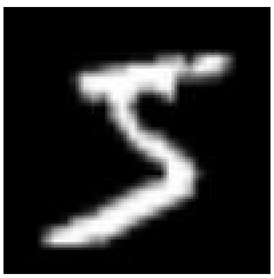
1.Dataset de imagenes MNIST

```
import tensorflow as tf
import numpy as np
import cv2
from google.colab.patches import cv2_imshow

mnist = tf.keras.datasets.mnist

(x_train,y_train),(x_test,y_test)=mnist.load_data()

x_train.shape

(60000, 28, 28)

cv2_imshow(cv2.resize(x_train[0],(100,100)))
```



```
y_train[0]

5

x_test.shape

(10000, 28, 28)
```

\2.Dataset de caracteristicas con hog features

```
img=x_train[1]
def get_hog():
  winSize=img.shape
  blockSize=(8,8)
  blockStride=(2,2)
  cellSize=(4,4)
  nbins=9
  hog=cv2.HOGDescriptor(winSize,blockSize,blockStride,cellSize,nbins)
  return hog
hog=get_hog()
hog.compute(img).shape

(4356,)
```

Calcular para cada imagen

```
def get_feature_dataset(x):
  features=[]

  for img in x:
    features.append(hog.compute(img))
  features=np.array(features)
  return features

features_train=get_feature_dataset(x_train)
features_test=get_feature_dataset(x_test)

features_train.shape

(60000, 4356)

features_test.shape

(10000, 4356)
```

3.Definicion e implementacion de la red neuronal +onehot labels

```
y_trainOneHot = tf.one_hot(y_train,np.max(y_train)+1)
y_testOneHot = tf.one_hot(y_test,np.max(y_train)+1)

y_trainOneHot[1]

<tf.Tensor: shape=(10,), dtype=float32, numpy=array([1., 0., 0., 0.,
0., 0., 0., 0., 0., 0.], dtype=float32)>

from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Input
import tensorflow as tf

def classifier(features_train):
    model = Sequential()

    # Define la forma de entrada usando Input
    model.add(Input(shape=(features_train.shape[1],)))
    model.add(Dense(200, activation='relu'))
    model.add(Dense(180, activation='relu'))
    model.add(Dense(150, activation='relu'))
    model.add(Dense(10, activation='softmax'))

    # Compila el modelo con el argumento correcto
    model.compile(
        loss='categorical_crossentropy',
        optimizer=tf.keras.optimizers.SGD(learning_rate=1e-3)
    )

    return model
```

```python
# Ejemplo de uso (asegúrate de que features_train esté definido)
model = classifier(features_train)
model.summary()
```

```
Model: "sequential_1"

┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳
━━━━━━━━━━━━━━━━━┓
┃ Layer (type)                          ┃ Output Shape                 ┃
Param # ┃
┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇
━━━━━━━━━━━━━━━━━┩
│ dense_4 (Dense)                       │ (None, 200)                  │
871,400 │
├───────────────────────────────────────┼──────────────────────────────┼
─────────────────┤
│ dense_5 (Dense)                       │ (None, 180)                  │
36,180 │
├───────────────────────────────────────┼──────────────────────────────┼
─────────────────┤
│ dense_6 (Dense)                       │ (None, 150)                  │
27,150 │
├───────────────────────────────────────┼──────────────────────────────┼
─────────────────┤
│ dense_7 (Dense)                       │ (None, 10)                   │
1,510 │
└───────────────────────────────────────┴──────────────────────────────┴
─────────────────┘

 Total params: 936,240 (3.57 MB)

 Trainable params: 936,240 (3.57 MB)

 Non-trainable params: 0 (0.00 B)
```

