Clasificador de imágenes

Lo que vamos a hacer es clasificar imágenes por visión artificial, donde el objetivo es analizar una imagen para asignarla en 2 categorías posibles “perro” ó “gato”. Esto lo vamos a hacer a través de redes neuronales convolucionales (CNN). Espero que el lector entienda que este problema puede extenderse a clasificar cualquier otro tipo de imágenes en otros conjuntos.

Redes neuronales convolucionales

Es una red neuronal diseñada específicamente para entender datos visuales. Son extremadamente efectivas para la visión por computadora.

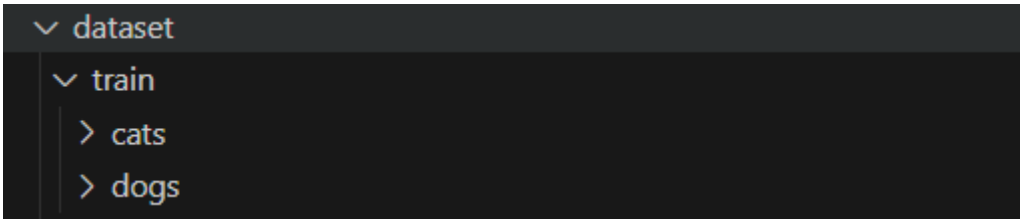
1 - Definición del problema

Hay que clasificar una imagen entre perro y gato con la ayuda de un modelo de red neuronal.

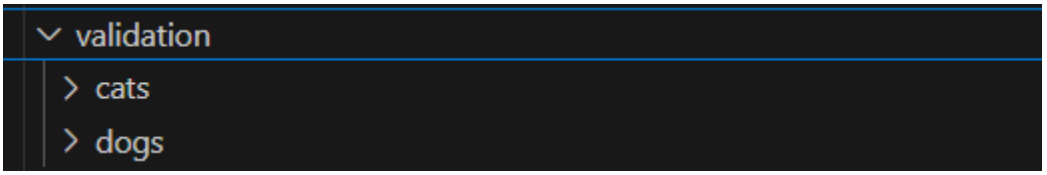
2 y 3 - Recolección y preprocesamiento de datos:

- Se necesita tener muchos ejemplos para nuestro caso vamos a tener más de 10 mil imágenes de perros y gatos (total más de 20 mil imágenes). Pero necesitamos 2 carpetas:

  CARPETA DE ENTRENAMIENTO: para los ejemplos de entrenamiento la cual la llamaremos “dataset/traint”

  CARPETA DE VALIDACIÓN: para saber si el entrenamiento es correcto.



- Redimensión hay que estandarizar la entrada de datos para nuestro caso es 128x128 y si el lector se pregunta cuál es el tamaño perfecto la respuesta es: “debe ser tan pequeña hasta el punto que sea distinguible para un ser humano”.

```
image_size = (128, 128)
batch_size = 32
```

```
train_generator = train_datagen.flow_from_directory(
    'dataset/train',
    target_size=image_size,
    batch_size=batch_size,
    class_mode='binary'
)
```

- Normalización vamos a ver píxeles y por ello todos los píxeles tienen que tener valores entre 0 y 255 pero esos valores no son adecuados para una red neuronal por ello vamos a colocarlo en valores entre 0 y 1.

```
train_datagen = ImageDataGenerator(
    rescale=1.0/255,
    rotation_range=20,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True
)
```

- Generación de los lotes (batches): Si utilizamos ImageDataGenerator de keras te facilita crear, redimensionar y transformar las imágenes.

```
validation_datagen = ImageDataGenerator(rescale=1.0/255)
```

4 - Construcción del modelo:

Vamos a crear un modelo CNN la cual es especialmente efectiva para problemas de visión por computadora. La CNN contiene las capas convolucionales y pooling para extraer las características relevantes.

