UNIVERSIDAD AUTONOMA DE AGUASCALIENTES

CENTRO DE CIENCIAS BASICAS

DEPARTAMENTO DE SISTEMAS DE INFORMACION

MATERIA: TECNOLOGIAS INMERSIVAS

Clasificador Perros y Gatos

PROFESOR(A): M.I.T.C. MA GUADALUPE IBARRA NAVA

Carlos Manuel Solis Hernandez
LITC | 31 de marzo de 2024

Código:

Importaciones: Importa las bibliotecas necesarias, como TensorFlow, TensorFlow Datasets, matplotlib y OpenCV.

```
#descargar gatos y perror
import tensorflow as tf
import tensorflow_datasets as tfds
import matplotlib.pyplot as plt
import cv2
```

Carga de datos: Utiliza tfds.load() para cargar el conjunto de datos "cats_vs_dogs" con la opción as_supervised=True para obtener los datos en formato (imagen, etiqueta). También utiliza with_info=True para obtener información sobre los metadatos del conjunto de datos.

```
datos, metadatos = tfds.load('cats_vs_dogs' , as_supervised = True ,
with_info=True)
```

Visualización de imágenes de gatos: Itera sobre las primeras 25 imágenes del conjunto de datos de entrenamiento, redimensiona cada imagen a un tamaño específico, convierte las imágenes a escala de grises y las muestra usando.

```
matplotlib.
plt.figure(figsize=(20,20))
TAMANO_IMG =100
for i, (imagen, etiqueta) in enumerate (datos['train'].take(25)):
imagen = cv2.resize (imagen.numpy(), (TAMANO_IMG, TAMANO_IMG))
imagen = cv2.cvtColor(imagen, cv2.COLOR_BGR2GRAY)
plt.subplot (5,5,i+1)
plt.xticks([])
plt.yticks([])
```

```
plt.imshow(imagen, cmap ='gray')
```

Preparación de datos de entrenamiento: Itera sobre todas las imágenes del conjunto de datos de entrenamiento, las redimensiona, convierte a escala de grises y las prepara en un formato adecuado para el entrenamiento del modelo. Cada imagen se guarda junto con su etiqueta en una lista datos_entrenamiento.

```
#datos entrenamiento
datos_entrenamiento = []
for i, (imagen, etiqueta) in enumerate(datos['train']):
imagen=cv2.resize(imagen.numpy(), (TAMANO_IMG, TAMANO_IMG))
imagen=cv2.cvtColor(imagen, cv2.COLOR_BGR2GRAY)
imagen= imagen.reshape(TAMANO_IMG, TAMANO_IMG, 1)
datos_entrenamiento.append([imagen, etiqueta])
X = [] #pixeles de las imagenes de entrada
y = [] #etiquetas (perros=1 y gatos=0)
```

Construcción del conjunto de datos de entrada y etiquetas: Extrae las imágenes y las etiquetas de la lista datos_entrenamiento y las guarda en listas separadas X e y respectivamente.

```
for imagen, etiqueta in datos_entrenamiento:
X.append(imagen)
y.append(etiqueta)
```

Normalización de datos: Normaliza las imágenes dividiendo los valores de píxeles por 255 y convierte las etiquetas a un array numpy.

```
import numpy as np
X = np.array(X).astype(float) / 255
y = np.array(y)
```

