SISTEMAS INTELIGENTES

CIM31 – Grupo Detección

Integrantes: Paul Brayan Rodríguez Isler, Diego Espinosa García y Manuel Neto Romero

MEMORIA DEL PROYECTO

En un principio decidimos hacer la práctica final sobre un sistema inteligente que sería capaz de identificar la raza que tiene un perro; sin embargo, tras más de una semana intentando sin éxito que acertara, vimos que la implementación estaba bien, pero por mucho que entrenara (tenía un porcentaje de éxito del 90% supuestamente), en muy pocas ocasiones daba la raza correcta. En un principio pensamos en reducir el número de razas a 15, pero el resultado era el mismo.

Finalmente llegamos a la conclusión de que el problema radicaba en la poca cantidad de dataset del que disponíamos por cada raza de perro (solamente unas 100 por raza) y de la perspectiva de estas (las fotos podían tener más de un perro u otros elementos que provocaban confusión). Es por ello que decidimos cambiar el tema a que distinguiera a un perro de un gato, ya que el dataset del que disponíamos era mucho mayor y su implementación era parecida.

Como nos parecía bastante simple la detección de un perro o un gato y no sabíamos si este cambio de tema iba a ser aceptado, decidimos hacer finalmente un sistema inteligente que fuera capaz de elegir entre más opciones y no solo entre dos (como perros o gatos): en este caso sería **capaz de detectar el tipo de flor entre cinco posibilidades (rosa, margarita, diente de león, girasol o tulipán) para que así fuera de mayor complejidad.**

IMPLEMENTACIÓN Y FUNCIONAMIENTO

A continuación vamos a explicar la implementación y el funcionamiento de la detección del tipo de flores (ya que el de perros vs gatos no requiere de mucha explicación y es mejor centrarse en el más importante).

Para ejecutar este programa, se puede usar un servidor web simple en Python. Hay que abrir una terminal en el directorio que contiene los archivos HTML y JavaScript y ejecutar el siguiente comando:
bash
Copy code
python -m http.server
Esto iniciará un servidor web en el puerto 8000. Puedes acceder a la aplicación abriendo tu navegador y visitando http://localhost:8000 en la barra de direcciones. Asegúrate de que tu modelo TensorFlow.js y tus archivos HTML/JavaScript estén en el mismo directorio.

Este código está diseñado para entrenar un modelo de red neuronal convolucional (CNN) utilizando el conjunto de datos "tf_flowers" de TensorFlow Datasets, que contiene imágenes de flores etiquetadas con diferentes razas. Luego, guarda el modelo entrenado y realiza la conversión a un formato compatible con TensorFlow.js para su uso en aplicaciones web. Aquí está una explicación detallada de cada parte del código:

Se importan las bibliotecas necesarias, como Matplotlib para visualización, NumPy para manipulación de datos, TensorFlow para construir y entrenar modelos, OpenCV para manipulación de imágenes, TensorFlow Datasets para cargar conjuntos de datos y LabelEncoder para codificar etiquetas.

Carga de datos:
python
Copy code
datos, metadatos = tfds.load('tf_flowers', as_supervised=True, with_info=True)

En este código, se carga el conjunto de datos "tf_flowers" con información supervisada y detalles del conjunto de datos almacenados en metadatos. Después, se visualizan las primeras 10

imágenes del conjunto de entrenamiento y se redimensionan las imágenes a un tamaño específico para almacenarlas en listas para su posterior procesamiento.

A continuación , las imágenes se normalizan dividiendo los valores de píxeles por 255 para estar en el rango [0, 1] y se define un **modelo secuencial de red neuronal convolucional con capas convolucionales, de pooling, de dropout y completamente conectadas.** Se compila el modelo con el optimizador Adam y la función de pérdida de entropía cruzada categórica dispersa.

El modelo se entrena con los datos de entrada y etiquetas, con un tamaño de lote de 32, una división del 15% para validación y durante 15 épocas.

