



# **MDS+ 2021/2022**

## **TRABAJO FIN DE MÁSTER**

### Clasificador Multiclasificación de Imágenes de Restauración

Madrid, octubre de 2022

**Autores:** Pedro Tavares

Toni Vila

Carlos Cejas

Carlos Huguet

**Tutor:** Javier de la Rosa

# Resumen

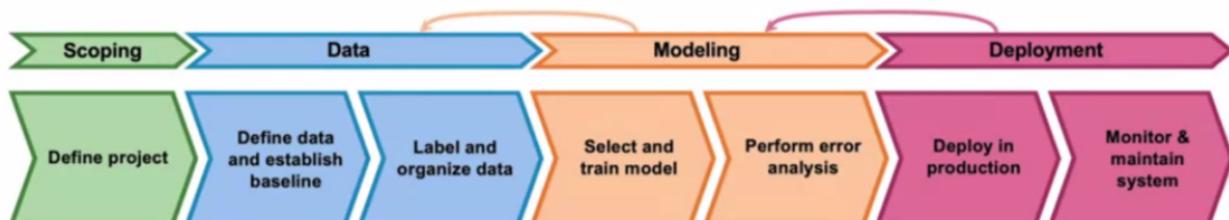
El presente trabajo fin de Máster persigue el objetivo de diseñar e implementar un sistema de clasificación de imágenes de restauración segmentadas en cinco clases distintas. Dichas imágenes proceden de una base de datos creada mediante técnicas de *scraping* sobre redes sociales. El fin último de la clasificación consiste en facilitar el proceso de decisión de los clientes finales en su elección de locales o tipos de comida.

La clasificación de imágenes consiste en la tarea de asignar una etiqueta a una imagen a partir de un conjunto predefinido de categorías, esto significa que nuestra tarea es analizar una imagen de entrada y devolver una etiqueta que categorice la imagen. Esta tarea de clasificación que para un humano es relativamente sencilla, plantea diversos desafíos para un modelo de *Deep Learning* al tener que enfrentarse a variaciones del punto de vista de la imagen, diversas condiciones de iluminación, deformaciones de los objetos, occlusiones parciales, alteraciones del fondo, variaciones de la escala, o variaciones intra clase.

La técnica escogida para la resolución del problema ha sido la confrontación de diversos modelos basados en redes convolucionales profundas creadas desde cero frente a arquitecturas ya preexistentes y, por último frente, a arquitecturas que integran *Transfer Learning* como Vison Transformer (ViT) o ResNet.

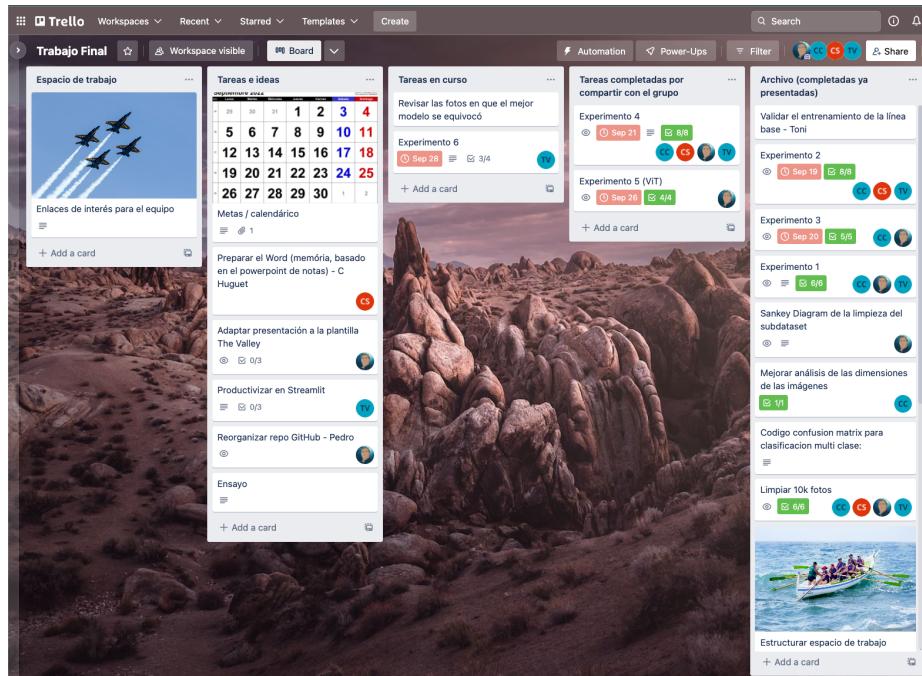
# Proceso

Se utilizó CRISP-DM como marco de trabajo, un proceso iterativo compuesto para las etapas siguientes, formando la base de un proceso científico de datos:



- A. Entender problema de negocio y objetivo
- B. Traducirlo en un problema de *Data Science*
- C. Explorar datos (EDA)
- D. Preparar datos
- E. Modelar/evaluar
- F. Despliegue

La coordinación del trabajo se hizo usando Scrum, con sprints semanales, reuniones de planning al inicio de la semana, dailies en cada reunión (clases y contactos adicionales) y reviews, soportados en un tablero kanban diseñado en la herramienta Trello.



Tablero kanban en Trello

## A. Problema de negocio

Los clientes de nuestra cadena suben a las redes sociales imágenes tanto de los interiores como de los exteriores de los locales, de su oferta en comida o bebida y de los menús. Toda esta ingente información posee un enorme potencial para dar a conocer la cadena y su variada oferta, facilitando el proceso de decisión tanto de su base de clientes actual como de los potenciales a alcanzar mediante la comunicación.

El problema radica en la absoluta desestructuración de toda esa información, lo que la hace inutilizable actualmente. El objetivo por tanto es clasificar dichas imágenes para integrarlas de forma adecuada en los canales de comunicación de la cadena.

## B. Problema de Data Science

El problema es de clasificación multi-clase [1], ya que cada foto pertenece solamente a una de las cinco categorías establecidas.

El objetivo es maximizar el accuracy total (número de predicciones correctas sobre el total de predicciones), sin definir un umbral de referencia.

### **Stack tecnológico utilizado:**

• Python3	• TensorFlow/Keras
• Contenedor Docker (Ubuntu)	• Scikit learn
• Google Colab/ Jupyter Notebook	• Torchvision
• Git / GitHub / HuggingFace Spaces	• Pillow / Open-cv
• MLflow	• Hugging Face
• IDEs: VSCode & PyCharm	• Streamlit / AWS EC2

### **Highlights sobre el stack tecnológico:**

#### **Repositorio de código**

Todo el código del trabajo está disponible en GitHub [2] y está preparado para ser ejecutado tanto en Colab como en un Jupyter Notebook.

#### **Contenedores Docker**

Emplearemos contenedores (Docker) porque nos permite realizar el empaquetado de nuestras aplicaciones junto a sus correspondientes dependencias dentro de unidades estandarizadas, y así poder por una parte ejecutar nuestro código en distintos ordenadores evitando problemas con librerías, y por otra parte, monitorizar los recursos del ordenador destinados al entrenamiento de los distintos modelos.

#### **MLflow**

Dados los múltiples experimentos que se desarrollarán sobre la base de la combinación de distintas arquitecturas, distintos Datasets y diferentes hiper parámetros es fundamental la utilización de la API de MLFlow Tracking para registrar los diferentes parámetros, las versiones de código, las métricas y ficheros de salida para posteriormente poder comparar experimentos.

### Google Colab Pro

Adicionalmente y **dada la aceleración por GPU**, se ha utilizado Google Colab Pro para poder reducir los tiempos de entrenamiento de los distintos modelos frente a los ordenadores personales.

