Practical Introduction to Deep Learning Basics

```
import tensorflow as tf
from tensorflow import keras

import numpy as np
import matplotlib.pyplot as plt

print(tf.__version__)
```

```
2.18.0
```

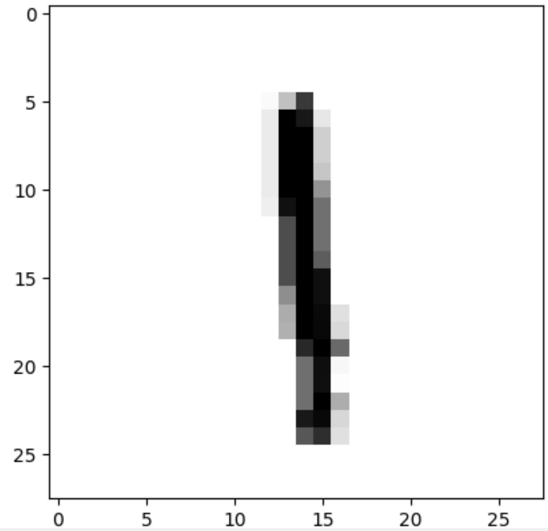
Load Data

```
mnist = tf.keras.datasets.mnist
(x_train, y_train), (x_test, y_test) = mnist.load_data()
#(x_train, y_train), (x_test, y_test) = mnist.load_data(path='/gpfs/projects/nct00/nct00002/basics-utils/mnist.npz')
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11490434/11490434 ━━━━━━━━━━━━━━━━━━━━ 2s 0us/step
```

```
import matplotlib.pyplot as plt
plt.imshow(x_train[8], cmap=plt.cm.binary)
```

```
<matplotlib.image.AxesImage at 0x7e82fbbb3e50>
```




```
print(y_train[8])
```

```
1
```

```
print(x_train.ndim)
```

```
3
```

```
print(x_train.shape)
```

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

```
print(x_train.dtype)
```

```
uint8
```

Prepare data

```
x_train = x_train.astype('float32')
x_test = x_test.astype('float32')
```

```
x_train /= 255
x_test /= 255
```

```
x_train = x_train.reshape(60000, 784)
x_test = x_test.reshape(10000, 784)
```

```
print(x_train.shape)
print(x_test.shape)
```

```
(60000, 784)
(10000, 784)
```

```
from tensorflow.keras.utils import to_categorical
```

```
print(y_test[0])
```

```
7
```

```
print(y_train[0])
```

```
5
```

```
print(y_train.shape)
```

```
(60000,)
```

```
print(x_test.shape)
```

```
(10000, 784)
```

```
y_train = to_categorical(y_train, num_classes=10)
y_test = to_categorical(y_test, num_classes=10)


print(y_test[0])
```

```
[0. 0. 0. 0. 0. 0. 0. 1. 0. 0.]
```

```
print(y_train[0])
```

```
[0. 0. 0. 0. 0. 1. 0. 0. 0. 0.]
```

```
print(y_train.shape)
```

```
(60000, 10)
```

```
print(y_test.shape)
```

```
(10000, 10)
```

Define Model

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

model = Sequential()
model.add(Input(shape=(784,)))  # Define the input shape here
model.add(Dense(10, activation='sigmoid'))
model.add(Dense(10, activation='softmax'))
```

Se muestra un resumen del modelo tal como se tiene tras las instrucciones anteriores

```
model.summary()
```

```
Model: "sequential"
```

Layer (type)	Output Shape	Param #
dense (Dense)	(None, 10)	7,850
dense_1 (Dense)	(None, 10)	110

```
Total params: 7,960 (31.09 KB)
Trainable params: 7,960 (31.09 KB)
```

Compile model (configuration)

```
model.compile(loss="categorical_crossentropy",
              optimizer="sgd",
              metrics = ['accuracy'])
```

Training the model

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

```
Epoch 1/5
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 5s 2ms/step - accuracy: 0.3374 - loss: 2.1362
Epoch 2/5
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 5s 3ms/step - accuracy: 0.6992 - loss: 1.4774
Epoch 3/5
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.7957 - loss: 1.0544
Epoch 4/5
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.8322 - loss: 0.8225
Epoch 5/5
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 5s 3ms/step - accuracy: 0.8518 - loss: 0.6885
<keras.src.callbacks.history.History at 0x7e82f6369550>
```

Evaluation the model

```
test_loss, test_acc = model.evaluate(x_test, y_test)
```

```
313/313 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.8540 - loss: 0.6551
```

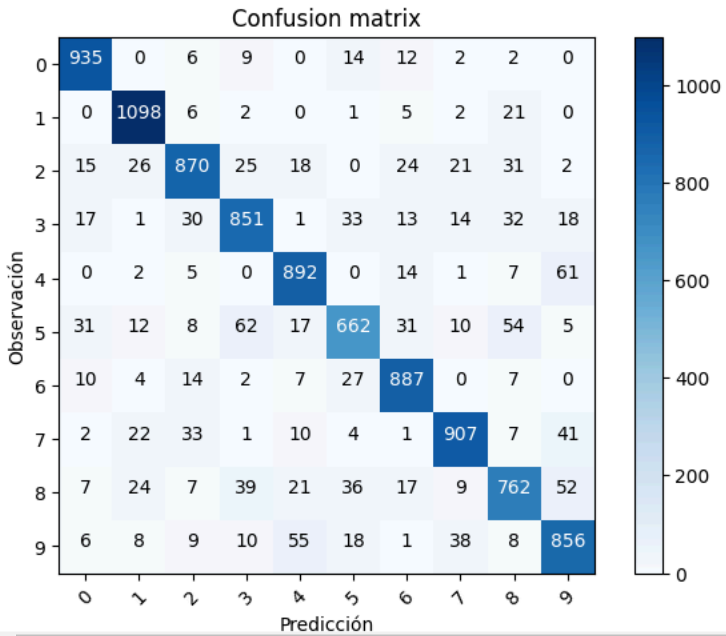
```
print('Test accuracy:', test_acc)
```

```
Test accuracy: 0.871999979019165
```

```
# Look at confusion matrix
#Note, this code is taken straight from the SKLEARN website, an nice way of viewing confusion matrix.
def plot_confusion_matrix(cm, classes,
                          normalize=False,
                          title='Confusion matrix',
                          cmap=plt.cm.Blues):
    """
    This function prints and plots the confusion matrix.
    Normalization can be applied by setting `normalize=True`.
    """
    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)

    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]

    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, cm[i, j],
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")

    plt.tight_layout()
    plt.ylabel('Observación')
    plt.xlabel('Predicción')
```

```
from collections import Counter
from sklearn.metrics import confusion_matrix
import itertools

# Predict the values from the validation dataset
Y_pred = model.predict(x_test)
# Convert predictions classes to one hot vectors
Y_pred_classes = np.argmax(Y_pred, axis = 1)
# Convert validation observations to one hot vectors
Y_true = np.argmax(y_test, axis = 1)
# compute the confusion matrix
confusion_mtx = confusion_matrix(Y_true, Y_pred_classes)
# plot the confusion matrix
plot_confusion_matrix(confusion_mtx, classes = range(10))
```

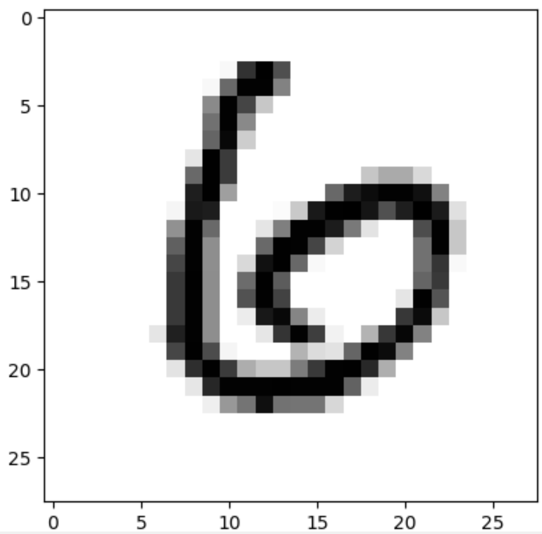
```
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step
```



Use the model

