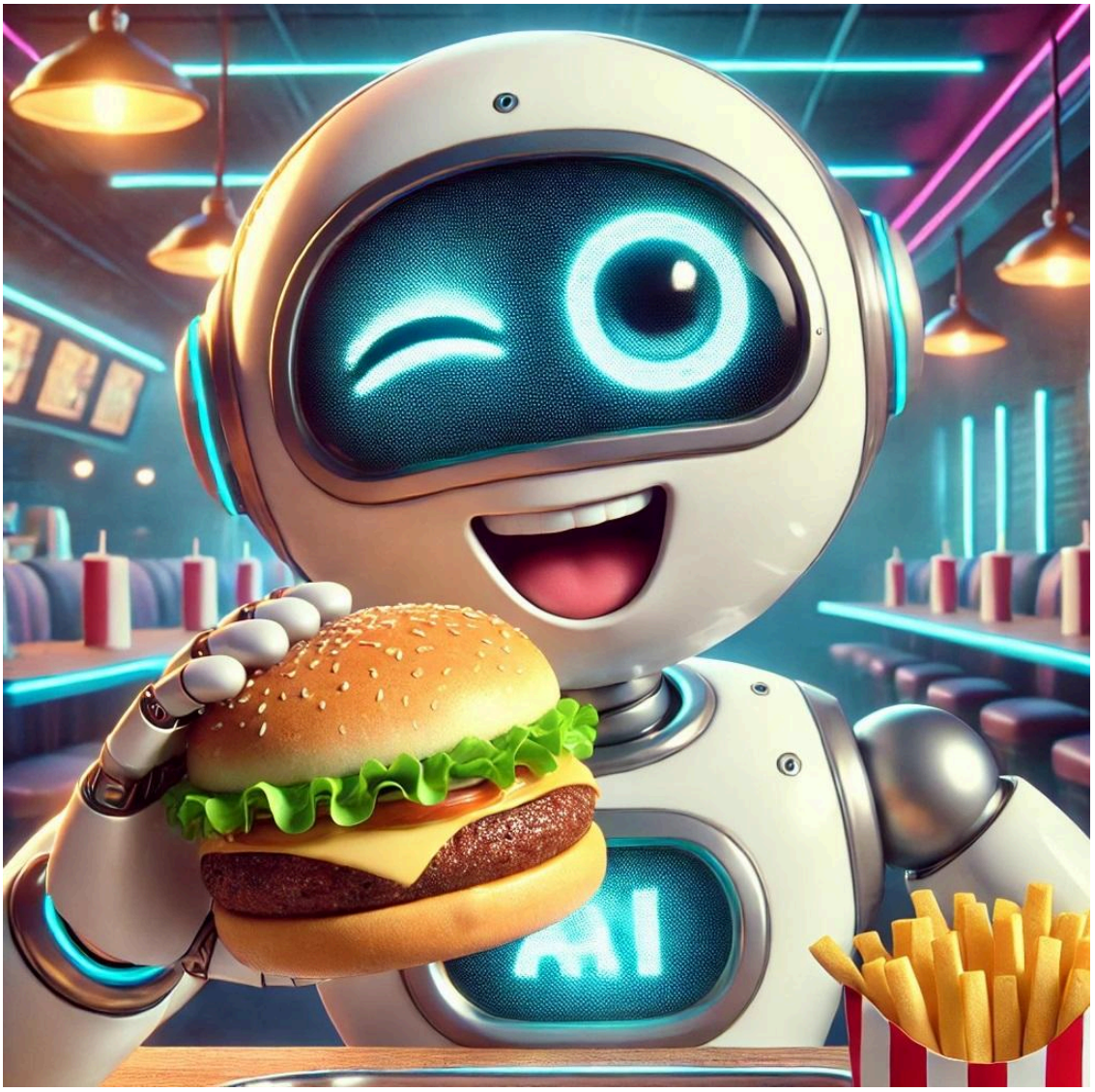
Proyecto: Clasificación de Platos, Valor Nutricional y Recetas

By: Adrián Madrid Romero



rclone copy "mi_drive:/content/drive/MyDrive/tokio/deep_learnin

g/proyecto_final_foodAi" "C:\Users\adria\Desktop" --progress --transfers=10 --fast-list

1. Introducción

Objetivo del Proyecto

El objetivo de este proyecto es desarrollar un sistema de clasificación de imágenes de platos de comida utilizando deep learning, para luego estimar su valor nutricional y generar recetas asociadas. Todo esto se integra en una API REST con una interfaz gráfica que facilita su uso.

El proyecto se compone de cuatro fases principales (divididas en 4 notebooks):

1. **Clasificación_platos**: Creación de un modelo de deep learning para identificar platos en imágenes.
2. **ValorNutricional_y_Recetas**: Obtención de información nutricional y recetas mediante una base de datos propia y APIs externas.
3. **Api_Rest**: Implementación de una API para acceder a las predicciones y datos nutricionales.
4. **Interfaz_Api**: Creación de una interfaz visual interactiva para mejorar la experiencia del usuario.

El nombre de estos puntos es el nombre de los cuadernos que se usaron para desarrollar el código.

