Cuando trabajes con datasets encontrarás bases de datos ya generadas y listas para consumo, pero eventualmente te toparás con la necesidad de crear tus propios datos, por lo que deberás encontrar una manera de cargarlos.

En el momento que cargas datos a memoria, lo haces directamente a la memoria RAM del sistema, por lo que si cargas un dataset pesado de golpe, es probable que termines colapsando tu entorno de trabajo por saturación de recursos.

Para evitar este problema, se crean los generadores, una estructura de datos que generará datos solo si es recorrida, optimizando memoria.

**Descargando el dataset de lenguaje de señas**

Descargaremos el repositorio desde GCP.

!wget --**no**-check-certificate https://storage.googleapis.**com**/platzi-tf2/**sign**-**language**-img.zip \

-O /tmp/**sign**-**language**-img.zip

Descomprimiremos el archivo, para esto usaremos los módulos de os y zipfile.

**import** os

**import** zipfile

local\_zip = "/tmp/sign-language-img.zip"

zip\_ref = zipfile.ZipFile(local\_zip, "r")

zip\_ref.extractall("/tmp/sign-language-img")

zip\_ref.close()

La estructura del dataset constará en 2 directorios principales: Test y Train, donde para cada clase (letra en señas) tendremos un directorio con sus respectivos ejemplos.

sign-language-image/

├── Test/

│ └── A-Z/

│ └── examples.jpg

└── Train/

└── A-Z/

└── examples.jpg

**Cargando el dataset con Keras dataset generator**

Para llevar a cabo el proceso de carga, haremos uso de varias librerías como TensorFlow, matplotlib y numpy.

**import** numpy as np

%matplotlib inline

**import** matplotlib.pyplot as plt

**import** matplotlib.image as mpimg

**import** string

**import** tensorflow as tf

from tensorflow.keras.preprocessing.image **import** ImageDataGenerator

Cargaremos las rutas donde se encuentran nuestros dataset.

train\_dir = "/tmp/sign-language-img/Train"

test\_dir = "/tmp/sign-language-img/Test"

Generaremos los data generators, para esta ocasión reescalaremos los datos al rango de 0 a 1 para mejorar la convergencia del modelo, además, dividiremos el 20% de los datos de prueba a validación para monitorear el rendimiento del modelo en vivo.

train\_datagen = ImageDataGenerator(rescale = 1/255)

test\_datagen = ImageDataGenerator(rescale = 1/255, validation\_split = 0.2)

Para cargar las imágenes, haremos uso del método flow\_from\_directory del generador, determinaremos el directorio fuente, el tamaño que tendrán las imágenes (el generador las redimensionará de ser necesario), los lotes de procesamiento, el tipo de clases, el tipo de escala de colores y el subset al que pertenecen.

train\_generator = train\_datagen.flow\_from\_directory(

train\_dir,

target\_size = (28,28),

batch\_size = 128,

class\_mode = "categorical",

color\_mode = "grayscale",

subset = "training"

)

Para los subsets de validación y prueba será el mismo proceso, donde cambiarán los nombres de las variables y las fuentes.

validation\_generator = test\_datagen.flow\_from\_directory(

test\_dir,

target\_size = (28,28),

batch\_size = 128,

class\_mode = "categorical",

color\_mode = "grayscale",

subset = "validation"

)

test\_generator = train\_datagen.flow\_from\_directory(

test\_dir,

target\_size = (28,28),

batch\_size = 128,

class\_mode = "categorical",

color\_mode = "grayscale",

)

Para generar las clases haremos una pequeña list comprehension recorriendo los caracteres ASCII omitiendo las letras J y Z.

classes = [**char** **for** **char** **in** **string**.ascii\_uppercase **if** **char** != "J" **if** **char** != "Z"]

Para graficar imágenes crearemos la función plotImages que recibirá un array de imágenes y las mostrará en pantalla en grupos de 5.

**def** **plotImages**(images\_arr):

fig, axes = plt.subplots(1, 5, figsize = (10, 10))

axes = axes.flatten()

**for** img, ax **in** zip(images\_arr, axes):

ax.imshow(img[:,:,0])

ax.axis("off")

plt.tight\_layout()

plt.show()

Para hacer uso de esta función generaremos un conjunto de imágenes, esto nos retornará un array de imágenes que daremos como parámetro.

sample\_training\_images, \_ = next(train\_generator)

**plotImages**(sample\_training\_images[:5])

Con esto hemos cargado imágenes en memoria sin necesidad de saturar la memoria del sistema, cada vez que requieras iterar sobre tu dataset el generador solo generará las imágenes necesarias.

**Contribución creada por** Sebastián Franco Gómez.