DIGIT RECOGNIZER USING NEURAL NETWORKS

1 Understand the problem and import the more important libraries

In [52]:
```
import pandas as pd
from tensorflow import keras
```

Pandas

Fast, powerful, flexible and easy to use open source library, built on top of the Python programming language.

Pandas Dataframe

It's a two-dimensional, size-mutable, potentially heterogeneous tabular data.

Data structure also contains labeled axes (rows and columns). Arithmetic operations align on both row and column labels. Can be thought of as a dict-like container for "array" objects. Pandas dataframes are based on numpy ndarrays which we should know some about.

In [53]:
```
#PUT THE CSV DATA INTO A PANDAS DATAFRAME
digits_train = pd.read_csv('C:/Users/torre/OneDrive/Escritorio/PERSONAL/DATA SCIENCE/I
digits_train.tail()
```

Out[53]:

	label	pixel0	pixel1	pixel2	pixel3	pixel4	pixel5	pixel6	pixel7	pixel8	...	pixel774	pixel775	pixe
41995	0	0	0	0	0	0	0	0	0	0	...	0	0	
41996	1	0	0	0	0	0	0	0	0	0	...	0	0	
41997	7	0	0	0	0	0	0	0	0	0	...	0	0	
41998	6	0	0	0	0	0	0	0	0	0	...	0	0	
41999	9	0	0	0	0	0	0	0	0	0	...	0	0	

5 rows × 785 columns

In [54]:
```
import numpy as np
import matplotlib.pyplot as pt
```

In [55]:
```
digits_train.shape
```

Out[55]: `(42000, 785)`

In [56]:
```
digits_train["label"]
```

Out[56]:
```
0        1
1        0
2        1
3        4
4        0
        ..
41995    0
41996    1
41997    7
41998    6
41999    9
Name: label, Length: 42000, dtype: int64
```

2 Data Preparation

In [57]:
```
import seaborn as sns

Y_train = digits_train["label"]
X_train = digits_train.drop(labels = "label", axis = 1)

g = sns.countplot(Y_train)

Y_train.value_counts().sort_values(ascending=True)
```

```
C:\Users\torre\anaconda3\envs\GonzaloTorres\lib\site-packages\seaborn\_decorators.py:3
6: FutureWarning: Pass the following variable as a keyword arg: x. From version 0.12,
the only valid positional argument will be `data`, and passing other arguments without
an explicit keyword will result in an error or misinterpretation.
  warnings.warn(
```

Out[57]:
```
5    3795
8    4063
4    4072
0    4132
6    4137
2    4177
9    4188
3    4351
7    4401
1    4684
Name: label, dtype: int64
```

We can see how is the proportion of the numbers "label" we have in our dataset. When we train the model it would be easier to recognize a 1 than a 5 due to a bigger training subset of data to learn.

2.1 Cleaning

In [58]:
```
X_train.isnull().any().describe()
```

Out[58]:
```
count       784
unique        1
top       False
freq        784
dtype: object
```

In [59]:
```
test = pd.read_csv('C:/Users/torre/OneDrive/Escritorio/PERSONAL/DATA SCIENCE/Kaggle/Di

test.isnull().any().describe()
```

Out[59]:
```
count       784
unique        1
top       False
freq        784
dtype: object
```

2.2 Normalization

Moreover de CNN converge faster on [0,1] data than on [0,255]

In [60]:
```
#test = test.drop(labels = "label", axis = 1)

#Normalize the data
X_train = X_train / 255
test = test / 255
```

In [61]:
```
test.isnull().any().describe()
```

Out[61]:
```
count       784
unique        1
top       False
freq        784
dtype: object
```

In [62]:
```
#TRAIN DATA
X_train.head()
```

Out[62]:

	pixel0	pixel1	pixel2	pixel3	pixel4	pixel5	pixel6	pixel7	pixel8	pixel9	...	pixel774	pixel775	pixel77
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.
1	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.
2	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.
3	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.
4	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.