## C. Exploración de Datos (EDA)

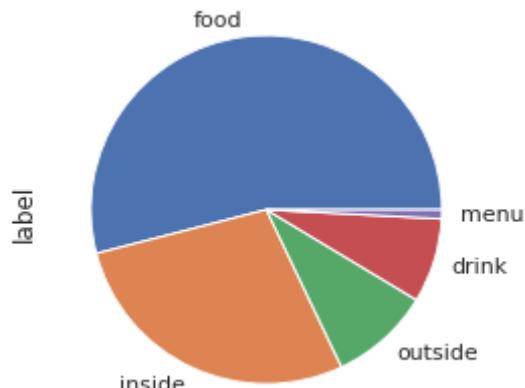
El análisis exploratorio (EDA) del dataset proporcionado, compuesto por imágenes con una sola etiqueta, es diferente del EDA de un dataset tabular típico, con muchas features, porque una imagen tiene pocos *features* relevantes (dimensiones, canales, por ej.) y los datos (pixeles) son uniformes (mismo datatype). Se podría ampliar el abanico de técnicas utilizadas en el análisis, pero los profesores y el tutor que acompañaron el trabajo confirmaron la percepción del equipo que tal no contribuiría de modo relevante al trabajo.

### Análisis de las imágenes:

El dataset proporcionado tiene un total de **200.100 imágenes** de las cuales **106 están corruptas** (fichero inválido). Dichas imágenes están etiquetadas bajo las clases (*labels*):

- **Inside**
- **Outside**
- **Menu**
- **Food**
- **Drink**

Adicionalmente a la carpeta de imágenes, el dataset incluye dos ficheros JSON, uno para train (180.094 fotos) y otro para test (20.006), que identifican la categoría de cada foto.



### C.1) Distribución etiquetas

El **54%** de las imágenes corresponden a la clase **Food**, **Inside supone un 28%**, seguido de **Outside con un 9%**, **Drink con un 8%** y **Menú**, que solo alcanza el **1%** del total de imágenes del dataset.

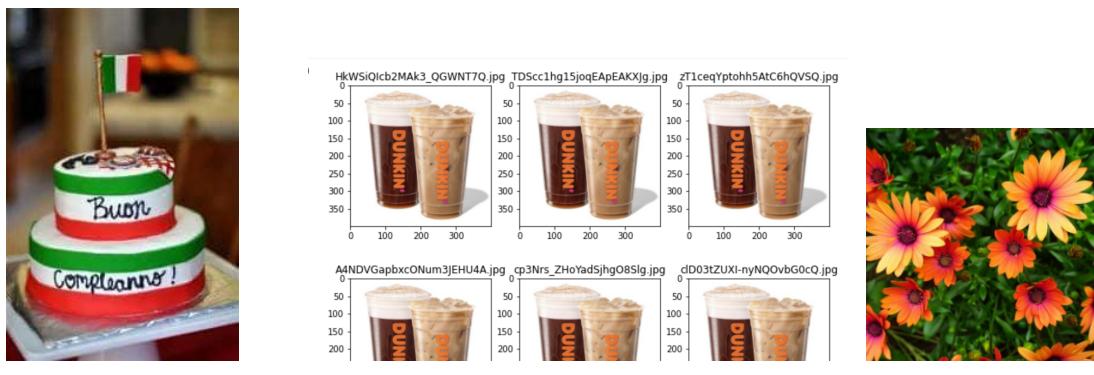
Es evidente el desbalanceo en el Dataset, que contiene clases muy poco representadas, alcanzando su máxima expresión en la clase Menú con solo el 1% de las imágenes disponibles. Esto puede dificultar el aprendizaje de los modelos en las clases peor representadas, por lo que habrá que evaluar su impacto y

valorar recurso a Data Augmentation (ampliación de imágenes, generando nuevas imágenes partiendo de las existentes, con rotaciones, traslados y cambios de color, por ejemplo).

### C.2) Análisis de las imágenes: Errores de clasificación

Durante la exploración inicial, se detectan múltiples errores de etiquetado, es decir clase no correspondiente a la imagen, imágenes duplicadas e irrelevantes, fuera de contexto tales como instalaciones industriales, mercados, parques o centros comerciales.

Será necesario **hacer una revisión manual para corregir los errores de etiquetado y eliminar las fotos irrelevantes**, así como **eliminar fotos duplicadas**.



**Mal etiquetada**

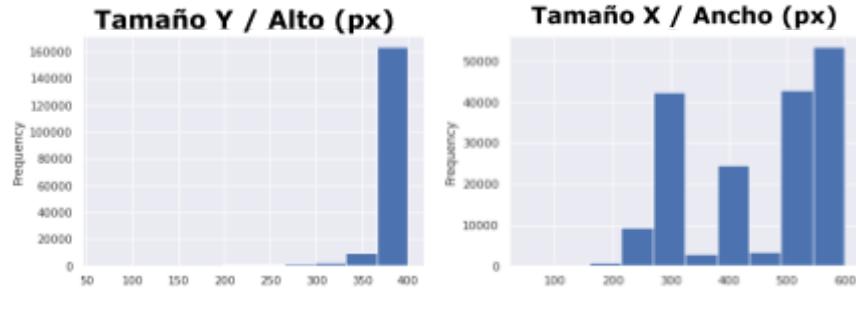
**Duplicada**

**Irrelevante**

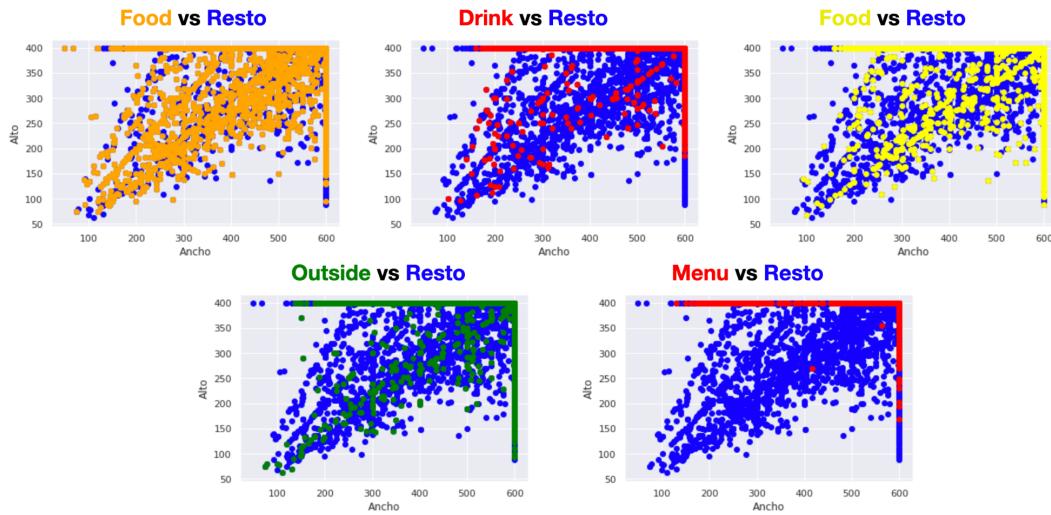
### C.3) Análisis de las imágenes: Tamaño de las imágenes

El tamaño de las imágenes se sitúa en el rango 62x49 a 400x600 (entre 5.625 y 240.000 píxeles), siendo 400x600, la resolución más común.

Tal y como se puede apreciar en los scatterplots, las fotos han sido procesadas en origen para no exceder 400x600. La distribución del tamaño por label es similar, excepto en la clase menu.