Se define una función hacer_prediccion que carga y preprocesa una imagen, luego utiliza el modelo entrenado para hacer predicciones sobre la clase de la flor. Finalmente, el modelo se guarda en un archivo "modelo_flores.h5", y luego se realiza la conversión a formato TensorFlow.js. El modelo convertido se comprime en un archivo ZIP y se descarga para su uso en aplicaciones web.

¿Por qué hemos utilizado este modelo neuronal?

Las capas convolucionales de una CNN permiten detectar características locales en regiones específicas de la imagen, como bordes, texturas o patrones pequeños. Esto es crucial para reconocer detalles importantes en imágenes de flores.

Las capas de pooling en una CNN ayudan a lograr invarianza a pequeños desplazamientos y escalas. Esto significa que el modelo puede reconocer patrones incluso si aparecen en diferentes ubicaciones o tamaños en las imágenes.

Las capas convolucionales más profundas aprenden representaciones de características más complejas, construyendo una jerarquía de información. Esto es beneficioso para clasificar objetos complejos, ya que el modelo puede aprender a reconocer patrones más abstractos a medida que profundiza.

Las CNN están diseñadas para trabajar con datos bidimensionales, como imágenes, y aprovechan la estructura espacial de la información. Esto permite que el modelo capture relaciones espaciales entre píxeles, lo cual es esencial en tareas de visión por computadora.

La capa de Dropout en el modelo ayuda a prevenir el sobreajuste al apagar aleatoriamente neuronas durante el entrenamiento. Además, la regularización se incorpora mediante la función de pérdida "sparse_categorical_crossentropy", lo que contribuye a un modelo más generalizado.

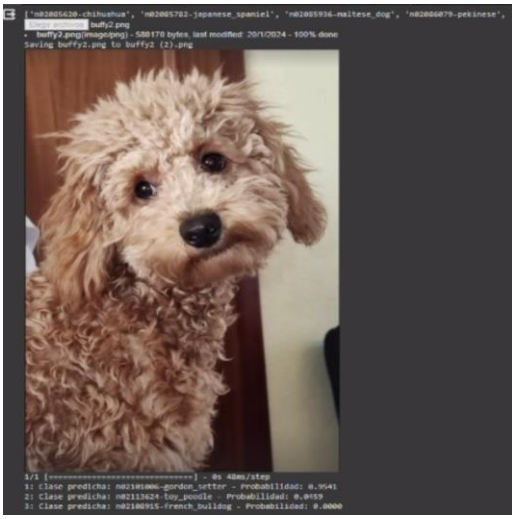
El conjunto de datos "tf_flowers" contiene imágenes de flores, y se ha especificado un tamaño de imagen de 100x100 píxeles. La arquitectura de la CNN se adapta a este tamaño de entrada.

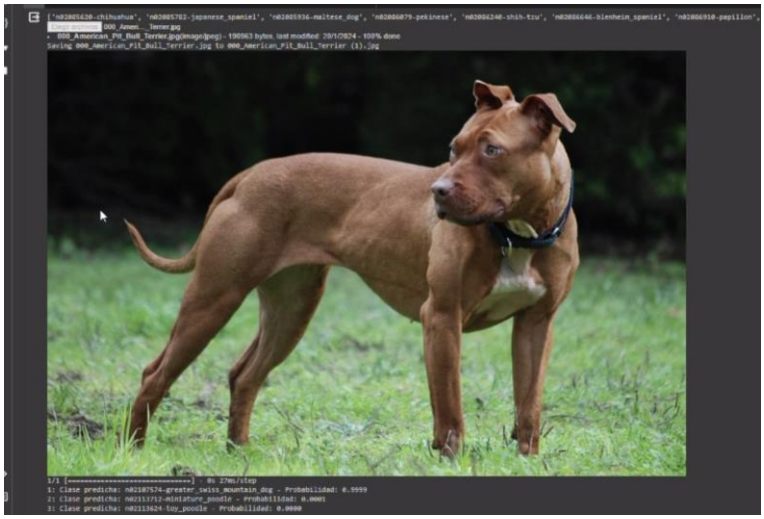
El conjunto de datos tiene cinco clases diferentes de flores. La última capa de la red tiene 5 neuronas con una función de activación softmax, lo que es apropiado para problemas de clasificación multiclase.