5 rows × 784 columns

In [63]:
```
#TEST DATA
test.head()
```

Out[63]:

	pixel0	pixel1	pixel2	pixel3	pixel4	pixel5	pixel6	pixel7	pixel8	pixel9	...	pixel774	pixel775	pixel77
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.
1	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.
2	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.
3	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.
4	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.

5 rows × 784 columns

2.3 Reshape Data

In [64]:
```
#Reshape images into 3D matrices
X_train = X_train.values.reshape(-1,28,28,1)
test = test.values.reshape(-1,28,28,1)
```

Keras requieres an extra parameter at the end to define channels. We use only one due to gray scale image. If we were working with RGB we may use 3.

2.4 Label Encoding

One-Hot Encoder

Though label encoding is straight but it has the disadvantage that the numeric values can be misinterpreted by algorithms as having some sort of hierarchy/order in them. This ordering issue is addressed in another common alternative approach called 'One-Hot Encoding'. In this strategy, each category value is converted into a new column and assigned a 1 or 0 (notation for true/false) value to the column.

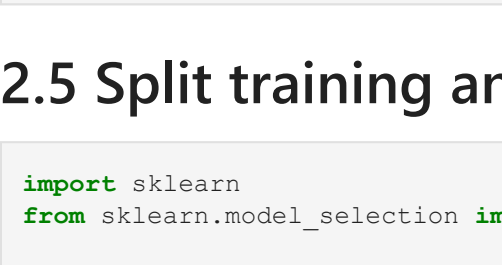
In [65]:
```
import keras
import keras.utils
from keras.utils import to_categorical
#Encode labels to one hot vector. Ex -> 2 = (0,0,1,0,0,0,0,0,0,0)
Y_train = to_categorical(Y_train, num_classes = 10)
```

2.5 Split training and validation sets

In [66]:
```
import sklearn
from sklearn.model_selection import train_test_split

#Set the random seed
random_seed = 2

#Split the train and validation set for the fitting
X_train, X_val, Y_train, Y_val = train_test_split(X_train, Y_train, test_size=0.1, ra
```