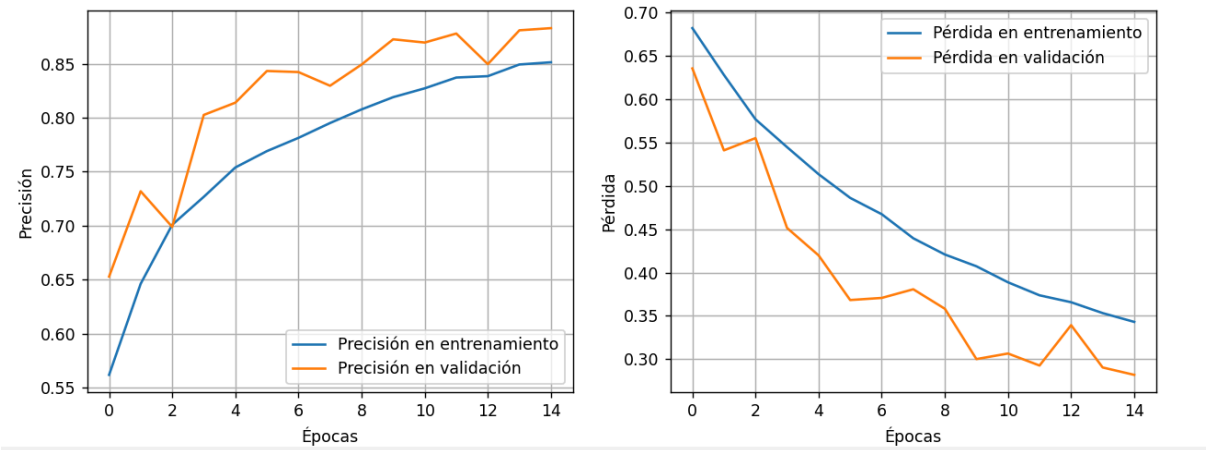
```
model = Sequential([
    Conv2D(32, (3, 3), activation='relu', input_shape=(image_size[0], image_size[1], 3)),
    MaxPooling2D(pool_size=(2, 2)),

    Conv2D(64, (3, 3), activation='relu'),
    MaxPooling2D(pool_size=(2, 2)),

    Conv2D(128, (3, 3), activation='relu'),
    MaxPooling2D(pool_size=(2, 2)),

    Flatten(),
    Dense(128, activation='relu'),
    Dropout(0.5),
    Dense(1, activation='sigmoid')
])
```

5 - Entrenamiento del modelo:

Hay que alimentar el modelo con las imágenes de entrenamiento. ESTE PROCESO VA A TARDAR y mientras que el proceso es completado se van ajustando los pesos para poder clasificar entre gatos o perros.

6 - Evaluación del modelo:

Luego de entrenar vamos a evaluar el modelo en el conjunto de validación para ver su precisión en imágenes que no había “visto antes” y vamos a graficar 2 parámetros la precisión y la pérdida.



Precisión: (Accuracy) es una medida (0-1) del porcentaje de efectividad de nuestro modelo para predecir. en otras palabras nos muestra que tan correcto está nuestro modelo para clasificar eso lo hace usando los datos “validate” un valor de 1 indica que nuestro modelo es perfectamente preciso.

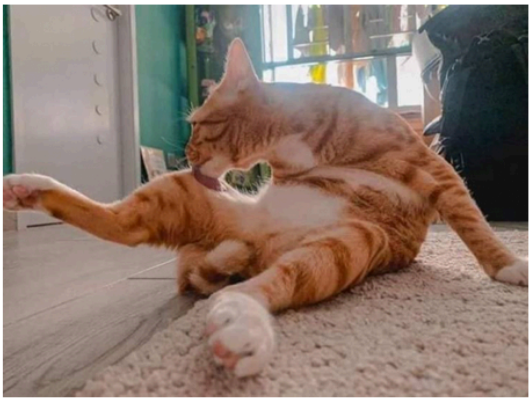
Pérdida: (Loss) es el valor promedio de errores cuadráticos, en otras palabras nos dice que tan lejos está nuestro modelo de la solución verdadera. Lo que se busca es un valor cercano a cero.

7 - Predicción:

Una vez que se completó el entrenamiento y tenemos el archivo de entrenamiento “modelo.keras” lo que tenemos que hacer es seleccionar cualquier imagen de un perro o un gato indicando la ruta y llamando al método “predict_image(path)”

```
def predict_image(img_path):
    img_array = preprocess_image(img_path)
    prediction = model.predict(img_array)
    if prediction[0] < 0.5:
        print("Es un gato 🐱")
    else:
        print("Es un perro 🐶")
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 126ms/step
Es un perro 🐶
```



```
def predict_image(img_path):
    img_array = preprocess_image(img_path)
    prediction = model.predict(img_array)
    if prediction[0] < 0.5:
        print("Es un gato 🐱")
    else:
        print("Es un perro 🐶")
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 132ms/step
Es un gato 🐱
```