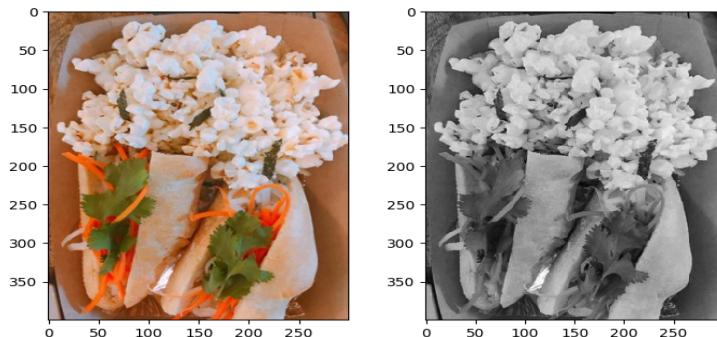
Distribución de las dimensiones



Scatterplot dimensiones por etiqueta

#### C.4) Análisis de las imágenes: Análisis de canales (RGB)

Todas las imágenes del dataset tienen 3 canales (RGB), por lo que se plantea la posible reducción a 1 único canal (escala de grises) para disminuir la previsiblemente elevada carga computacional y reducir a un tercio los recursos necesarios para entrenar los modelos.



Reducción de 3 a 1 canal

Se debe evaluar el impacto de la reducción de canales en el accuracy y tiempo de entrenamiento, ya que permitiría, potencialmente, entrenar el modelo que muestre mejor desempeño con el Dataset completo.

## D. Preparación de los datos

En esta fase, las imágenes van a requerir técnicas específicas (para detección de duplicados y data augmentation, por ejemplo), distintas de las usadas con datos tabulares.

### **D.1) Tratamiento de un subconjunto de imágenes:**

El dataset completo tiene 199.994 imágenes válidas y 106 corruptas por lo que no disponer de estas últimas no tendrá impacto alguno para el desarrollo de la solución.

Estimamos que con el fin de contrastar el rendimiento de distintos modelos, y tras varias pruebas, sería suficiente generar un dataset reducido compuesto por 10.000 imágenes procedentes del Dataset original, y siguiendo con la misma proporción de imágenes por etiqueta (desbalanceado).

Sobre el subconjunto de imágenes definido (10.000), y en base a los *findings* obtenidos en el proceso de EDA, se ha elaborado un proceso de limpieza descrito a continuación.

#### **Eliminación de imágenes duplicadas**

Para el tratamiento de duplicados se realizó un *script* en Python (ver GitHub) para su identificación y eliminación, utilizando la librería difPy [3].

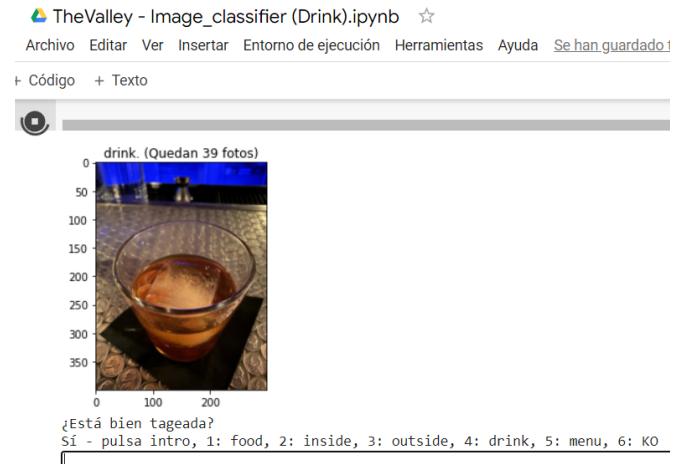
Dicho script, agrupa en carpetas las imágenes por *label* y hace un escaneo comparativo entre imágenes, identificando aquellas que están duplicadas, las cuales son eliminadas del subconjunto de imágenes.

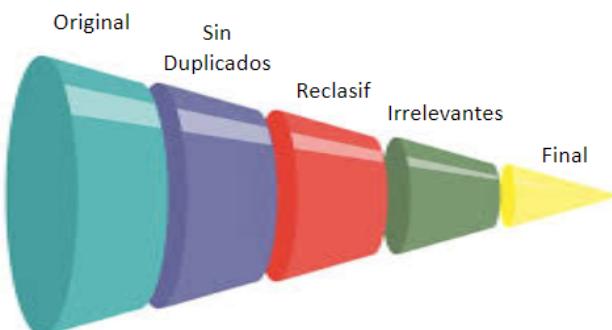
#### **Revisión de fotos mal clasificadas o no válidas.**

Así mismo, y con el fin de identificar imágenes mal clasificadas se creó un Motor de Etiquetado (ver GitHub), con el que se han revisado todas las imágenes del subconjunto de 10k.

#### **D.2) Impacto del proceso de limpieza**

Partimos de una submuestra del Dataset original compuesta por 10.000 imágenes. En una primera criba se eliminan 546 duplicadas, y pasan al Motor de etiquetado 9.454 fotos para su chequeo y reclasificación. Tras este proceso se reclasifican 107 y se identifican 1.142 como irrelevantes.





	Reclasificados							Irrelevantes	Final
	Original	Duplicados	Food	Inside	Outside	Drink	Menu		
Food	5.405	-404	+74			-11		-278	4.786 (-11%)
Inside	2.800	-40	-71		-7	-6	-9	-517	2.150 (-23%)
Outside	928	-8			+7			-316	611 (-34%)
Drink	783	-94	-3			+17		-26	677 (-14%)
Menu	84	0					+9	-5	88 (+5%)
<b>TOTAL</b>	<b>10.000</b>	<b>-546</b>						<b>-1.142</b>	<b>8.312 (-17%)</b>

### Conclusiones:

- Tras la limpieza el sub dataset se queda en 8.312 fotos, lo que representa una **reducción del 17%** de su tamaño.
- Hay pocas fotos mal clasificadas (1%), pero la categoría menú tiene más errores de clasificación (10%) que las demás (entre 2 y 4%).
- Gran parte de las imágenes etiquetadas bajo las clases de Inside y Outside resultan **confusas y potencialmente engañosas para los modelos** ya que contienen de forma relevante elementos pertenecientes a distintas clases (p.ej. Food y Drink a la vez).

