**Cargando BASE64**

Para empezar, es de vital importancia reconocer los datos con los que estamos trabajando, en esta ocasión revisaremos los datos de sign\_mnist\_base64 que tendrá dentro un JSON con una estructura de llave/valor con la codificación de la imagen.

Para el primero parte manipularemos imágenes en **BASE64,** por lo que importaremos este módulo

``` {code-**block}** python

import **base64**

Definimos en la variable url la ubicación del archivo a extraer la información.

url = "/tmp/databasesLoadData/sign\_mnist\_base64/data.json"

Cargamos los datos del JSON a memoria.

**with** open(url) **as** f:

**data** = json.load(**f**)

Cargaremos los datos en una variable, determinamos la locación a guardar la imagen y abrimos un archivo con el seudónimo de file\_to\_save, decodificaremos el BASE64 y lo guardaremos en un buffer, finalmente lo guardaremos en esta locación y con esto quedará escrito en disco.

base64\_img\_bytes = data['b'].encode('utf-8')

path\_img = "/tmp/decoded\_image.png"

**with** open(path\_img, "wb") **as** file\_to\_save:

decoded\_image\_data = base64.decodebytes(base64\_img\_bytes)

file\_to\_save.write(decoded\_image\_data)

Podemos consumir esta imagen abriéndola con PIL.

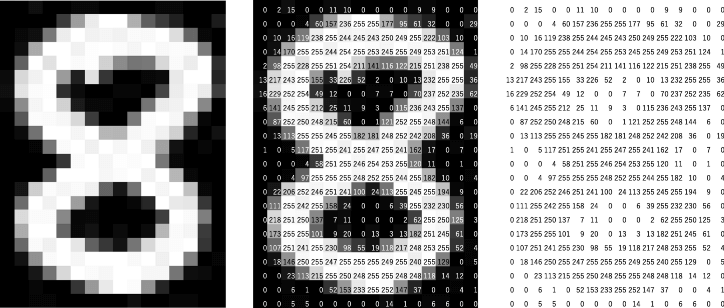
**img** = Image.open(path\_img)

**img**

**Entendiendo la anatomía de las imágenes**

Las imágenes son arrays anidados con diferentes canales (cada canal representando un color diferente), en esta ocasión el único canal es de la escala de grises, donde cada pixel puede encontrarse entre los valores de 0 - 255, siendo 0 negro y 255 blanco.

Esta interpretación matricial nos permite guardar imágenes en estructuras de datos como vectores, donde diferentes formatos son directamente compatibles con esta abstracción.



**Cargando CSV**

Para esta ocasión haremos uso de pandas para el manejo de CSV, además de otras librerías para la manipulación e interpretación de datos.

**import** numpy as np

**import** pandas as pd

%matplotlib inline

**import** matplotlib.pyplot as plt

from PIL **import** Image

**import** seaborn as sns

Leeremos los archivos CSV con la función read\_csv de pandas en las locaciones de los archivos de entrenamiento y prueba.

train = pd.read\_csv("/tmp/databasesLoadData/sign\_mnist\_train/sign\_mnist\_train.csv")

test = pd.read\_csv("/tmp/databasesLoadData/sign\_mnist\_test/sign\_mnist\_test.csv")

Podemos analizar los primeros datos del dataset con el método head que nos traerá los primeros 5 registros, notaremos que son 785 elementos, donde el primero pertenece a la etiqueta de la imagen y el resto a cada pixel (28x28=784).

**train**.head()

5 **rows** × 785 **columns**

Tenemos acceso a las dimensiones del dataset con el atributo shape, que nos retornará 27455 imágenes de 28x28 con etiqueta.

train.shape

(27455, 785)

Para arreglar la dimensionalidad, separaremos la etiqueta de los datos, por lo que aislaremos los valores en una nueva variable.

labels = train['label'].values

A su vez, eliminaremos esta columna del dataset original.

train.drop('label', axis = 1, inplace = **True**)

train.head()

5 **rows** × 784 columns

Para almacenar las imágenes en memoria las cargaremos desde el dataframe (únicamente tomando los atributos de values), si verificamos las dimensiones tendremos un array de numpy de 27455x784 que podremos graficar desde matplotlib.

images = train.values

images.shape

(27455, 784)

plt.imshow(images[1].reshape(28,28))

Con esto hemos cargado imágenes a partir de formato BASE64 y CSV a memoria para ser interpretadas como tensores.

**Contribución creada por** Sebastián Franco Gómez.