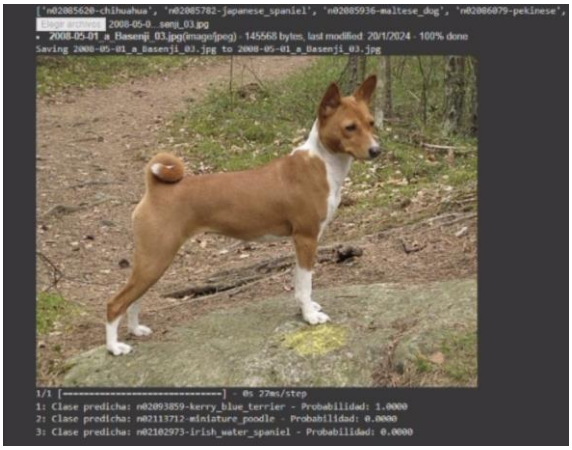
RESULTADOS

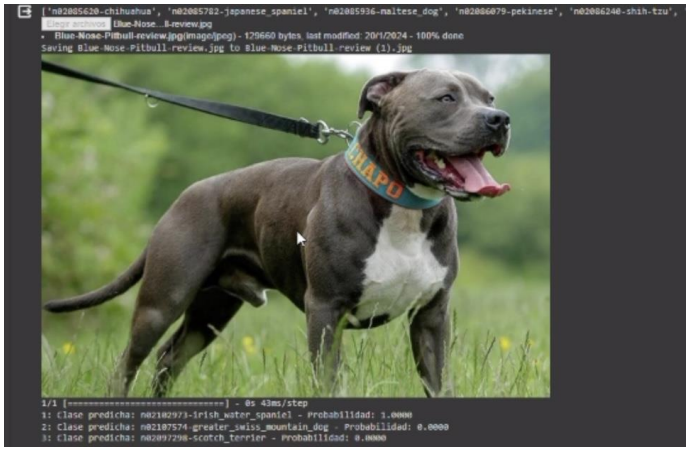
Respecto a los resultados obtenidos en los temas de tipos de flores y perros vs gatos, la tasa de acierto en las pruebas que hemos hecho es mayor al 90%, confundiéndose en casos bastante puntuales, como una imagen muy poco legible, una cantidad alta de flores distintas, etc…

En el caso de la detección de razas de perro, en las pruebas que recabamos se puede llegar a observar que la tasa de acierto era hasta menor al 5%, mostrando entre las 3 posibles razas un supuesto acierto que llegaba prácticamente al 100% cuando no era ninguna de las 3 posibilidades. A continuación adjuntamos hasta 4 ejemplos de los resultados que arrojaba donde, salvo en casos puntuales (como en el ejemplo 1 que sale la raza correcta como segunda opción), no sale la raza correcta en prácticamente ningún momento cuando hay una supuesta probabilidad de 1/15 y ha sido entrenado para mejorarla:









CONCLUSIONES

En conclusión, hace falta una cantidad de como mínimo 500 imágenes en el dataset para que pueda arrojar unos resultados decentemente correctos, además que si hay más variables posibles a elegir, la cantidad de imágenes será aún mayor. Es por esto que no conseguimos finalmente nuestra idea inicial de saber qué raza tiene un perro, ya que la implementación era la adecuada.

Algunas mejoras o ampliaciones podrían ser la posibilidad de decir si ha acertado en su resultado para que así esa resolución se añadiera a su entrenamiento y así complementarlo para que mejorara, además de, en el caso de los tipos de flores, añadir más de estas y que pueda decir que no es ninguna de las flores posibles.