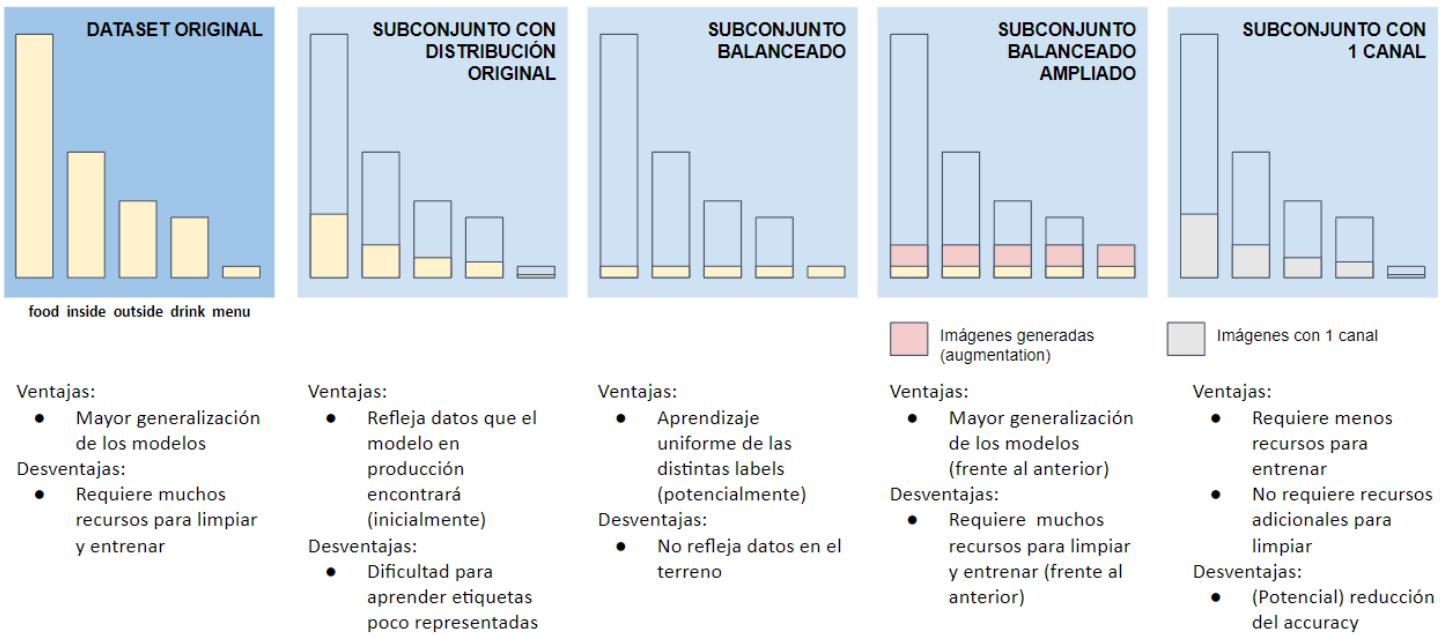
### D.3) Alternativas para generar los sub Datasets:

El rendimiento del entrenamiento de un modelo de *Deep Learning* depende inversamente del tamaño del dataset. Entrenar la capa final de clasificación (head) de un transformer ViT con el dataset completo, por ejemplo, tarda aproximadamente 4 días en un equipo con 6 cores i9.

Por otro lado, los recursos requeridos para las actividades manuales de limpieza del dataset también crecen proporcionalmente con el tamaño del dataset.

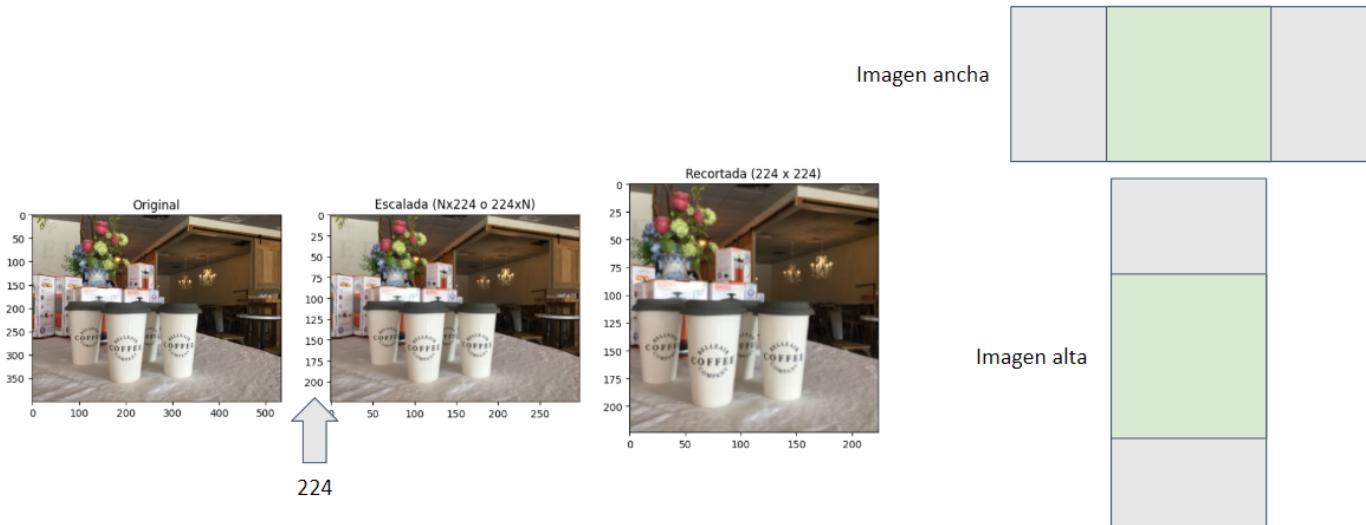
Por tanto se usará un subconjunto del dataset para evaluar el desempeño de los modelos, con un rendimiento razonable, y posteriormente se entrenará el mejor modelo con más datos, evaluando la mejora en *accuracy*.

A continuación se explica visualmente las alternativas consideradas para generar subconjuntos del dataset original.



#### D.4) Escalado y recorte:

De cara a maximizar el área de la foto original presente en la versión 224x2424, se escala la foto para que su menor dimensión sea 224 antes de recortar el cuadrado central (código disponible en GitHub).



## E. Modelación y Evaluación

Con el fin de establecer una **línea de base sobre la que evaluar el desempeño de distintos modelos** partimos de una **Red Neuronal Convolucional** de una prueba de concepto previa por el Bluetab, alimentada con el Dataset de 10K, que tiene la distribución de etiquetas del dataset original.

Una vez establecida la línea base (experimento 1), se hizo un conjunto de entrenamientos con la misma CNN y datasets alternativos, para evaluar el impacto de las características de los datasets en el aprendizaje del modelo (experimentos 2 y 3).

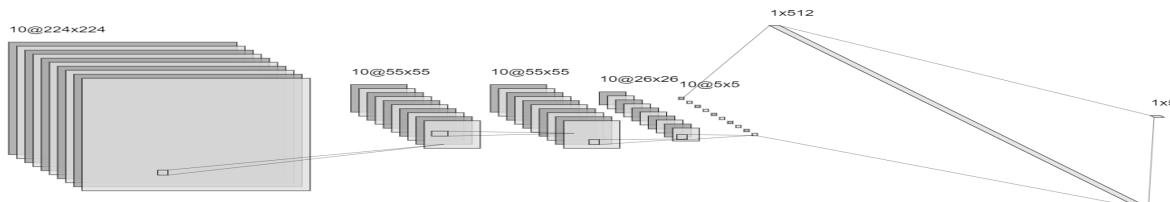
Una vez identificado el mejor dataset, se efectúan entrenamientos con distintas arquitecturas e hiper parámetros para optimizar el accuracy (experimento 4) y se entrena dos arquitecturas del estado del arte de clasificación de imágenes (experimentos 5 y 6).

## Experimento 1

**Objetivo:** establecer una línea base que sirva como referencia para entrenamientos posteriores.

### Arquitectura:

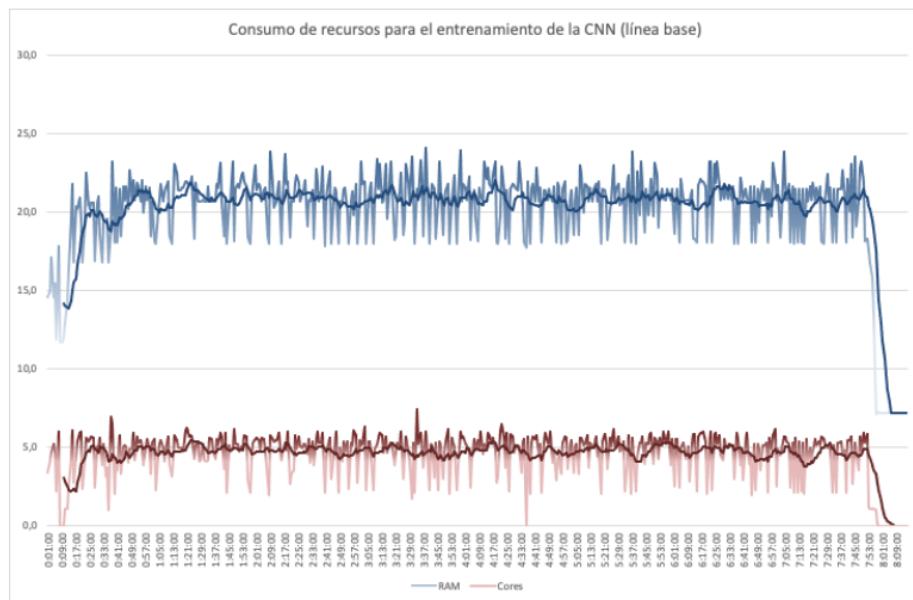
- CNN [4] de una prueba de concepto realizada por BlueTab, con 78 millones de parámetros (summary disponible en el repositorio del código en GitHub).



### Dataset:

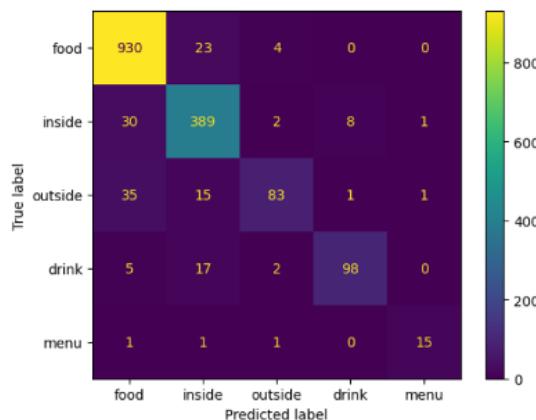
- Se usa el “dataset 10k”, un sub dataset compuesto por 8.312 fotos limpias, representando todas las categorías del dataset original en la misma proporción en que se encuentran en ese dataset.

Metrics (12)	
Name	Value
acc ↗	0.886
f1_m ↗	0.883
loss ↘	0.327
precision_m ↗	0.909
recall_m ↗	0.859
restored_epoch ↗	7
stopped_epoch ↗	17
val_acc ↗	0.858
val_f1_m ↗	0.852
val_loss ↘	0.436
val_precision_m ↗	0.881
val_recall_m ↗	0.825



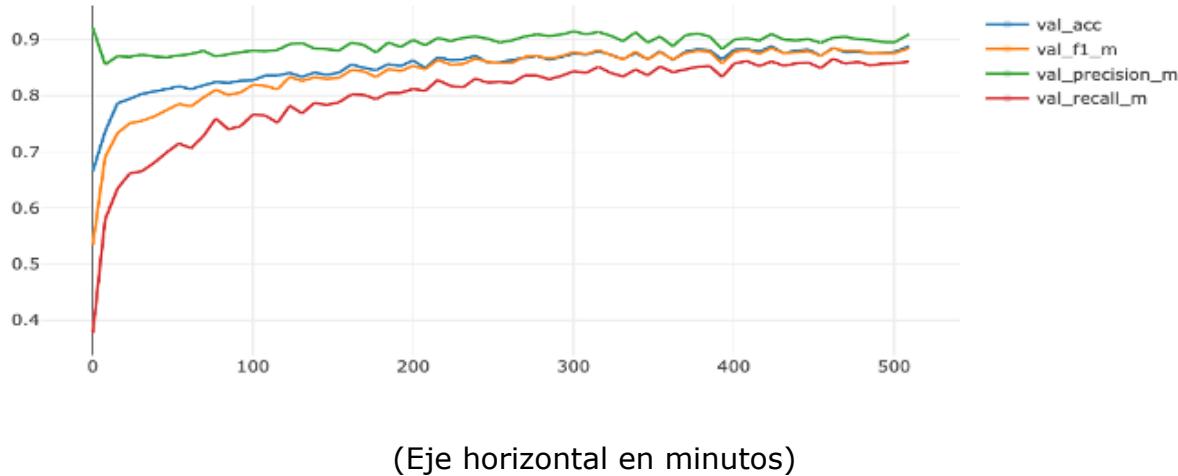
Monitorización del CPU y RAM a lo largo de un entrenamiento (run)

De acuerdo a la definición del problema de negocio la única métrica [5] relevante es el accuracy, pero se evaluó, además del accuracy global, el accuracy por etiqueta, el F1-score, la precisión y el recall.



Matriz de confusión

Las tres métricas convergen a medida que se entrena el modelo y el F1-score, una métrica de referencia para datasets desbalanceados converge en el valor del accuracy.



### Resultados del experimento 1:

- La línea base de este trabajo tiene un **accuracy del 85,8%** (validation accuracy).
- La evolución de las métricas a lo largo del entrenamiento permitió identificar el valor adecuado para la patience del **early stopping** (10 epochs).
- El entrenamiento de esta CNN con el dataset 10k cargado en memoria requiere más de 8 GB de RAM.
- Con la máquina de referencia<sup>1</sup>, cada epoch tarda 26 minutos en completarse.

## Experimento 2

**Objetivo:** validar si el desbalanceo de etiquetas afecta de modo negativo al accuracy del modelo base.

### Arquitectura:

- Se utiliza la misma arquitectura que en el experimento 1.

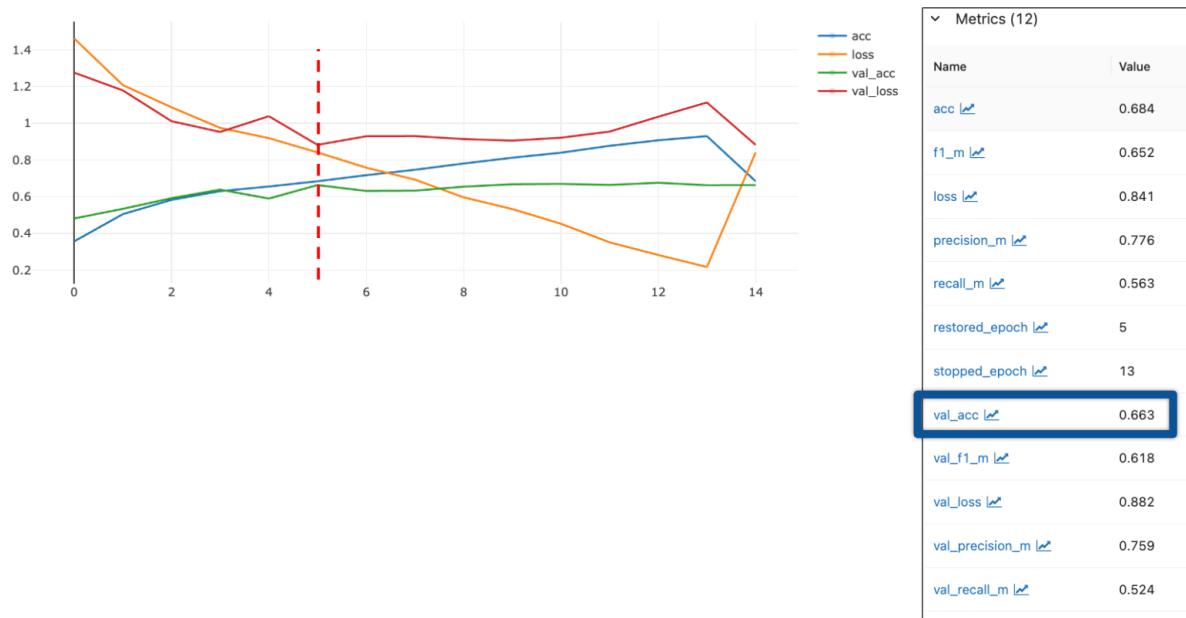
### Dataset:

- Dataset 10K **balanceado** en todas las etiquetas.

<sup>1</sup> MacBook Pro, 2,9 GHz Intel Core i9 de 6 núcleos, 32 GB 2400 MHz DDR4, macOS Monterey 2.6

## Resultados:

- El entrenamiento de la CNN base con el dataset balanceado disminuyó el **accuracy en el set de validación al 66,3%**, por lo que se:
  - Descartó este dataset para futuros entrenamientos;
  - Se descartó también el recurso a data augmentation para generar un dataset balanceado con más fotos (el gap de accuracy es demasiado para justificar la inversión necesaria para crear dicho dataset).



# Experimento 3

**Objetivo:** determinar si se puede reducir el número de canales (y por ende la memoria necesaria y el tiempo de entrenamiento) sin afectar al *accuracy*.

## Arquitectura:

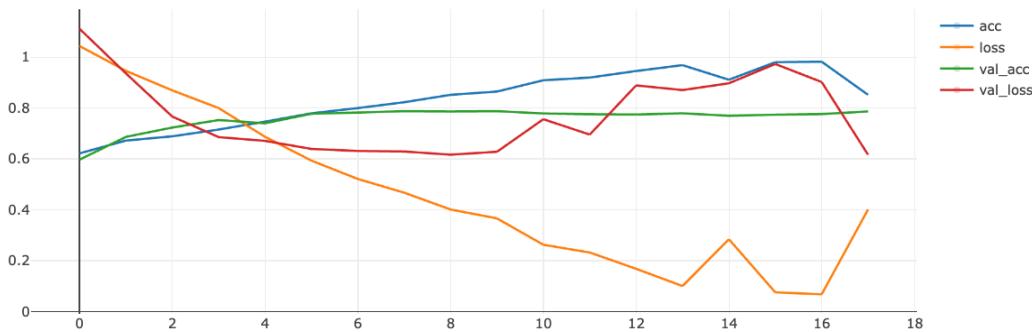
- Se utiliza la misma arquitectura que en el experimento 1.

## Dataset:

- Dataset 10K con fotos en gris.

## Resultados:

- El entrenamiento de la CNN base con el dataset con 1 canal disminuyó el **accuracy en el set de validación al 78,6%**, por lo que se descartó el recurso a datasets de 1 canal es entrenamientos posteriores.



Evolución de las métricas Loss y Accuracy a lo largo del entrenamiento

## Experimento 4

**Objetivo:** optimizar hiperparámetros y probar arquitecturas CNN alternativas.

Se completan 35 entrenamientos para ejecutar un grid search y probar otras CNN.

### Arquitectura:

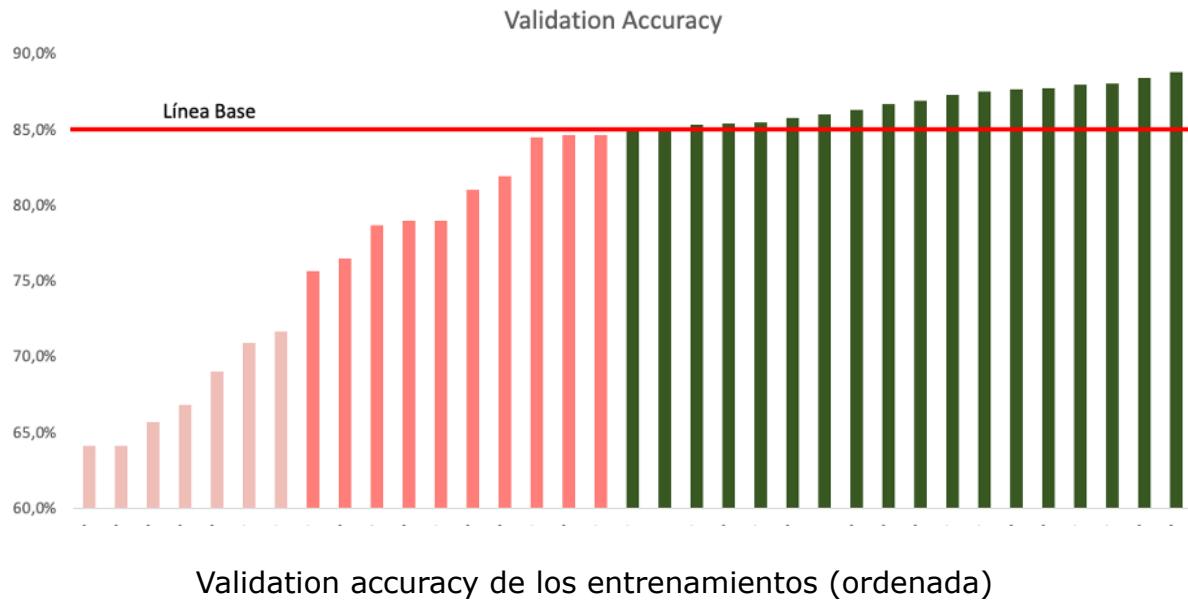
- Arquitectura de la línea base - grid search de:
  - Batch size: 32, 64 y 128
- Con el soporte del profesorado de The Valley, se definió otra arquitectura, con el objetivo de **reducir el número de parámetros** frente a la arquitectura del Experimento 1. Con esta arquitectura se realizó un grid search de:
  - Batch size: 32, 64 y 128
  - Kernels de las 3 capas max pooling: [5,5] y [3,3]
  - Filtros de las 4 capas convolucionales (manteniendo la progresión entre capas de la arquitectura base)
- Se entrenó también alguna red neuronal de **Kaggle** [6], pero sin obtener buenos resultados.

### Dataset:

- Se usa el dataset 10k.

## Resultados:

- El mejor de los 35 entrenamientos completados tiene una accuracy en el set de validación del **88,8%, un 3,0% superior a los resultados obtenidos en el experimento base.**



Model: "sequential_1"		
Layer (type)	Output Shape	Param #
conv2d_4 (Conv2D)	(None, 224, 224, 20)	8660
activation_6 (Activation)	(None, 224, 224, 20)	0
conv2d_5 (Conv2D)	(None, 222, 222, 30)	5430
activation_7 (Activation)	(None, 222, 222, 30)	0
max_pooling2d_2 (MaxPooling 2D)	(None, 44, 44, 30)	0
dropout_3 (Dropout)	(None, 44, 44, 30)	0
conv2d_6 (Conv2D)	(None, 44, 44, 40)	10840
activation_8 (Activation)	(None, 44, 44, 40)	0
conv2d_7 (Conv2D)	(None, 42, 42, 50)	18050
activation_9 (Activation)	(None, 42, 42, 50)	0
max_pooling2d_3 (MaxPooling 2D)	(None, 8, 8, 50)	0
dropout_4 (Dropout)	(None, 8, 8, 50)	0
flatten_1 (Flatten)	(None, 3200)	0
dense_2 (Dense)	(None, 512)	1638912
activation_10 (Activation)	(None, 512)	0
dropout_5 (Dropout)	(None, 512)	0
dense_3 (Dense)	(None, 5)	2565
activation_11 (Activation)	(None, 5)	0

Total params: 1,684,457  
 Trainable params: 1,684,457  
 Non-trainable params: 0

Name	Value
acc ↗	0.911
f1_m ↗	0.909
loss ↘	0.261
precision_m ↗	0.925
recall_m ↗	0.893
restored_epoch ↗	55
stopped_epoch ↗	65
val_acc ↗	0.888
val_f1_m ↗	0.885
val_loss ↘	0.37
val_precision_m ↗	0.91
val_recall_m ↗	0.861

CNN con mejor desempeño

# Experimento 5

Imagenet es una base de datos con 14 millones de imágenes clasificadas en más de 20.000 categorías. El estado del arte en clasificación de imágenes se encuentra en los resultados de la competición anual Imagenet, donde las mejores arquitecturas de redes neuronales compiten anualmente desde 2010 para alcanzar el mejor *accuracy*.

Los Transformers se introdujeron en 2017 y se han extendido ampliamente en el campo del procesamiento del lenguaje natural. Son modelos de Deep Learning que adoptan el mecanismo de autoatención, ponderando diferencialmente la importancia de cada parte de los datos de entrada.

En 2020 los Transformers fueron adaptados para tareas en visión artificial. La idea es básicamente descomponer las imágenes de entrada como una serie de segmentos que, una vez transformados en vectores, se ven como palabras en un transformador normal, para capturar las relaciones entre diferentes porciones de una imagen.

## Arquitectura:

- Para el experimento 5 se seleccionó la arquitectura vit-base-patch16-224 [7], que es la arquitectura de clasificación de imágenes más descargada en Hugging Face, la comunidad de AI.
- Se utilizó transfer learning, cargando los pesos pre entrenados y tuneando únicamente la capa de salida con las 5 categorías en pocos epochs.

## Dataset:

- Se usa el dataset 10k.

## Resultados:

- Este modelo alcanzó una accuracy del **96,8%, un 8,0% superior a los resultados obtenidos en el experimento 4.**

Name	Value
epoch ↗	3
eval_accuracy ↗	0.968
eval_loss ↗	0.123
eval_runtime ↗	293.7
eval_samples_per_second ↗	4.246
eval_steps_per_second ↗	0.531
learning_rate ↗	2.866e-6
loss ↗	0.116
total_flos ↗	1642486881965045800
train_loss ↗	0.207
train_runtime ↗	11905.9
train_samples_per_second ↗	1.78
train_steps_per_second ↗	0.223

Métricas del experimento 5

# Experimento 6

Una **red neuronal residual** ganó la competición en 2015 y desde entonces se ha convertido en la red neuronal más citada del siglo XXI.

## Arquitectura:

- Para el experimento 5 se probó la arquitectura **ResNet152V2** [8][9], una red neuronal residual muy profunda ,con cientos de capas y varios atajos entre ellas, que se utilizan para saltar sobre las capas, evitando el problema del *vanishing gradient* y de saturación de precisión (donde agregar más capas conduce a un mayor error de entrenamiento). Al entrenar este modelo, en una fase inicial, los atajos reducen la complejidad de la red, y posteriormente se quitan los atajos para entrenar la totalidad de la red.
- En este experimento también se utilizó *transfer learning*.

## Dataset:

- Se usa el dataset 10k.

## Resultados:

- La arquitectura ResNet alcanzó **97,2% de accuracy**, una mejora del 11,4% del accuracy sobre la línea base y 0,4% sobre la arquitectura ViT del experimento 5.

Model: "ResNet152V2"		
Layer (type)	Output Shape	Params
resnet152v2 (Functional)	(None, 7, 7, 2048)	583
global_average_pooling2d_2 (GlobalAveragePooling2D)	(None, 2048)	0
dense_4 (Dense)	(None, 250)	5122
dropout_2 (Dropout)	(None, 250)	0
dense_5 (Dense)	(None, 5)	1255
<hr/>		
<b>Total params: 58,845,153</b>		
<b>Trainable params: 513,505</b>		
<b>Non-trainable params: 58,331,648</b>		

Arquitectura ResNet

Name	Value
acc ↗	0.98
f1_m ↗	0.98
loss ↘	0.06
precision_m ↗	0.981
recall_m ↗	0.979
restored_epoch ↗	2
stopped_epoch ↗	32
val_acc ↗	0.972
val_f1_m ↗	0.971
val_loss ↘	0.091
val_precision_m ↗	0.973
val_recall_m ↗	0.97

métricas del experimento 6

# Comparativa de los experimentos

Experimento	Validation Accuracy
1 CNN / ORIG	85,8%
2 CNN / BAL	66,3%
3 CNN / GRIS	78,6%
4 CNN+ / ORIG	88,8%
5 ViT / ORIG	96,8%
6 ResNet / ORIG	<b>97,2%</b>

## F. DESPLIEGUE

El despliegue de la aplicación requiere de tres distintas etapas:

- 1) Proceso de scrapping de imágenes [BATCH]
- 2) Etiquetado de las imágenes [BATCH]
- 3) Web / App con las imágenes para usuario final [REAL TIME]

De cara a demostrar este proceso se creó una aplicación en Streamlit [10]. Durante la implantación de esta solución se optó por desarrollar una solución alternativa, para garantizar la disponibilidad de la demostración al terminar el trabajo.

Ambas aplicaciones, simulan el mismo proceso: una aplicación de **frontend**, donde los usuarios pueden navegar entre las diferentes galerías de imágenes, y otra de **backend**, usada por la empresa para subir, etiquetar las imágenes y añadirlas a la galería del frontend.

### F.1) Streamlit

Streamlit es un *framework* que permite desplegar aplicaciones de *Machine Learning* de manera ágil, que cuenta con la ventaja, entre otras, de ser un entorno 100% Python y no requerir experiencia de *frontend* en caso de que no se quiera customizar en exceso la aplicación.



## Backend. Etiquetado de imágenes.

Esta sección simula un proceso de *backend* en el cual se cagarían en *batch* las fotos que han sido scrappeadas de distintas fuentes (social media, portales de opinión, etc.) y subidas por los usuarios para su proceder a su etiquetado.

**Cargar y etiquetar foto:**

! Selecciona una imagen para clasificar:

Drag and drop file here
Browse files

 \_64R-FbpjfSFLevQyjiO6g.jpg 35.4KB

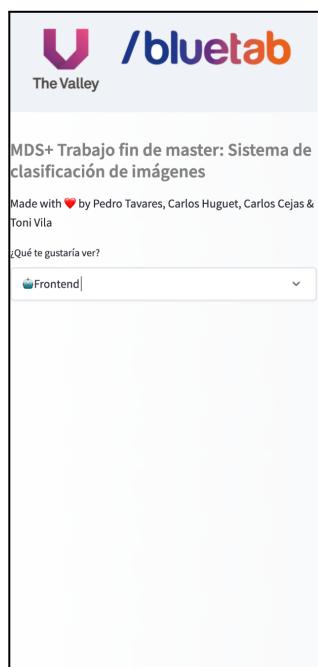
**Input Image**


**Predict**

La categoría es: food 🍔, con una probabilidad del 100.0 %

¿Está la foto bien clasificada?

## Backend



## Frontend: Explorador de imágenes

Esta sección simula la experiencia que podría tener el usuario final, que vería desde web / app la galería de imágenes y sobre la cual podría filtrar por las distintas categorías. Las imágenes se incluyen en la galería una vez han sido etiquetadas

A efectos de la demo y para limitar los tiempos de carga, únicamente se muestran 10 imágenes de cada categoría. Las imágenes se muestran de forma aleatoria en cada carga.

Escoge una categoría:

food

Mostrando 15 imágenes.






## Frontend

Algunas de las dificultades que nos hemos encontrado trabajando con **Streamlit**, y que finalmente se consiguieron solucionar para contar con una app fluida:

- **Tiempos de carga y optimización:** Con cada interacción con Streamlit, este hace ejecución del código de Python. Esto hacía que se cargase el modelo entrenado múltiples veces y la experiencia de usuario fuera muy mala. Finalmente se consiguió implementar **un caché que permitía la carga del modelo una única vez** al arranque de la aplicación.
- **Encoding fotos y diseño galería:** Por defecto, Streamlit no tiene ningún componente que permita crear una galería de imágenes. Por ello, se tuvo que recurrir a componentes desarrollados por la comunidad y que requerían serializar las fotos en JSON para su correcta visualización.
- **Poner la aplicación Live.** El despliegue de Streamlit se hace utilizando el código almacenado en Github. Éste tiene una limitación en el tamaño de los archivos, por lo que no nos permitía almacenar el modelo entrenado. Finalmente se recurrió a Hugging Face Spaces para almacenar el código y poner la aplicación online.

La aplicación está **disponible en Hugging Face Spaces** [11].

## F.2) Aplicación web en AWS

Esta aplicación web fué creada en Python y servida con Flask (para efectos de demostración), alojada en un servidor EC2 [12] en Amazon Web Services (AWS).

La estructura de esa aplicación es la misma que se utilizó con Streamlit: un frontend (sitio web) donde los clientes visitan la página del restaurante y ven las fotos clasificadas, y un backend donde simulamos la carga en el frontend de una foto obtenida por scrapping y clasificada con AI.



Back-office

**Restaurante ACME**

Fotos del restaurante en las redes sociales

Nuestra comida



Nuestros espacios



### Sitio web del restaurante (Frontend)

Tras la creación de la aplicación y su alojamiento en AWS se logró una buena experiencia de usuario en Streamlit. La infraestructura en AWS requerida para esta solución conlleva costes, por lo que la aplicación de referencia para demostrar la productivización de este trabajo es la app de Streamlit.

## G. PRÓXIMOS PASOS

En cuanto a los próximos pasos se plantea lo siguiente:

### Incremento del dataset de imágenes

Dado el *accuracy* obtenido (97% en el set de validación) no se estima necesario ampliar el dataset de imágenes y realizar un nuevo proceso de limpieza. El recorrido en el incremento en el *accuracy* es bajo, con una alta carga de trabajo en el proceso de revisión del etiquetado.

### Test del modelo con dataset de imágenes alternativo

Es importante destacar que todo el dataset de imágenes con el que se ha trabajado procede íntegramente de Yelp [13], plataforma muy popular en EEUU. Por lo tanto, se deberá revisar si el *accuracy* devuelve los mismos buenos resultados utilizando imágenes alternativas.

### **Monitorización del *drift* del modelo y posibilidad de reentrenamiento.**

Se deberá revisar que los valores de *accuracy* se sostienen en el tiempo y que el modelo no sufre *drift* [14]. Podría darse el caso que ante un cambio de concepto, por ejemplo, un cambio en las tendencias en gastronomía, el modelo presente un *accuracy* inferior.

En ese caso, el modelo tendrá que ser reentrenado y desplegado de forma controlada para no impactar negativamente en el *day-to-day* del negocio (Canary Release).

### **Nuevas funcionalidades**

Posibilidad de incorporar una funcionalidad que sea capaz de identificar el tipo de comida que aparece en las imágenes (Pasta, pizza, hamburguesa, asiático, etc.). Esto añadiría un plus de información a explotar tanto para el negocio como para el usuario final.

## **H. CONCLUSIONES**

Este proyecto ha supuesto una excelente oportunidad de aprendizaje, en donde hemos sido capaces de poner en práctica muchos de los conocimientos adquiridos durante el último año.

Como conclusiones del proyecto, destacar lo siguiente:

### **Trabajar con imágenes.**

Aunque trabajar con imágenes, a priori, resulta más sencillo que hacerlo con datos tabulares, identificar y corregir *outliers* (fotos mal etiquetadas, no relevantes, etc.) requiere de un proceso de revisión manual que supone un coste en tiempo elevado, y que resulta especialmente relevante cuando se trabaja con dataset grandes.

Además, trabajar con requiere de alta capacidad computacional para tratar las imágenes (reescalar, convertirlas en array, etc.) y realizar los entrenamientos.

### **Excelente resultado conseguido (97,2 % Accuracy en validación) .**

Tras múltiples experimentos y *runs* utilizando distintas arquitecturas y parámetros, el resultado final nos devuelve un modelo con elevado *accuracy* y que generaliza muy bien con fotos propias en distintas etiquetas, tamaños y situaciones.

### **Solape entre etiquetas.**

Tras realizar el proceso de revisión de imágenes, se detecta que existe ambigüedad en múltiples fotos, y que probablemente deberían tener varias

etiquetas (food / drink, food / inside, etc.), y que por lo tanto, nos encontramos ante un problema de clasificación multi-etiqueta.

### **Importancia del estado del arte.**

Dados los constantes avances en este campo y la existencia de nuevos algoritmos, es importante investigar cuál es el estado del arte antes de iniciar un proyecto de Data Science, lo que resultará en un ahorro en costes y en tiempos.

En el caso al que nos hemos enfrentado, se ha demostrado que utilizar algoritmos punteros como ViT o ResNet, se traduce en una importante diferencia en el *accuracy* del modelo y en la reducción de tiempos de entrenamiento, al ser algoritmos que utilizan *Transfer Learning*. Por lo tanto, en este caso, no sería conveniente utilizar una CNN con una arquitectura propia creada *ad-hoc*.

# I. REFERENCIAS

- [1] Clasificación multi-clase vs. Multi-etiqueta:  
<https://unipython.com/multiclass-and-multilabel-algorithms-algoritmos-mutliclase-multietiqueta/>
- [2] Repositorio de código del proyecto:  
[https://github.com/ptavaressilva/web\\_image\\_classifier](https://github.com/ptavaressilva/web_image_classifier)
- [3] Libreria difPy:  
<https://github.com/elisemercury/Duplicate-Image-Finder>
- [4] Redes neuronales convolucionales:  
[https://en.wikipedia.org/wiki/Convolutional\\_neural\\_network](https://en.wikipedia.org/wiki/Convolutional_neural_network)
- [5] Matriz de confusión y métricas para evaluación de clasificadores:  
<https://blog.paperspace.com/deep-learning-metrics-precision-recall-accuracy/>
- [6] Kaggle:  
<https://www.kaggle.com/>
- [7] Arquitectura vit-base-patch16-224:  
<https://medium.com/machine-intelligence-and-deep-learning-lab/vit-vision-transformer-cc56c8071a20>
- [8] Arquitectura ResNet152V2:  
[https://www.tensorflow.org/api\\_docs/python/tf/keras/applications/resnet/ResNet152](https://www.tensorflow.org/api_docs/python/tf/keras/applications/resnet/ResNet152)
- [9] Redes Neuronales Residuales:  
<https://towardsdatascience.com/an-overview-of-resnet-and-its-variants-5281e2f56035>
- [10] Streamlit:  
<https://docs.streamlit.io/>
- [11] Aplicación Streamlit para demostración del proceso de clasificación:  
[https://huggingface.co/spaces/tvila/streamlit\\_image\\_classifier](https://huggingface.co/spaces/tvila/streamlit_image_classifier)
- [12] Capacidad de computación en Amazon Web Services EC2:  
<https://aws.amazon.com/es/ec2/>
- [13] Dataset de imágenes de Yelp:  
<https://www.yelp.com/dataset>
- [14] Model drift:  
<https://www.dominodatalab.com/blog/data-drift-detection-for-image-classifiers>