Definición de arquitecturas de modelos: Define tres modelos diferentes utilizando diferentes arquitecturas: un modelo denso, un modelo de red neuronal convolucional (CNN) y otro modelo CNN con regularización de Dropout.

```
#arquitectura y parametros
modeloDenso = tf.keras.models.Sequential([
tf.keras.layers.Flatten(input_shape=(100, 100, 1)),
tf.keras.layers.Dense(150, activation='relu'),
tf.keras.layers.Dense(150, activation='relu'),
tf.keras.layers.Dense(1, activation='sigmoid')
])
modeloCNN = tf.keras.models.Sequential([
tf.keras.layers.Conv2D(32, (3,3), activation='relu', input_shape=(100,
100,
1)),
tf.keras.layers.MaxPooling2D(2, 2),
tf.keras.layers.Conv2D(64, (3,3), activation='relu'),
tf.keras.layers.MaxPooling2D(2, 2),
tf.keras.layers.Conv2D(128, (3,3), activation='relu'),
tf.keras.layers.MaxPooling2D(2, 2),
tf.keras.layers.Flatten(),
tf.keras.layers.Dense(100, activation='relu'),
tf.keras.layers.Dense(1, activation='sigmoid')
])
modeloCNN2 = tf.keras.models.Sequential([
tf.keras.layers.Conv2D(32, (3,3), activation='relu', input_shape=(100,
100,
1)),
tf.keras.layers.MaxPooling2D(2, 2),
tf.keras.layers.Conv2D(64, (3,3), activation='relu'),
tf.keras.layers.MaxPooling2D(2, 2),
tf.keras.layers.Conv2D(128, (3,3), activation='relu'),
```

```
tf.keras.layers.MaxPooling2D(2, 2),
tf.keras.layers.Dropout(0.5),
tf.keras.layers.Flatten(),
tf.keras.layers.Dense(250, activation='relu'),
tf.keras.layers.Dense(1, activation='sigmoid')
])
```

Compilación de modelos: Compila cada modelo con el optimizador 'adam' y la función de pérdida 'binary_crossentropy' para problemas de clasificación binaria. Se monitorea la precisión como métrica.

```
modeloDenso.compile(optimizer='adam',
loss='binary_crossentropy',
metrics=['accuracy']
)
modeloCNN.compile(optimizer='adam',
loss='binary_crossentropy',
metrics=['accuracy']
)
modeloCNN2.compile(optimizer='adam',
loss='binary_crossentropy',
metrics=['accuracy']
)
```

Entrenamiento de modelos: Entrena el modelo denso utilizando los datos preparados. Se utiliza TensorBoard para guardar los resultados del entrenamiento en la carpeta 'logs/denso'.

```
from tensorflow.keras.callbacks import TensorBoard
tensorboardDenso = TensorBoard(log_dir='logs/denso') #guarda los
resultados de la
red densa en la carpeta denso
modeloDenso.fit(X, y, batch_size=32,
```

```
validation_split=0.15,

epochs=100,

callbacks=[tensorboardDenso])
```

Limpieza de RAM

```
import gc

gc.collect()
```

Se visualizan las imágenes de la variable X sin aplicar ninguna modificación. Para ello, se crea una figura con un tamaño específico y se utiliza un bucle for para mostrar las primeras 10 imágenes en subgráficos individuales. Se eliminan las marcas en los ejes x e y para una presentación más limpia de las imágenes.

```
#ver las imagenes de la variable X sin modificaciones por aumento de
datos

plt.figure(figsize=(20, 8))

for i in range(10):

  plt.subplot(2, 5, i+1)

  plt.xticks([])

  plt.yticks([])

  plt.imshow(X[i].reshape(100, 100), cmap="gray")
```

Se utiliza ImageDataGenerator de TensorFlow para aplicar diversas transformaciones a las imágenes, como rotación, desplazamiento, cambio de inclinación, zoom y volteo horizontal y vertical. Estas transformaciones ayudan a diversificar el conjunto de datos y mejorar la capacidad del modelo para generalizar. Se ajusta el generador a las imágenes en X.

```
#Realizar el aumento de datos con varias transformaciones. Al final,
graficar 10 como ejemplo

from tensorflow.keras.preprocessing.image import ImageDataGenerator

datagen = ImageDataGenerator(

    rotation_range=30,

    width_shift_range=0.2,

    height_shift_range=0.2,

    shear_range=15,
```

```
    zoom_range=[0.7, 1.4],
    horizontal_flip=True,
    vertical_flip=True
)
datagen.fit(X)
plt.figure(figsize=(20,8))
```

Se itera a través de los datos aumentados utilizando datagen.flow(), generando un lote de 10 imágenes en cada iteración debido al batch_size. Cada imagen se muestra en un subgráfico individual, manteniendo las mismas configuraciones de presentación que en la sección anterior. El bucle se detiene después de la primera iteración para mostrar solo 10 ejemplos.

```
#el batch_size al ser de 10, hace que en la primera
#iteracion se muestren las 10 imagenes. Por lo tanto,
#solo hay 1 iteracion en el for-loop más externo

for imagen, etiqueta in datagen.flow(X, y, batch_size=10, shuffle=False):
  for i in range(10):
    plt.subplot(2, 5, i+1)
    plt.xticks([])
    plt.yticks([])
    plt.imshow(imagen[i].reshape(100, 100), cmap="gray")
  break
```