10% for evaluation, 90% for training

In [67]:
```
#some examples
g = pt.imshow(X_train[0][:,:,0])
```



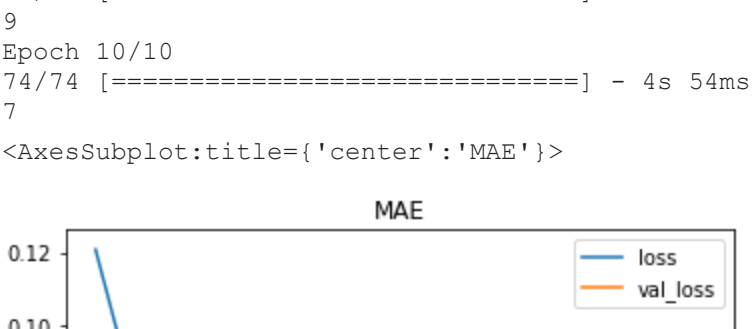
3. CNN

3.1 Define the model

In [68]:
```
#1. Neural Network with 2 hidden layers of 1024 units

# - The input shape is a Flatten layer of 28x28 input shape (784)

# - We introduce here the mean absolute error or MAE. For each prediction y_pred, MAE
#   the true target y_true by an absolute difference abs(y_true - y_pred)

# - We should make a neural network with 10 binary outputs (True, False) depending on

#INMPORTANT INFO: The loss function measures the disparity between the the target's t

from tensorflow.keras import layers
from sklearn import preprocessing


model1 = keras.Sequential([
    layers.Flatten(input_shape=(28,28,1)),
    layers.Dense(units=1024, activation='relu'),
    layers.Dense(units=1024, activation='relu'),
    layers.Dense(units=10),
])

model1.compile(
    optimizer='adam',
    loss='mae'
)

history1 = model1.fit(
    X_train, Y_train,
    validation_data=(X_val, Y_val),
    batch_size=512,
    epochs=10
)

history_df = pd.DataFrame(history1.history)
history_df.loc[:, ['loss', 'val_loss']].plot(title="MAE")
```

```
Epoch 1/10
74/74 [==============================] - 5s 66ms/step - loss: 0.1207 - val_loss: 0.066
4
Epoch 2/10
74/74 [==============================] - 4s 48ms/step - loss: 0.0510 - val_loss: 0.039
9
Epoch 3/10
74/74 [==============================] - 4s 50ms/step - loss: 0.0339 - val_loss: 0.031
7
Epoch 4/10
74/74 [==============================] - 4s 53ms/step - loss: 0.0275 - val_loss: 0.026
9
Epoch 5/10
74/74 [==============================] - 4s 51ms/step - loss: 0.0236 - val_loss: 0.024
2
Epoch 6/10
74/74 [==============================] - 4s 50ms/step - loss: 0.0207 - val_loss: 0.022
3
Epoch 7/10
74/74 [==============================] - 4s 50ms/step - loss: 0.0183 - val_loss: 0.020
2
Epoch 8/10
74/74 [==============================] - 4s 53ms/step - loss: 0.0163 - val_loss: 0.018
8
Epoch 9/10
74/74 [==============================] - 4s 52ms/step - loss: 0.0146 - val_loss: 0.017
9
Epoch 10/10
74/74 [==============================] - 4s 54ms/step - loss: 0.0132 - val_loss: 0.016
7
```

Out[68]: `<AxesSubplot:title={'center':'MAE'}>`



It's not too bad but we should prevent overfitting because the deviation between the loss function and the validation loss is growing while we are still training our net.

We will add Dropout to prevent this in our model.

In [69]:
```
#2. Neural Network with 2 hidden layers of 1024 units and Dropout


model2 = keras.Sequential([
    layers.Flatten(input_shape=(28,28,1)),
    layers.Dropout(0.3),
    layers.Dense(units=1024, activation='relu'),
    layers.Dropout(0.3),
    layers.Dense(units=1024, activation='relu'),
    layers.Dropout(0.3),
    layers.Dense(units=10),
])

model2.compile(
    optimizer='adam',
    loss='mae'
)

history2 = model2.fit(
    X_train, Y_train,
    validation_data=(X_val, Y_val),
    batch_size=512,
    epochs=10
)

history_df = pd.DataFrame(history2.history)
history_df.loc[:, ['loss', 'val_loss']].plot(title="MAE")
```

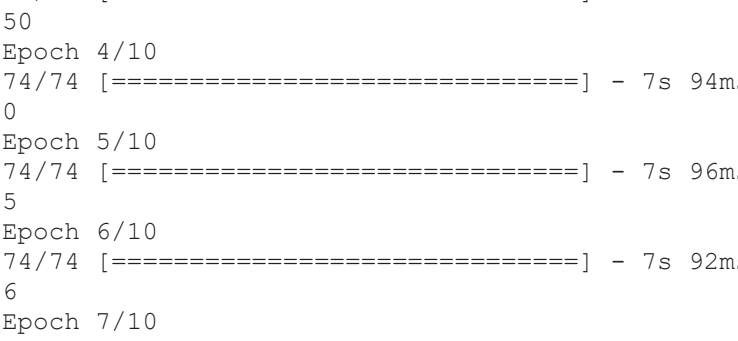
```
Epoch 1/10
74/74 [==============================] - 5s 68ms/step - loss: 0.1423 - val_loss: 0.087
5
Epoch 2/10
74/74 [==============================] - 6s 75ms/step - loss: 0.0876 - val_loss: 0.076
7
Epoch 3/10
74/74 [==============================] - 5s 68ms/step - loss: 0.0811 - val_loss: 0.072
1
Epoch 4/10
74/74 [==============================] - 5s 69ms/step - loss: 0.0777 - val_loss: 0.071
4
Epoch 5/10
74/74 [==============================] - 5s 71ms/step - loss: 0.0746 - val_loss: 0.071
3
Epoch 6/10
74/74 [==============================] - 5s 70ms/step - loss: 0.0724 - val_loss: 0.070
9
Epoch 7/10
74/74 [==============================] - 5s 69ms/step - loss: 0.0708 - val_loss: 0.069
8
Epoch 8/10
74/74 [==============================] - 6s 75ms/step - loss: 0.0697 - val_loss: 0.068
9
Epoch 9/10
74/74 [==============================] - 5s 69ms/step - loss: 0.0687 - val_loss: 0.069
3
Epoch 10/10
74/74 [==============================] - 6s 75ms/step - loss: 0.0681 - val_loss: 0.069
2
```