```
x_test_old = x_test.reshape(10000, 28,28)
plt.imshow(x_test_old[11], cmap=plt.cm.binary)
```

```
<matplotlib.image.AxesImage at 0x7e82bff11ad0>
```



```
predictions = model.predict(x_test)
```

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

```
np.argmax(predictions[11])
```

```
6
```

```
print(predictions[11])
```

```
[0.02460003 0.07596562 0.22225824 0.07534765 0.07584362 0.07906184
 0.3106948  0.00432798 0.12130976 0.01059039]
```

```
np.sum(predictions[11])
```

```
0.99999994
```

Convolutional Neural Network

```
from tensorflow.keras import Sequential
from tensorflow.keras.layers import Conv2D
from tensorflow.keras.layers import MaxPooling2D
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import Flatten
from tensorflow.keras.layers import Input

model = Sequential()
model.add(Input(shape=(28, 28, 1)))  # Define the input shape here
model.add(Conv2D(32, (5, 5), activation='relu'))
model.add(MaxPooling2D((2, 2)))
model.add(Conv2D(64, (5, 5), activation='relu'))
model.add(MaxPooling2D((2, 2)))

from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import Flatten

model.add(Flatten())
model.add(Dense(10, activation='softmax'))
model.summary()
```

Model: "sequential_1"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 24, 24, 32)	832
max_pooling2d (MaxPooling2D)	(None, 12, 12, 32)	0
conv2d_1 (Conv2D)	(None, 8, 8, 64)	51,264
max_pooling2d_1 (MaxPooling2D)	(None, 4, 4, 64)	0
flatten (Flatten)	(None, 1024)	0
dense_2 (Dense)	(None, 10)	10,250

Total params: 62,346 (243.54 KB)
Trainable params: 62,346 (243.54 KB)

```
from tensorflow.keras.utils import to_categorical

#mnist = tf.keras.datasets.mnist(train_images, train_labels), (test_images, test_labels) = mnist.load_data(path='/gpfs/projects/nct00/n

mnist = tf.keras.datasets.mnist
(train_images, train_labels), (test_images, test_labels) = mnist.load_data()

print (train_images.shape)
print (train_labels.shape)
train_images = train_images.reshape((60000, 28, 28, 1))
train_images = train_images.astype('float32') / 255

test_images = test_images.reshape((10000, 28, 28, 1))
test_images = test_images.astype('float32') / 255

train_labels = to_categorical(train_labels)
test_labels = to_categorical(test_labels)

print (train_images.shape)
print (train_labels.shape)
```

```
(60000, 28, 28)
(60000,)
(60000, 28, 28, 1)
(60000, 10)
```

```
model.compile(loss='categorical_crossentropy',
              optimizer='sgd',
```

```
              metrics=['accuracy'])
```

```
model.fit(train_images, train_labels, batch_size=100, epochs=5, verbose=1)
```

```
Epoch 1/5
600/600 ━━━━━━━━━━━━━━━━━━━━ 4s 3ms/step - accuracy: 0.5847 - loss: 1.5815
Epoch 2/5
600/600 ━━━━━━━━━━━━━━━━━━━━ 4s 4ms/step - accuracy: 0.9181 - loss: 0.2799
Epoch 3/5
600/600 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.9439 - loss: 0.1900
Epoch 4/5
600/600 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.9555 - loss: 0.1525
Epoch 5/5
600/600 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.9638 - loss: 0.1247
<keras.src.callbacks.history.History at 0x7e82d00fb2d0>
```

```
test_loss, test_acc = model.evaluate(test_images, test_labels)

print('Test accuracy:', test_acc)
```

```
313/313 ━━━━━━━━━━━━━━━━━━━━ 2s 5ms/step - accuracy: 0.9687 - loss: 0.1160
Test accuracy: 0.972100019454956
```