2. Clasificación de Imágenes

Descripción del Dataset y Análisis Exploratorio

Se ha utilizado el dataset Food-101 de kaggle que contiene 101 clases de platos diferentes con un total de imágenes para train de 75.750 y 25.250 imágenes para

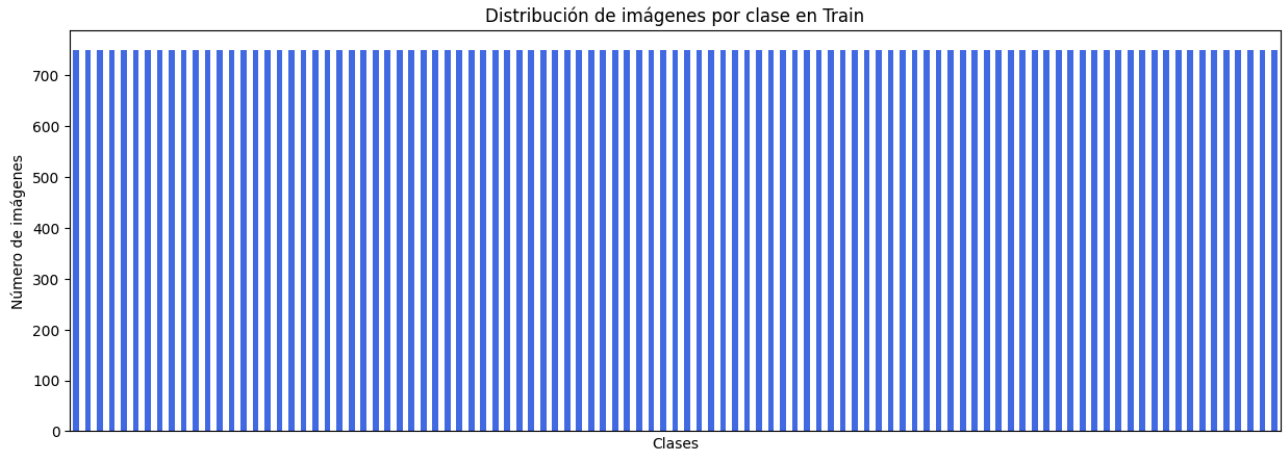
test. Se descargó y organizó en carpetas para facilitar su uso en el entrenamiento del modelo.

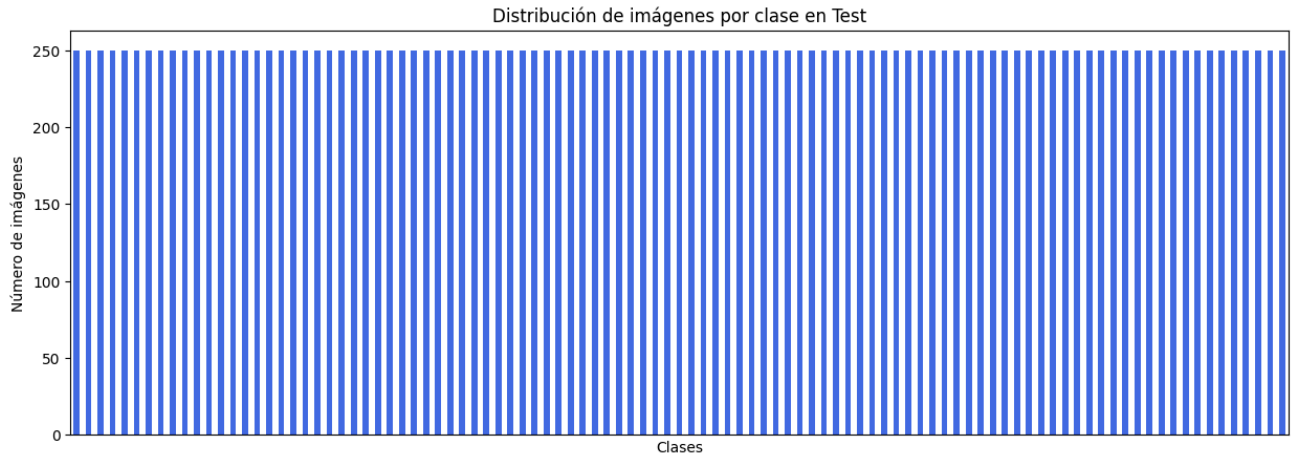
El dataset partía con 1000 imágenes de cada tipo de plato, lo cual es algo maravilloso a la hora de entrenar el modelo ya que no tendremos problemas de desbalance..

```
Imágenes aleatorias de Train:
```


```
Imágenes aleatorias de Test:
```

En cuanto al **balanceo de clases** está de la siguiente manera:





Todas las clases están balanceadas en ambos conjuntos, por lo que nos hace confiar más en que elegimos bien al quedarnos con el train y test de los creadores, ya que podíamos haber separado nosotros mismos el train y test pero la verdad es que ya venía hecho y bien organizadas las clases.

Preprocesamiento

Este punto fue crucial ya que antes de hacer el preprocesamiento, en nuestro cuaderno de colab se tuvo que cargar todas las imágenes tanto de train como de test a la Ram de Colab para cargar las imágenes más rápido durante el entrenamiento ya que contamos con un dataset bastante grande y esto nos facilitará acelerar el proceso.

PARTE CRUCIAL

```
#Copio dataset a RAM de Colab para acelerar la carga de imágenes
!cp -r "/content/drive/MyDrive/tokio/deep_learning/proyecto_final_foodAi/food-101-preprocesado" /content/
ruta_dataset = "/content/food-101-preprocesado"
```

En cuanto al preprocesamiento propio las partes principales fueron:

- **Reducción del dataset:** Se tuvo que usar únicamente el **50%** del conjunto de nuestros datos por motivos de tiempo y falta de recursos ya que con todo el conjunto tardaría una barbaridad en entrenarse y aunque solo se tenga la mitad, puede funcionar bastante bien.
- **Normalización de imágenes:** Todas las imágenes se escalaron entre 0 y 1 para mejorar el rendimiento del modelo
- **Redimensión de imágenes:** A 198x198, donde antes se partía de 224x224 pero supondría de nuevo más cálculos para el modelo y más tiempo de entrenamiento
- **Data Argumentation:** Donde se generan imágenes variadas para que el modelo no memorice los datos y prevenir el sobreajuste. Además aumentamos el dataset creando nuevas versiones sin necesidad de recopilar más datos. Para ello se simulan variaciones naturales como rotaciones, desplazamientos, zoom y otros cambios como ocurren en las fotos reales.

Creación del Modelo

Se compararon dos enfoques:

Modelo CNN desde cero

Se define primero un modelo convolucional con tres capas convolucionales ya que tiene que capturar muchos datos y a cada capa se le va agregando cada vez más filtros buscando cada vez que sea más preciso. Se agrega una capa de pooling a todas para ir reducciendo un poco la dimensionalidad y el posible sobreajuste, evitando mucho coste y poca generalización.

Modelo desde Cero:
Model: "sequential"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 222, 222, 32)	896
max_pooling2d (MaxPooling2D)	(None, 111, 111, 32)	0
conv2d_1 (Conv2D)	(None, 109, 109, 64)	18,496
max_pooling2d_1 (MaxPooling2D)	(None, 54, 54, 64)	0
conv2d_2 (Conv2D)	(None, 52, 52, 128)	73,856
max_pooling2d_2 (MaxPooling2D)	(None, 26, 26, 128)	0
flatten (Flatten)	(None, 86528)	0
dense (Dense)	(None, 256)	22,151,424
dropout (Dropout)	(None, 256)	0
dense_1 (Dense)	(None, 101)	25,957

Total params: 22,270,629 (84.96 MB)
Trainable params: 22,270,629 (84.96 MB)
Non-trainable params: 0 (0.00 B)

Transfer Learning (Mobilenet)

Una de las cosas más interesantes que se pueden hacer, en vez de construir un modelo desde cero, es construir un modelo preentrenado o ya existente, que se entrenó en su momento para un problema pero que con su arquitectura de capas y parámetros podemos adaptarlo a nuestro problema. Además, al estar preentrenado, no se tiene que empezar a entrenar todo desde cero y supone más rapidez que nos vendrá bien ya que contamos con muchos datos, y ya se puede intuir que necesitaremos muchas horas para construirlo desde cero.

Se utilizará el modelo preentrenado Mobilenet diseñado para ser eficiente en dispositivos con recursos limitados, donde en este caso tenemos un dataset enorme, y contamos con la cpu de colab, alternando con la gpu. Lo cual nos vendrá bien.

En este caso, se configura con include_top=False para eliminar la capa de clasificación original y agregar nuestras propias capas densas adaptadas a las 101 clases de nuestro dataset.

En cuanto al resto de capas:

-GlobalAveragePooling2D() :Reduce cada mapa de características a un único valor promedio por canal, comprimiendo la información sin necesidad de capas densas grandes, lo que ayudará a evitar el sobreajuste.

-Dense(256, activation="relu"): Capa densa con 256 neuronas para aprender patrones avanzados antes de la clasificación final

-Dropout(0.4): Queremos apaga aleatoriamente el 40% de neuronas para evitar sobreajuste.

-Por último otra capa densa (101) neuronas, una por cada clase a predecir y activación softax para predicción multiclase

Modelo con Transfer Learning:
Model: "sequential_1"

Layer (type)	Output Shape	Param #
mobilenetv2_1.00_224 (Functional)	(None, 7, 7, 1280)	2,257,984
global_average_pooling2d (GlobalAveragePooling2D)	(None, 1280)	0
dense_2 (Dense)	(None, 256)	327,936
dropout_1 (Dropout)	(None, 256)	0
dense_3 (Dense)	(None, 101)	25,957

Total params: 2,611,877 (9.96 MB)
Trainable params: 2,215,333 (8.45 MB)
Non-trainable params: 396,544 (1.51 MB)

Elección de modelo

¿CON CUAL QUEDARME?

CNN desde Cero vs **Transfer Learning** Tiempo de Entrenamiento

CNN Muy lento // **Transfer Learning** 5-10 veces más rápido

CNN Memoria Usada 85 MB (grande) **Transfer Learning** // 10 MB (pequeño)

CNN Capacidad de Aprender desde Cero // **Transfer Learning** Solo tiene conocimientos previos

CNN Precisión inicial baja (necesita más datos) // **Transfer Learning** Alta (usa MobileNetV2) **CNN** Overfitting Probable // **Transfer Learning** Menos probable

Al no tener demasiado tiempo y querer más eficiencia, Se elegirá el modelo de Transfer Learning (MobileNetV2), ya que irá más rápido y para este problema me puede ir igual de bien.

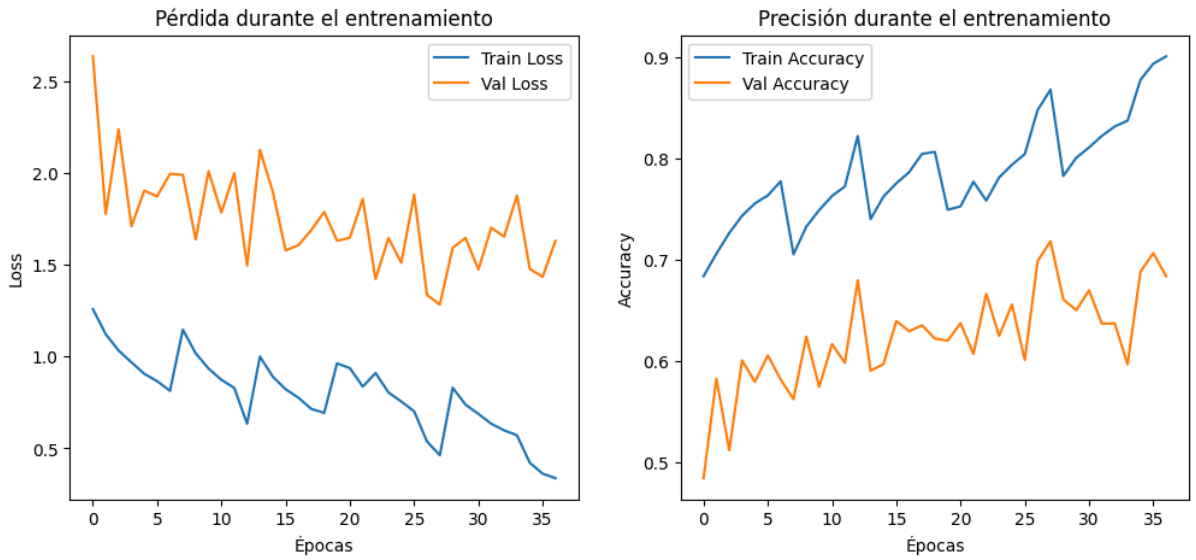
Solo se tendría que ir probando ciertos ajustes para buscar el mejor accuracy

Entrenamiento con Mobilenet y Evaluación

A la hora de entrenar este modelo (a veces con CPU, otras con la GPU de colab) lo que se hace es congelar algunas capas iniciales, para que no se entrene con todo el modelo preentrenado sino que coja y aprenda de las capas densas para la clasificación.

Después se guardan los modelos con el último mejor val_loss y esto es porque al entrenarse tanto tiempo debido al volumen de datos, puede desconectarse la sesión. Se utilizan callbacks para mejorar el entrenamiento, como EarlyStopping para evitar sobreajuste, ReduceLROnPlateau para ajustar la tasa de aprendizaje, TensorBoard para visualizar el entrenamiento y ModelCheckpoint para guardar los mejores pesos del modelo. Al final tanto el modelo como los callbacks se guardan en dos archivos llamados “mobilenet_food101_checkpoint.h5” y “mobilenet_food101_historial.json” que estan dentro de la carpeta del proyecto.

En resumen, el código ajusta un modelo MobileNetV2 para clasificar imágenes de comida, se guarda el progreso y el resultado final en Drive, y se emplean técnicas para un entrenamiento robusto y eficiente.



Las gráficas muestran que la pérdida de entrenamiento disminuye de forma estable, mientras que la pérdida de validación es más irregular, lo que sugiere posible sobreajuste. La precisión de entrenamiento mejora constantemente, pero la de validación fluctúa, indicando que el modelo no generaliza perfectamente. Para mejorar, se puede aumentar el Dropout, reducir la tasa de aprendizaje y usar EarlyStopping para evitar entrenar de más.

Prueba con una imágen de internet

Se carga el modelo de nuevo y evaluamos el test:

```
]

ruta_guardado = "/content/drive/MyDrive/tokio/deep_learning/proyecto_final_foodAi/mobilenet_food101_checkpoint.h5"
modelo_mejor = load_model(ruta_guardado)

print("Modelo cargado exitosamente.")
```

```
resultado = modelo_mejor.evaluate(test_generator)
print(f"Pérdida en test: {resultado[0]:.4f}")
print(f"Accuracy en test: {resultado[1] * 100:.2f}%")
```

```
198/198 ━━━━━━━━━━━━━━━━━━━━ 562s 3s/step - accuracy: 0.7033 - loss: 1.4387
Pérdida en test: 1.4324
Accuracy en test: 70.04%
```

Se obtiene una precisión del 70 por ciento sobre el test, nada mal teniendo en cuenta que se usó el 50 por ciento del conjunto finalmente

Después se creo un archivo dentro de la carpeta del proyecto llamada utils.py donde importaremos una función creada para predecir el plato a través de esa imagen obteniendo un resultado como este:

```
ruta_imagen = "/content/drive/MyDrive/tokio/deep_learning/proyecto_final_foodAi/imagenes_prueba/pizzacorazon.jpg"
nombre_predicho = predecir_imagen(ruta_imagen, modelo_mejor)
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 70ms/step
```

Predicción: Pizza

3. Base de Datos de Información Nutricional y Recetas

Obtención de Datos Nutricionales

Para estimar la información nutricional, se utilizó la API de FatSecret, que proporciona detalles sobre calorías, macronutrientes y otros valores. Se automatizó la consulta de datos y su almacenamiento en un dataset propio.

API de fastsecret: https://www.fatsecret.es/Default.aspx?pa=m

Creación del Dataset

Los datos recopilados fueron almacenados en un archivo CSV estructurado con las siguientes columnas:

- Nombre del plato
- Calorías
- Grasas
- Carbohidratos

- Proteínas
- Ingredientes
- Enlace a receta

	Plato	Nombre API	Calorías (kcal)	Grasas (g)	Carbohidratos (g)	Proteínas (g)	Ingredientes	URL_Receta
0	Apple pie	Apple Pie	265.0	12.50	37.10	2.40	- apples (2 Red apples, cored, peeled and cut ...	https://food52.com/recipes/55787-apple-pie-smo...
1	Baby back ribs	Baby Back Ribs	210.0	15.00	4.00	17.00	- baby back ribs (1 or 2 racks baby back ribs,...	https://www.epicurious.com/recipes/food/views/...
2	Baklava	Baklava	5246.0	355.81	461.09	82.12	- granulated sugar (2 cups granulated sugar (1...	https://www.seriouseats.com/pistachio-baklava-...
3	Beef carpaccio	Carpaccio	119.0	2.62	0.00	22.31	- olive oil (2 tbsp. olive oil)\n- Parmesan ch...	https://www.delish.com/cooking/recipe-ideas/re...
4	Beef tartare	Steak Tartare (Raw Ground Beef and Egg)	210.0	14.56	0.63	17.78	- sirloin (1 1/2 pounds fatty sirloin or chuck...	http://www.foodrepublic.com/2014/05/07/beef-ta...
...	...	...	...	...	...	...	...	...
96	Tacos	Taco or Tostada with Beef, Cheese and Lettuce	221.0	12.65	16.20	10.88	- Vegetable Oil (2 tsp Vegetable Oil (picadill...	http://www.lottieanddoof.com/2009/07/picadillo/
97	Takoyaki	Takoyaki	1489.0	64.32	151.25	71.89	- oyster sauce (1 tablespoon oyster sauce)\n- ...	https://www.allrecipes.com/recipe/283774/takoy...
98	Tiramisu	Tiramisu	3944.0	253.62	340.15	66.47	- Crepes (Crepes (Makes 20 crepes))\n- butter ...	https://food52.com/recipes/5286-tiramisu-crepe...
99	Tuna tartare	Tuna	108.0	0.95	0.00	23.38	- mayonnaise (1 tablespoon mayonnaise)\n- soy ...	https://www.marthastewart.com/355745/spicy-tun...
100	Waffles	Plain Waffles	291.0	14.10	32.90	7.90	- nonfat Greek yogurt (8 tablespoons nonfat Gr...	https://www.bonappetit.com/recipe/waffles-with...

.

Para ello se obtendrá información nutricional de platos identificados por el modelo de clasificación de imágenes, utilizando la API de FatSecret. Se cargarán las credenciales desde el archivo .env, extrayendo CONSUMER_KEY y CONSUMER_SECRET para la autenticación.

Luego, mediante una función una función se generarán los parámetros de autenticación OAuth, incluyendo la firma HMAC-SHA1, necesaria para que la API valide la solicitud.

A continuación, se itera sobre cada plato en la lista clases_modelo (sacada del archivo creado en la carpeta del proyecto llamada “`clases_food101.npy”),` realizando una consulta a la API para obtener información nutricional. Una vez recibida la respuesta, intentará seleccionar el primer resultado de tipo "Generic", que representa la versión más estándar del alimento, pero en caso de no encontrarlo, usará el primer resultado disponible.

Si la API devuelve una descripción del alimento, la función parse_nutrition() extrae los valores de calorías, grasas, carbohidratos y proteínas mediante expresiones regulares.

Toda esta información se almacena en una lista de diccionarios donde se registran los datos del plato original, el nombre devuelto por la API, los valores nutricionales y el enlace a la página de la API correspondiente.

Si la consulta no arroja resultados, se almacenan valores None en su lugar. Además, se incluyen medidas para manejar errores de conexión o respuestas vacías de la API, y se introduce un time.sleep(1) entre cada consulta para evitar hacer demasiadas peticiones en poco tiempo y así prevenir bloqueos por parte de la API.

Por último todos los datos recolectados se convierten en un DataFrame de Pandas y se guardan en un archivo CSV en Google Drive bajo el nombre food_nutrition_data.csv, para así tener con una base de datos estructurada con información nutricional de los platos analizados.

El resultado que queda probando con una imagen que elegimos (todas las imágenes de prueba están en una carpeta del proyecto llamada “imagenes_prueba”) y utilizando una función creada en utils.py sería:

Probando con una imágen

```
test_image_path = "/content/drive/MyDrive/tokio/deep_learning/proyecto_final_foodAi/imagenes_prueba/rissoto.jpg"
predecir_plato_y_nutricion(test_image_path, modelo, clases, df_nutricion)
```

Predicción: Risotto

```
Información nutricional de Risotto:
   Plato     Nombre API  Calorías (kcal)  Grasas (g)  Carbohidratos (g)  Proteínas (g)
Risotto Cheese Risotto           1382.0       50.94              171.8          54.44
```

Para **LA RECETA** se usará otra API. Se agrega una columna de ingredientes y otra con el url de la receta a nuestra base de datos ya existente.

Utilizaremos la API de **Edadam** donde su objetivo principal es facilitar el acceso a información nutricional y herramientas para analizar y crear recetas saludables.

URL --> https://www.edamam.com/

Se hará una función que haga llamadas a la API utilizando nuestras claves, se vea de que si la petición está ok me devuelve un json con las respuestas que esperamos. Verificaremos con un código si hay recetas en la respuesta y de ser así se formatean un poco para que se vea de manera bonita y estilosa (y no un json). Probaremos con algo sencillo,y probamos con "carbonara" que nos saldrá alguna receta de pasta o algo similar relacionado con ese nombre de plato.

```python
import requests

# Función para obtener ingredientes y URL de receta
def get_receta(plato):
    url_busqueda = f"https://api.edamam.com/api/recipes/v2?type=public&q={plato}&app_id={app_id}&app_key={app_key}"

    headers = {
        "Edamam-Account-User": user_id
    }

    response = requests.get(url_busqueda, headers=headers)

    if response.status_code == 200:
        data = response.json()

        if "hits" in data and len(data["hits"]) > 0:
            receta = data["hits"][0]["recipe"]
            ingredientes = "\n".join([f"- {ing['food']} ({ing['text']})" for ing in receta["ingredients"]])
            url_receta = receta["url"]
            return ingredientes, url_receta
    return "No disponible", "No disponible"


plato = "carbonara"
ingredientes, url_receta = get_receta(plato)

print("🥦 Ingredientes:")
print(ingredientes)
print("\n🔗 URL de la receta:")
print(url_receta)
```

```
🥦 Ingredientes:
- Pecorino Romano (50 grams Pecorino Romano (1.8 ounces) grated)
- egg (1 large egg)
- olive oil (2 tablespoons olive oil)
- black pepper (fresh ground black pepper to taste)
- Pancetta (170 grams Guanciale (6 ounces) Pancetta or Bacon will also work)
- pasta (170 grams dried pasta (6 ounces) boiled according to package directions)
- eggs (2 slow cooked eggs (optional))

🔗 URL de la receta:
http://norecipes.com/blog/spaghetti-carbonara-recipe/
```

Y sabiendo que funciona se ejecuta el siguiente código para aplicarlo a todo el dataframe:

Sabiendo que funciona podemos agregar la columna de la url de la receta y otra de los ingredientes

```python
[ ] #Aplico la función a la columna "Plato" y expando en dos nuevas columnas
    df_nutricion[["Ingredientes", "URL_Receta"]] = df_nutricion["Plato"].apply(lambda plato: pd.Series(get_receta(plato)))

    # Guardar el dataset actualizado
    df_nutricion.to_csv(ruta_nutricion, index=False)
    print("Dataset actualizado con ingredientes y URLs de recetas.")
```

```
Dataset actualizado con ingredientes y URLs de recetas.
```

Quedando el dataframe de esta manera ya vista anteriormente:

	Plato	Nombre API	Calorías (kcal)	Grasas (g)	Carbohidratos (g)	Proteínas (g)	Ingredientes	URL_Receta
0	Apple pie	Apple Pie	265.0	12.50	37.10	2.40	- apples (2 Red apples, cored, peeled and cut ...	https://food52.com/recipes/55787-apple-pie-smo...
1	Baby back ribs	Baby Back Ribs	210.0	15.00	4.00	17.00	- baby back ribs (1 or 2 racks baby back ribs,...	https://www.epicurious.com/recipes/food/views/...
2	Baklava	Baklava	5246.0	355.81	461.09	82.12	- granulated sugar (2 cups granulated sugar (1...	https://www.seriouseats.com/pistachio-baklava-...
3	Beef carpaccio	Carpaccio	119.0	2.62	0.00	22.31	- olive oil (2 tbsp. olive oil)\n- Parmesan ch...	https://www.delish.com/cooking/recipe-ideas/re...
4	Beef tartare	Steak Tartare (Raw Ground Beef and Egg)	210.0	14.56	0.63	17.78	- sirloin (1 1/2 pounds fatty sirloin or chuck...	http://www.foodrepublic.com/2014/05/07/beef-ta...
...	...	...	...	...	...	...	...	...
96	Tacos	Taco or Tostada with Beef, Cheese and Lettuce	221.0	12.65	16.20	10.88	- Vegetable Oil (2 tsp Vegetable Oil (picadill...	http://www.lottieanddoof.com/2009/07/picadillo/
97	Takoyaki	Takoyaki	1489.0	64.32	151.25	71.89	- oyster sauce (1 tablespoon oyster sauce)\n- ...	https://www.allrecipes.com/recipe/283774/takoy...
98	Tiramisu	Tiramisu	3944.0	253.62	340.15	66.47	- Crepes (Crepes (Makes 20 crepes))\n- butter ...	https://food52.com/recipes/5286-tiramisu-crepe...
99	Tuna tartare	Tuna	108.0	0.95	0.00	23.38	- mayonnaise (1 tablespoon mayonnaise)\n- soy ...	https://www.marthastewart.com/355745/spicy-tun...
100	Waffles	Plain Waffles	291.0	14.10	32.90	7.90	- nonfat Greek yogurt (8 tablespoons nonfat Gr...	https://www.bonappetit.com/recipe/waffles-with...

101 rows × 8 columns

4. Desarrollo de la API REST

Implementación con FastAPI

Se desarrolló una API REST con FastAPI para exponer el modelo y la base de datos nutricional. Se definieron los siguientes endpoints:

- `/predict`: Recibe una imagen y devuelve la clasificación del plato.

Puntos Clave de la API

1. **Carga del modelo y dataset**: Se importa el modelo entrenado y la base de datos nutricional.
2. **Procesamiento de imágenes**: Se reciben imágenes y se convierten en el formato adecuado para la predicción.
3. **Predicción y respuesta**: Se ejecuta el modelo sobre la imagen y se obtiene el plato clasificado.
4. **Integración de información nutricional y recetas**: Se enlazan los datos predichos con la base de datos.
5. **Exposición de la API con FastAPI y Ngrok**: Se utiliza Ngrok para hacer la API accesible desde cualquier dispositivo.

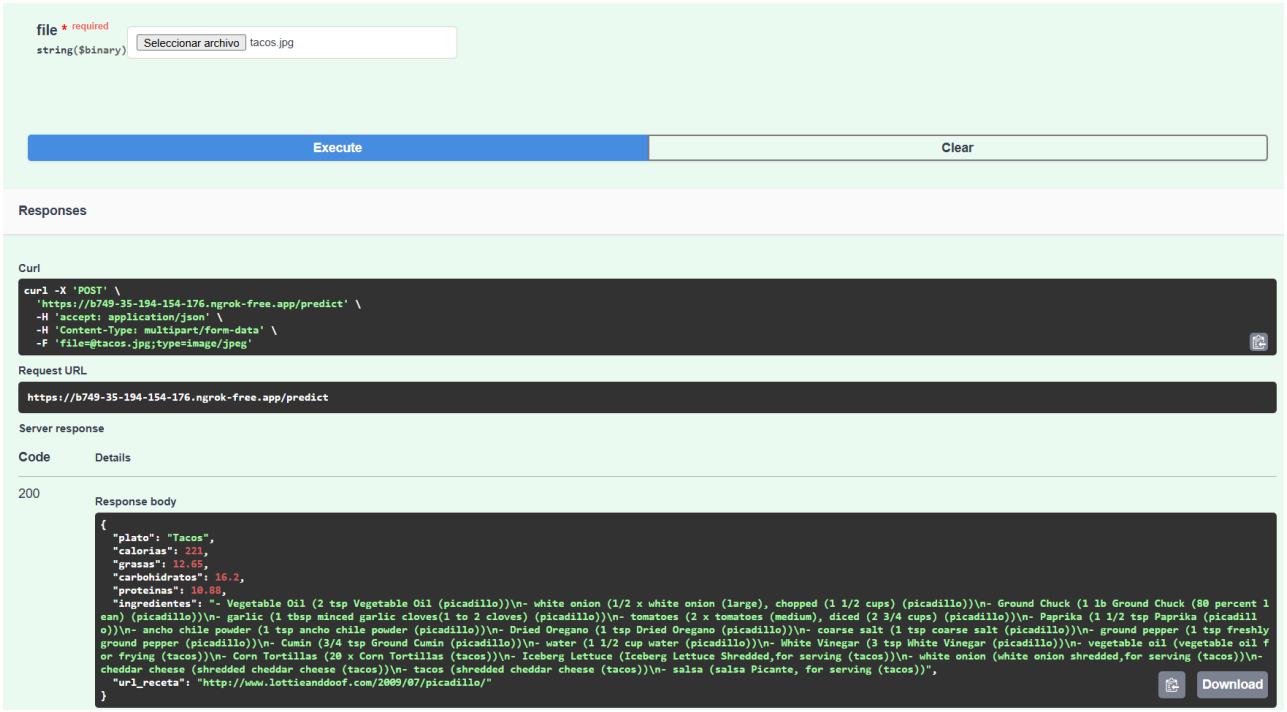
Librerías Utilizadas

- **FastAPI**: Framework para construir la API de forma rápida y eficiente.
- **TensorFlow/Keras**: Para la carga y uso del modelo de clasificación.
- **Pandas**: Para la manipulación y consulta del dataset de información nutricional.
- **OpenCV**: Para la lectura y preprocesamiento de imágenes.
- **Pyngrok**: Para exponer la API en la web a través de un túnel seguro.
- **Uvicorn**: Servidor ASGI para ejecutar la API de FastAPI.

Cuando se mandan peticiones a la API en la misma celda donde está corriendo podemos ver lo que va devolviendo la misma pero en formato de texto adaptada a colab, ya que con aplicaciones como Postman o en la misma Interfaz que ofrece FastApi utilizando URI/docs, donde se prueban las peticiones salen en formato json.

Así aparece lo que devuelve la API en colab:

Así devuelve las llamadas utilizando la URL/docs en un formato json (que suele ser el normal)



:

5. Interfaz Gráfica

Implementación con Streamlit

Para darle un nivel superior a este proyecto se diseñó una interfaz con Streamlit para mejorar la interacción del usuario. La interfaz permite:

-Subir una imagen para predicción.

-Ver el nombre del plato detectado.

-Consultar la información nutricional.

- Obtener una receta con enlace directo.

Se integró la API de predicción en la interfaz, asegurando que cada consulta muestre la información en un formato atractivo.

Es importante aclarar que cuando se ejecuta la API de nuevo, te genera una URL para acceder a ella que es la misma URL que se debe poner en el código de la interfaz para que se conecte con ella, y cada vez que se genere una nueva URL en la api por volver a correrla hay que volver a cambiar la URL en la interfaz para que no haya problemas de configuración y de conectividad.

En la carpeta del proyecto se encuentra un video utilizando la interfaz de Streamlit para hacer peticiones a la API y que devuelve el plato, el valor nutricional, ingredientes y la url de la receta. El video se llama video_proyecto_final_prueba

6. Conclusión y Mejoras Futuras

En este último apartado y hablando en primera persona, me detendré un poco más para hablar de todas las dificultades que tuve en este proyecto y de cómo podría encaminarlo de cara al futuro para que pueda mejorarlo.

ESCALABILIDAD Y FUTURO DEL PROYECTO

Sin duda, la idea del proyecto es interesante, y que este proyecto es una base de como podría llegar a serlo en el futuro, no solo contando con 101 platos, sino con miles de platos diferentes.

Al final, sería tener una app que sea capaz de predecir cualquier tipo de plato, incluso alguno que no se conozca del todo pero que el modelo sepa predecir en función de todos los elementos el tipo de plato que es, su valor nutricional (obviamente más acertado que obteniendolo de la API de fast secret), incluso la receta que en vez de sugerir únicamente una, sugiera varias.

Al final, el dataset que creamos con los valores nutricionales y las recetas de google de esos platos o similares solo eran de los platos que podía predecir el modelo y nada más, pero en un futuro se podría llegar a tener cientos de platos nuevos con valores nutricionales más exactos, y como he dicho antes múltiples recetas solamente haciendo una foto de un plato cualquiera y subiendolo a la API que tenemos creada.

DIFICULTADES Y POSIBLES MEJORAS

Una de las dificultades más grandes con las que tuve que lidiar en este proyecto fue el tamaño del dataset y la falta de recursos. Dependía de los recursos de google colab (bastante limitados si no pagas), ya que mi intención de primeras fue entrenar un modelo de convolución desde cero. Me di cuenta de que era una locura que me llevaría días y dinero. El dataset era enorme y no disponía de lo necesario para entrenarlo, por lo que decidí usar Transfer Learning, con un modelo preentrenado y rápido como fue Mobilenet.

También tuve que deducir a la mitad el conjunto del dataset, así como data argumentation, y el tamaño de los píxeles. De esta manera me pude permitir entrenarlo alrededor de 30 épocas, y guardando el modelo cada vez que mejoraba. De esta manera, si se me desconectaba el entorno (ya sea utilizando la GPU de colab o CPU , mucho más lenta) me curaba en salud y podía volver a reentrenarlo sin perder los pesos. Aunque igual, de esta manera tardé largas horas en entrenarlo.

Con la API hubo dificultades aunque menos, ya que llevo un tiempo configurarla y poder hacer que funcione cacharreando con el código, ya que requería que tanto la interfaz como la misma API estuviera todo bien configurado para que encontrara la página.

Asique, dicho esto, como posibles mejoras y si quisiera el día de mañana hacer una app profesional con todo lo descrito al principio, necesitaria o bien usar un modelo de transfer learning más robusto como ResNet, congelar más capas, o incluso construir mi modelo desde cero para que se adapte bien al problema pero controlando el sobreajuste claro está.

También usaría todo el conjunto del dataset, así como un tamaño de pixel más grande (224x224) por ejemplo, y generar más imágenes artificiales con data argumentation. Solo así podría hacer que mi score llegara a quizá un 90 por ciento.

Obviamente se podría mejorar la interfaz de manera más profesional y que devuelva más recetas similares a esos platos y con los valores nutricionales más concretos, es decir, al final hice que buscara valores nutricionales genéricos en la mayoría, pero en los platos puede haber muchas variantes de ingredientes y lo que predice mi modelo son platos muy clásicos, además que se enfoca en el plato a lo general y no en el plato y los ingredientes que lleva.

Por último, como reflexión final, diré que la tecnología está avanzando muy rápido y que cada vez los proyectos de esta tipología se resolverán de manera más rápida con ayuda de las IAS y las herramientas con las que disponemos hoy en día. La tecnología y la IA están en nuestras vidas para hacernos la vida más cómoda, y no hay que verla como una enemiga que se encargará de eliminar la mano del ser humano, sino que la transformará hacia otro nivel donde cada vez con poco haremos mucho.