4.Calculo de la matrix de confusion

```python
#Epocas, cuantas veces se le pasa el dataset, batch es cuantas
muestras se van a tomar para calcular el gradiente, en este caso 100
imagenes para calcular el gradiente solo apra esas y actualizar los
pesos.
#si hay datasets de teras, una forma de no cargar todo en memoria, es
cargar pedazos osea batches, calculamos, y asi de manera secuencial.
model.fit(features_train,y_trainOneHot,epochs=10, batch_size=100)
```

```
Epoch 1/10
600/600 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - loss: 2.2469
Epoch 2/10
600/600 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - loss: 1.9716
Epoch 3/10
```

```
600/600 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - loss: 1.4719
Epoch 4/10
600/600 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - loss: 0.9165
Epoch 5/10
600/600 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - loss: 0.5784
Epoch 6/10
600/600 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - loss: 0.4077
Epoch 7/10
600/600 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - loss: 0.3193
Epoch 8/10
600/600 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - loss: 0.2684
Epoch 9/10
600/600 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - loss: 0.2314
Epoch 10/10
600/600 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - loss: 0.2055
```

```
<keras.src.callbacks.history.History at 0x780dfe95d450>
```

```python
#Evaluar el sistema
prediction_train=model.predict(features_train)
prediction_test=model.predict(features_test)
```

```
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 3s 2ms/step
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step
```

```python
prediction_train.shape
```

```
(60000, 10)
```

```python
prediction_train[0]
```

```
array([1.6841936e-04, 1.7363981e-04, 1.3027936e-03, 4.7962430e-01,
       4.6220011e-04, 5.0014567e-01, 1.2696250e-03, 7.0529588e-04,
       1.5862295e-02, 2.8578119e-04], dtype=float32)
```

```python
y_pred_train=np.argmax(prediction_train,1)
y_pred_test=np.argmax(prediction_test,1)
```

```python
y_pred_train
```

```
array([5, 0, 4, ..., 5, 6, 8])
```

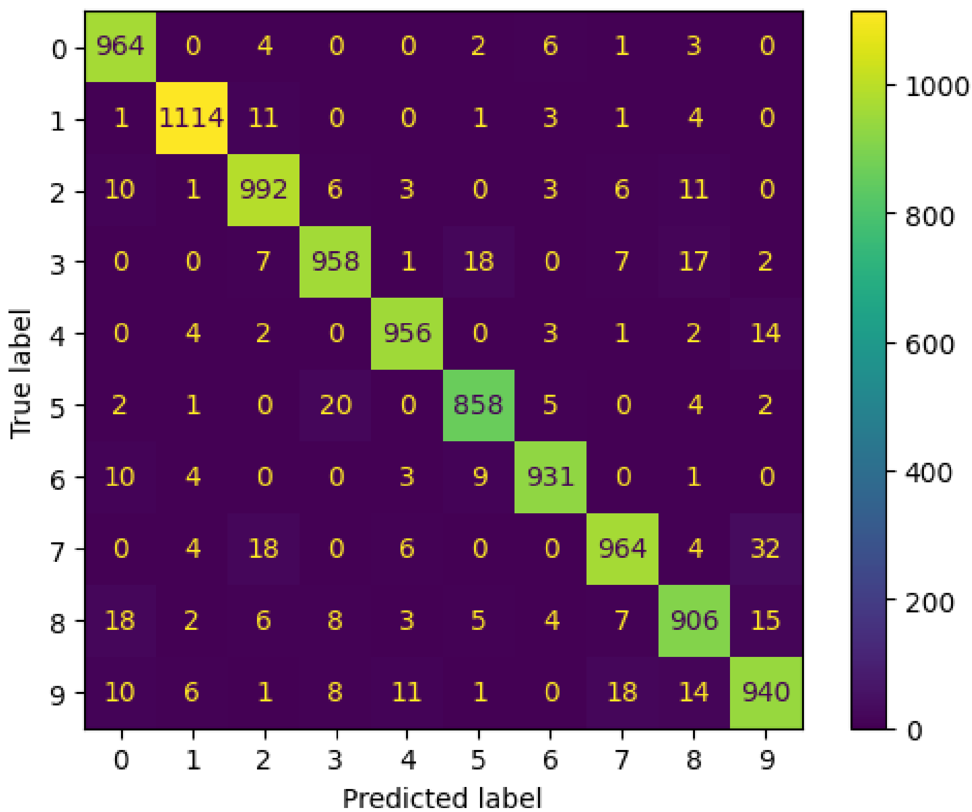
```python
errorTrain= 100*np.sum(y_pred_train!=y_train)/len(y_train)
errorTest= 100*np.sum(y_pred_test!=y_test)/len(y_test)
print("Error de entrenamiento:")
print(errorTrain,"%")
print("Error de prueba:")
print(errorTest,"%")
```

```
Error de entrenamiento:
4.466666666666667 %
```

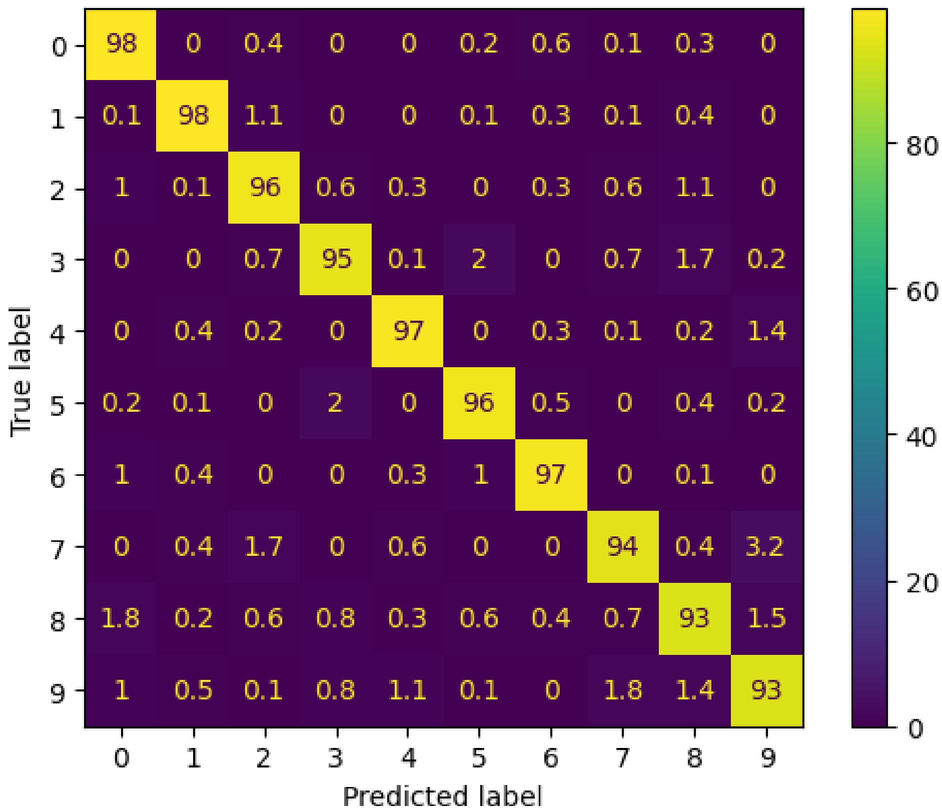
```
Error de prueba:
4.17 %
```

4:Matriz de confusion

```
#la matriz de confusion nos indica visualmente cuantos aciertos y
fracasos se tienen con el dataset de prueba
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay

conf_mat=confusion_matrix(y_test,y_pred_test)
conf_mat
disp=ConfusionMatrixDisplay(confusion_matrix=conf_mat)
disp.plot()

<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at
0x780d36726980>
```




```
conf_mat_norm = np.round(100*conf_mat/np.sum(conf_mat,1),1)

disp2=ConfusionMatrixDisplay(confusion_matrix=conf_mat_norm)
disp2.plot()

<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at
0x780e01f9d780>
```



True label
Predicted label
0 1 2 3 4 5 6 7 8 9
98 0 0.4 0 0 0.2 0.6 0.1 0.3 0
0.1 98 1.1 0 0 0.1 0.3 0.1 0.4 0
1 0.1 96 0.6 0.3 0 0.3 0.6 1.1 0
0 0 0.7 95 0.1 2 0 0.7 1.7 0.2
0 0.4 0.2 0 97 0 0.3 0.1 0.2 1.4
0.2 0.1 0 2 0 96 0.5 0 0.4 0.2
1 0.4 0 0 0.3 1 97 0 0.1 0
0 0.4 1.7 0 0.6 0 0 94 0.4 3.2
1.8 0.2 0.6 0.8 0.3 0.6 0.4 0.7 93 1.5
1 0.5 0.1 0.8 1.1 0.1 0 1.8 1.4 93
0 20 40 60 80