Out[69]: `<AxesSubplot:title={'center':'MAE'}>`

In [70]:
```
# 3. Introducing the BatchNormalization

model3 = keras.Sequential([
    layers.Flatten(input_shape=(28,28,1)),
    layers.Dropout(0.3),
    layers.BatchNormalization(),
    layers.Dense(units=1024, activation='relu'),
    layers.Dropout(0.3),
    layers.BatchNormalization(),
    layers.Dense(units=1024, activation='relu'),
    layers.Dropout(0.3),
    layers.BatchNormalization(),
    layers.Dense(units=10),
])

model3.compile(
    optimizer='adam',
    loss='mae'
)

history3 = model3.fit(
    X_train, Y_train,
    validation_data=(X_val, Y_val),
    batch_size=512,
    epochs=10
)

history_df = pd.DataFrame(history3.history)
history_df.loc[:, ['loss', 'val_loss']].plot(title="MAE")
```

```
Epoch 1/10
74/74 [==============================] - 7s 96ms/step - loss: 0.5306 - val_loss: 0.139
3
Epoch 2/10
74/74 [==============================] - 7s 94ms/step - loss: 0.2382 - val_loss: 0.113
4
Epoch 3/10
74/74 [==============================] - 7s 100ms/step - loss: 0.1702 - val_loss: 0.10
50
Epoch 4/10
74/74 [==============================] - 7s 94ms/step - loss: 0.1429 - val_loss: 0.094
0
Epoch 5/10
74/74 [==============================] - 7s 96ms/step - loss: 0.1311 - val_loss: 0.086
5
Epoch 6/10
74/74 [==============================] - 7s 92ms/step - loss: 0.1175 - val_loss: 0.076
6
Epoch 7/10
74/74 [==============================] - 7s 99ms/step - loss: 0.1052 - val_loss: 0.074
6
Epoch 8/10
74/74 [==============================] - 7s 94ms/step - loss: 0.0950 - val_loss: 0.077
7
Epoch 9/10
74/74 [==============================] - 7s 96ms/step - loss: 0.0875 - val_loss: 0.060
8
Epoch 10/10
74/74 [==============================] - 7s 93ms/step - loss: 0.0813 - val_loss: 0.056
3
```

Out[70]: `<AxesSubplot:title={'center':'MAE'}>`



In my opininon the best model is the 3rd, so lets use it to do a prediction

In [71]:
```
# make a prediction
#test = to_categorical(test, num_classes = 10)
y = model3.predict(test)
y.shape
```

Out[71]: `(28000, 10)`

In [72]:
```
print(y.max())
```

```
3.255351
```

In [73]:
```
scaler = preprocessing.MinMaxScaler()
print(scaler.fit(y))

print(scaler.data_max_)

print(scaler.transform(y))
```