Nuevos modelos

```
modeloDenso_AD = tf.keras.models.Sequential([
  tf.keras.layers.Flatten(input_shape=(100, 100, 1)),
  tf.keras.layers.Dense(150, activation='relu'),
  tf.keras.layers.Dense(150, activation='relu'),
  tf.keras.layers.Dense(1, activation='sigmoid')
])

modeloCNN_AD = tf.keras.models.Sequential([
```

```
  tf.keras.layers.Conv2D(32, (3,3), activation='relu', input_shape=(100,
100, 1)),
  tf.keras.layers.MaxPooling2D(2, 2),
  tf.keras.layers.Conv2D(64, (3,3), activation='relu'),
  tf.keras.layers.MaxPooling2D(2, 2),
  tf.keras.layers.Conv2D(128, (3,3), activation='relu'),
  tf.keras.layers.MaxPooling2D(2, 2),

  tf.keras.layers.Flatten(),
  tf.keras.layers.Dense(100, activation='relu'),
  tf.keras.layers.Dense(1, activation='sigmoid')
])

modeloCNN2_AD = tf.keras.models.Sequential([
  tf.keras.layers.Conv2D(32, (3,3), activation='relu', input_shape=(100,
100, 1)),
  tf.keras.layers.MaxPooling2D(2, 2),
  tf.keras.layers.Conv2D(64, (3,3), activation='relu'),
  tf.keras.layers.MaxPooling2D(2, 2),
  tf.keras.layers.Conv2D(128, (3,3), activation='relu'),
  tf.keras.layers.MaxPooling2D(2, 2),

  tf.keras.layers.Dropout(0.5),
  tf.keras.layers.Flatten(),
  tf.keras.layers.Dense(250, activation='relu'),
  tf.keras.layers.Dense(1, activation='sigmoid')
])

modeloDenso_AD.compile(optimizer='adam',
                    loss='binary_crossentropy',
                    metrics=['accuracy'])

modeloCNN_AD.compile(optimizer='adam',
```

```
                    loss='binary_crossentropy',
                    metrics=['accuracy'])

modeloCNN2_AD.compile(optimizer='adam',
                    loss='binary_crossentropy',
                    metrics=['accuracy'])
```

Este código separa los datos en dos conjuntos: entrenamiento y validación. Primero, se calcula el 85% de la longitud total de X para determinar cuántos datos se usarán para entrenamiento (19700), dejando el restante 15% para validación (3562). Luego, se asignan los primeros 19700 elementos de X a X_entrenamiento y los elementos restantes a X_validacion. De manera similar, se hace lo mismo con las etiquetas y para obtener y_entrenamiento y y_validacion.

```
#Separar los datos de entrenamiento y los datos de pruebas en variables
diferentes
len(X) * .85 #19700
len(X) - 19700 #3562

X_entrenamiento = X[:19700]
X_validacion = X[19700:]

y_entrenamiento = y[:19700]
y_validacion = y[19700:]
```

Usar la funcion flow del generador para crear un iterador que podamos enviar como entrenamiento a la funcion FIT del modelo

```
data_gen_entrenamiento = datagen.flow(X_entrenamiento, y_entrenamiento,
batch_size=32)
tensorboardCNN_AD = TensorBoard(log_dir='logs-new/cnn_AD')
```

Mejor modelo

```
modeloCNN_AD.fit(
    data_gen_entrenamiento,
    epochs=100, batch_size=32,
    validation_data=(X_validacion, y_validacion),
```

```
    steps_per_epoch=int(np.ceil(len(X_entrenamiento) / float(32))),
    validation_steps=int(np.ceil(len(X_validacion) / float(32))),
    callbacks=[tensorboardCNN_AD]
)
```

La línea modeloCNN_AD.save('perros-gatos-cnn-ad.h5') guarda el modelo de red neuronal convolucional (CNN) entrenado en un archivo en formato h5.

```
modeloCNN_AD.save('perros-gatos-cnn-ad.h5')
```

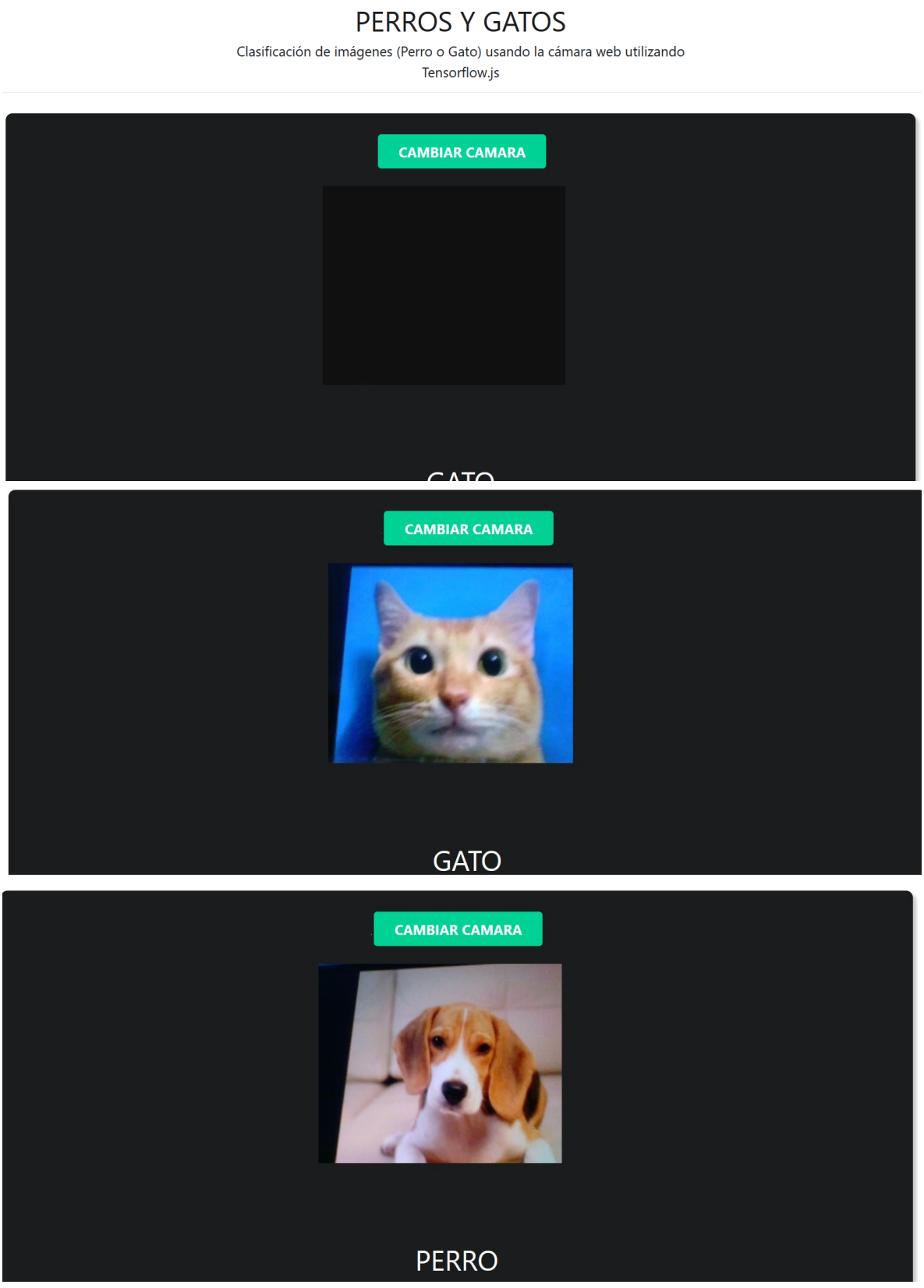
La línea !pip install tensorflowjs utiliza el comando para instalar la biblioteca tensorflowjs. Esta biblioteca es necesaria para convertir modelos de TensorFlow a formatos compatibles con TensorFlow.js.

```
!pip install tensorflowjs
```

Para convertir el modelo guardado en formato h5 a un formato compatible con TensorFlow.js.

```
!tensorflowjs_converter --input_format keras perros-gatos-cnn-ad.h5
carpeta_salida
```

Resultados:



[10]