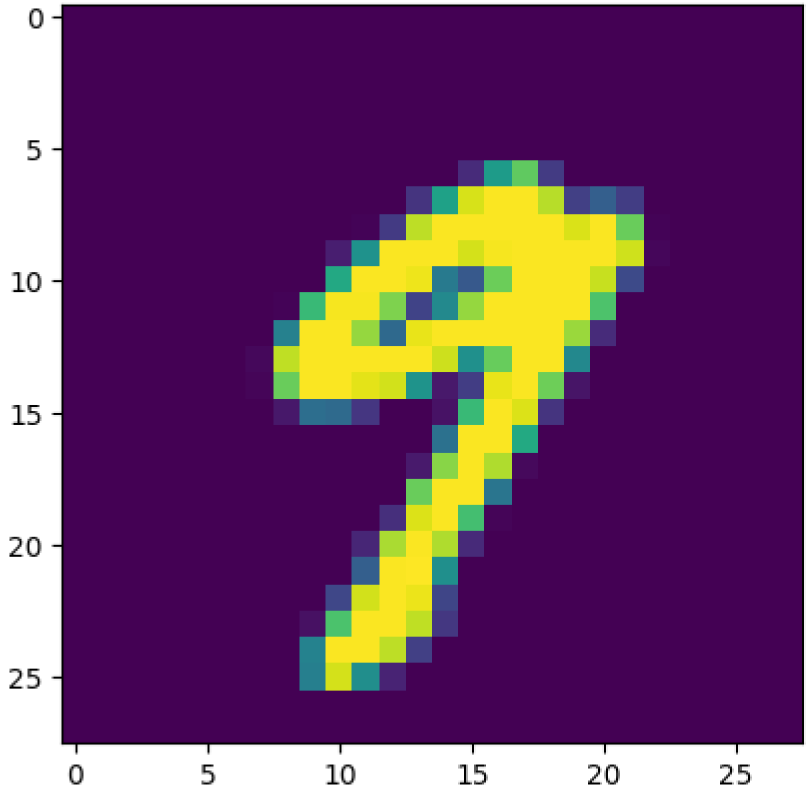
```
313/313 [==============================] - 1s 3ms/step - loss: 0.1093 - accuracy:
0.9711
0.10926541686058044
0.9710999727249146
```

E- Example:

In [ ]:
```
Y_pred = model.predict(X_test)
```

```
313/313 [==============================] - 1s 3ms/step
```

In [ ]:
```
plt.imshow(X_test[20])
plt.show()
```



In [ ]:
```
print("The value is:", np.argmax(Y_pred[20]))
```

```
The value is: 9
```

F- All the prediction of the model:

In [ ]:
```
Y_pred_value = [np.argmax(i) for i in Y_pred]
print(Y_pred_value[47]) #Exemple
```

2

5. Confusion Matrix:

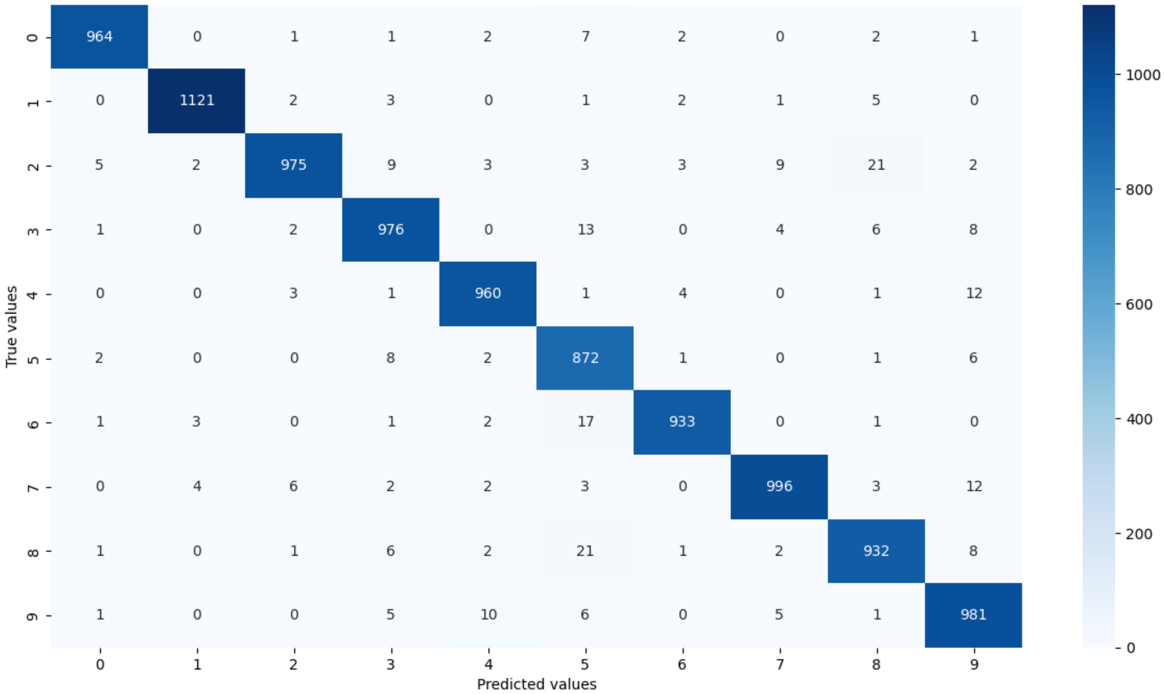
A- Creating the matrix:

In [ ]:
```
M = confusion_matrix(Y_test , Y_pred_value) #True value vs predictin value
```

B- Construction of the heatmap:

In [ ]:
```
plt.figure(figsize= (15, 8))
sns.heatmap(M, annot=True, fmt='d', cmap='Blues')
plt.ylabel('True values')
plt.xlabel('Predicted values')
```

Out[ ]:
```
Text(0.5, 58.7222222222222, 'Predicted values')
```

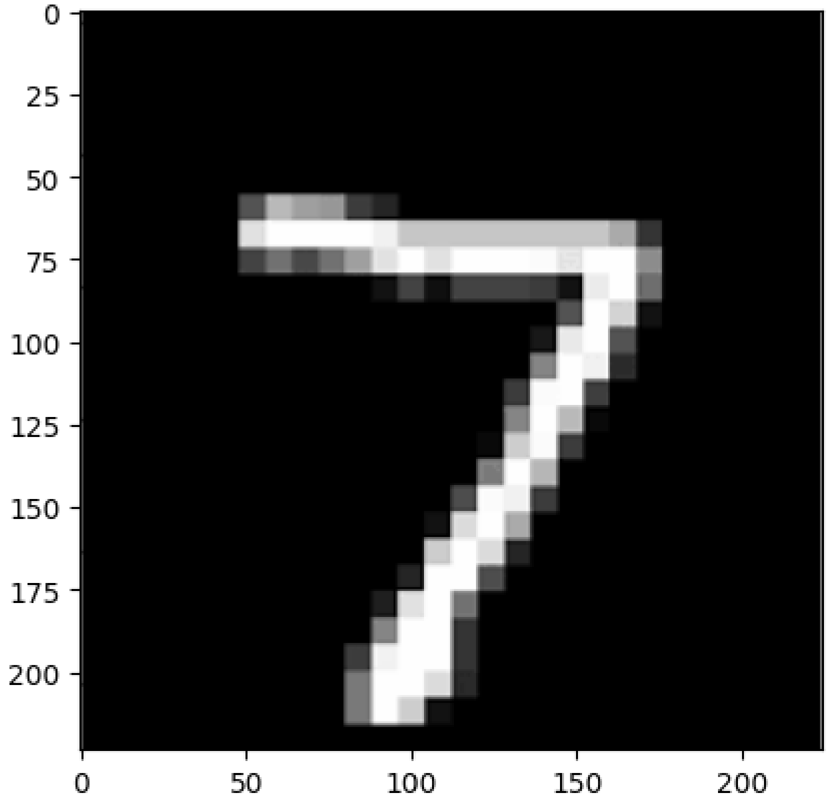


6. Example:

In [ ]:
```
def mnistPredictionPIL(path):
    input_image = Image.open(path)
    # Convertir l'image en niveaux de gris
    image_gray = input_image.convert('L')
    # Redimensionner l'image à 28x28 pixels
    image_resize = image_gray.resize((28, 28))
    # Convertir l'image PIL en un array numpy et normaliser les pixels
    image_np = np.array(image_resize) / 255.0
    image_reshaped = np.reshape(image_np, [1, 28, 28])
    # Prédire le chiffre
    predict = model.predict(image_reshaped)
    predictNum = np.argmax(predict)
    # Afficher l'image originale
    plt.imshow(input_image)
    plt.show()
    print('The predicted number is:', predictNum)

# Exemple d'utilisation
path = 'C:/Deep_learning Python/projects/MNIST Digit Classification/télécharger.
mnistPredictionPIL(path)
```

```
1/1 [==============================] - 0s 51ms/step
```




```
The predicted number is: 7
```