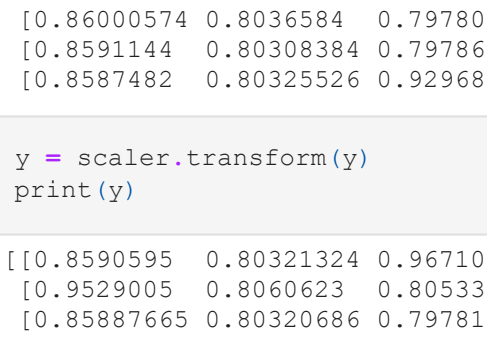
```
MinMaxScaler()
[1.0141789 1.2026794 1.1342145 1.2769444 1.3517382 1.5347857 3.255351
 2.5804074 1.6731478 2.045288 ]
[[0.8590595  0.80321324 0.9671085  ... 0.23708358 0.40468493 0.5145989 ]
 [0.9529005  0.8060623  0.8053316  ... 0.23759292 0.40913996 0.51517946]
 [0.85887665 0.80320686 0.7978183  ... 0.23649366 0.4076995  0.5941118 ]
 ...
 [0.86000574 0.8036584  0.7978039  ... 0.23394582 0.4052097  0.515112  ]
 [0.8591144  0.80308384 0.79786664 ... 0.23652704 0.40723002 0.6319632 ]
 [0.8587482  0.80325526 0.92968404 ... 0.23681265 0.4057161  0.51470625]]
```

In [74]:
```
y = scaler.transform(y)
print(y)
```

```
[[0.8590595  0.80321324 0.9671085  ... 0.23708358 0.40468493 0.5145989 ]
 [0.9529005  0.8060623  0.8053316  ... 0.23759292 0.40913996 0.51517946]
 [0.85887665 0.80320686 0.7978183  ... 0.23649366 0.4076995  0.5941118 ]
 ...
 [0.86000574 0.8036584  0.7978039  ... 0.23394582 0.4052097  0.515112  ]
 [0.8591144  0.80308384 0.79786664 ... 0.23652704 0.40723002 0.6319632 ]
 [0.8587482  0.80325526 0.92968404 ... 0.23681265 0.4057161  0.51470625]]
```

In [75]:
```
image_index = 2853
pt.imshow(test[image_index].reshape(28, 28),cmap='Greys')
predict = test[image_index].reshape(28,28)
pred = model3.predict(test[image_index].reshape(1, 28, 28, 1))
print(pred.argmax())
```

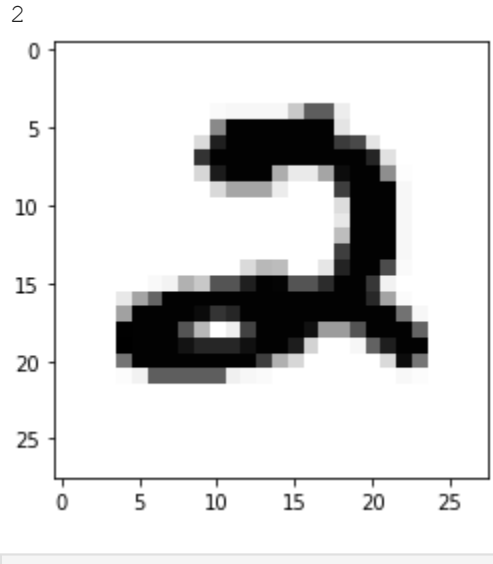
```
1
```



In [76]:
```
y_pred = model3.predict(test)
y_pred = np.argmax(y_pred,axis=1)

#Create a  DataFrame with the passengers ids and our prediction regarding whether the
sub_file = pd.read_csv('C:/Users/torre/OneDrive/Escritorio/PERSONAL/DATA SCIENCE/Kagg
predictions = model3.predict(test)
submission = pd.DataFrame({'ImageId': list(range(1, len(y_pred)+1)), 'Label':y_pred})

submission.to_csv('C:/Users/torre/OneDrive/Escritorio/PERSONAL/DATA SCIENCE/Kaggle/Di
digits_train["label"].to_csv('C:/Users/torre/OneDrive/Escritorio/PERSONAL/DATA SCIENC
```

In [77]:
```
image_index = 0
pt.imshow(test[image_index].reshape(28, 28),cmap='Greys')
predict = test[image_index].reshape(28,28)
pred = model.predict(test[image_index].reshape(1, 28, 28, 1))
print(pred.argmax())
```

```
2
```



In [ ]: