UNIVERSITAT DE BARCELONA

Trabajo Final de Grado

GRADO DE INGENIERÍA INFORMÁTICA

Aplicatiu per la base de dades de Marques d'impressors per tal de poder detectar imatges similars de marques

---

Cristopher German Quispe Quispe

Director:	Eloi Puertas, Neus Verger
Realitzat a:	Departament de Matemàtiques i Informàtica

Barcelona, 11 de septiembre de 2020

Índice

1. Resumen

1.1. English

This project is based on the creation of an application that allows the end user to consult the database of images of printer's brands in charge of the Reserve Library of the University of Barcelona and to see images similar to a brand. For this, different methods are used to extract characteristics from the image, then techniques to improve the results obtained and finally, a web development that allows to carry out all the search functionalities. The project is divided into two phases, one of comparison and research and another part of software development for the creation of the application. Throughout the project we will see different technologies used for both the comparison part and for the development part of the final web. In the research and comparison we will see how to extract the descriptive vectors of different models and then compare them to obtain K similar images, which is currently called Transfer learning. Then we will apply a fine tuning to the selected model to improve the results. Once we have this model located we will go to the development of the final application in which consists of two parts: an API using the Django framework to facilitate the work and then the Frontend using the framework of Angular which will serve to show all the images and printers, always taking into account a satisfactory user experience. During the whole project we will experiment with different things that we will see in order to obtain the results we want at the end of the project.

1.2. Español

Este proyecto se basa en la creación de una aplicación que permita al usuario final poder consultar la base de datos de imágenes de Marcas de impresores a cargo de la Biblioteca de Reserva de la Universidad de Barcelona y poder ver imágenes similares a una marca. Para esto se utilizan diferentes métodos para la extracción de características de la imagen, luego técnicas para mejorar los resultados obtenidos y por ultimo, un desarrollo web que permita llevar a cabo todas las funcionalidades de búsqueda. El proyecto de divide en dos fases, una de comparación e investigación y otra parte de desarrollo de software para la creación del aplicativo. Durante todo el proyecto veremos diferentes tecnologías

usadas tanto para la parte de la comparación como para la parte de desarrollo de la web final. En la de investigación y comparación veremos como extraer los vectores descriptores de diferentes modelos para luego compararlos y poder obtener K imágenes similares, a lo que actualmente se denomina Transfer learning. Después aplicaremos un Fine tuning al modelo seleccionado para poder mejorar los resultados. Una vez tengamos dicho modelo localizado pasaremos al desarrollo del aplicativo final en cual constara de dos partes: una API usando el framework de Django para facilitarnos el trabajo y luego el Frontend usando el framework de Angular el cual nos servirá para mostrar todas las imágenes y los impresores, siempre teniendo en cuenta una experiencia de usuario satisfactoria. Durante todo el proyecto se experimentara con distintas cosas que iremos viendo para poder obtener los resultados que queremos al final de este mismo.

1.3. Catalán

Aquest projecte es basa en la creació d'una aplicació que permeti a l'usuari final poder consultar la base de dades d'imatges de Marques d'impressors a càrrec de la Biblioteca de Reserva de la Universitat de Barcelona i poder veure imatges similars a una marca. Per això s'utilitzen diferents mètodes per a l'extracció de característiques de la imatge, després tècniques per millorar els resultats obtinguts i per últim, un desenvolupament web que permeti dur a terme totes les funcionalitats de cerca. El projecte de divideix en dues fases, una de comparació i investigació i una altra part de desenvolupament de programari per a la creació de l'aplicatiu. Durant tot el projecte veurem diferents tecnologies usades tant per la part de la comparació com per la part de desenvolupament del web final. En la d'investigació i comparació veurem com extreure els vectors descriptors de diferents models per després comparar-los i poder obtenir K imatges similars, al que actualment s'anomena Transfer learning. Després farem un Fine tuning a el model seleccionat per poder millorar els resultats. Un cop tinguem aquest model localitzat passarem a el desenvolupament de l'aplicatiu final en qual constarà de dues parts: una API usant el framework de Django per facilitar-nos el treball i després el Frontend usant el framework de Angular el qual ens servirà per mostrar totes les imatges i els impressors, sempre tenint en compte una experiència d'usuari satisfactòria. Durant tot el projecte es experimentés amb diferents coses que anirem veient per poder obtenir els resultats que volem a el final d'aquest

mateix.

2. Introducción

2.1. El proyecto

El proyecto viene definido por la Biblioteca Biblioteca de Reserva de la Universidad de Barcelona, en este caso para el análisis de las imágenes de marcas de impresores y como encontrar una solución al problema de la obtención de imágenes similares dada otra imagen para el dataset dado. En este caso la limitación de tener acceso directo a la base de datos fue un problema que se pudo solucionar como se comenta más adelante. Las imágenes se centran en la marca del impresor y contienen detalles de dicha marca, algunas mas visibles que otras y con algún objeto poco relevante en ellas como se puede ver a continuación en el ejemplo de las imágenes.

Figura 1: Imágenes

El proyecto consiste en dos partes, la primera que sera obtener las características de las imágenes para luego mediante el algoritmo de Nearest Neighbors obtener las K imágenes más similares a la imagen objetivo, para esta primera parte usaremos modelos pre-entrenados para la extracción de dichas características, ya que son modelos entrenados con millones de imágenes con mas de 1000 clases diferentes, luego trataremos de mejorar los resultados mediante técnicas de fine-tuning,

volviendo a entrenar la red de estas para poder obtener mejores resultados en la extracción de características.

Para la segunda parte, mediante tecnologías web haremos una interfaz con la cual el usuario podrá interactuar. En este caso Angular para la parte de Frontend que sera con la cual el usuario final interactuará y por otra Django en la parte del Backend que sera la que contendrá toda la información referente a las imágenes a analizar y devolver toda la información que se solicite. La idea seria tener una API con la cual al solicitar imágenes similares, dada una imagen seleccionada por el usuario, devuelva las más relevantes, esta información sera representada en formato JSON la cual consumirá el frontend para poder mostrar el resto de imágenes de manera que el usuario pueda ver e interactuar con dichas imágenes.

2.2. Motivación

La motivación principal del proyecto era el experimentar con tecnologías que se han visto en la carrera pero no se ha llevado a cabo un estudio profundo en dichas materias, en este caso los conocimientos adquiridos en Visión artificial, Desarrollo web, Factores humanos e Ingeniería de Software, en este proyecto se ha intentado combinar el conocimiento de todas estas y así poder aprender un poco más ambas materias y nuevas tecnologías.

Por otra parte, se ha tenido en cuenta los requerimientos del cliente para la web final y tener una valoración de las imágenes mostradas en la web a la hora de obtener imágenes similares, y así poder mostrar los avances que se iban realizando en el proyecto. Siguiendo una metodología agile se ha podido lograr tener una demo en condiciones para todo lo mostrado en este proyecto y para que el cliente pueda ver que dicha solución es viable para mostrar imágenes similares.

El interés es la de probar la técnica de transfer learning con imágenes diferentes para las cuales fueron entrenadas y ver que tal se comportan fue una de las motivaciones de este proyecto, y optimizar esto modificando una red pre-entrenada para poder obtener mejores resultado, también para el aprendizaje de como funcionan estas por dentro y adquirir nuevos conocimientos.

Como se ha dicho anteriormente en este trabajo se ha intentado plasmar diferentes ideas de diferentes asignaturas tanto como Programación Web, Desarrollo de Software, Visión Artificial y Factores Humanos. Todas estas combinadas aportaban un conocimiento extra a este proyecto, tanto para la parte de desarrollo de software orientado a las tecnologías web, como el conocimiento de redes convencionales para la extracción y clasificación de imágenes que se han visto en Visión Artificial. Por otra parte, el tener una aplicación web nos ayuda a centrarnos también en el apartado de usabilidad para el usuario final teniendo en cuenta lo visto en Factores humanos en el tema de interfaces de usuario.

3. Objetivos

Se han planteado una serie de objetivos antes de empezar con el proyecto, los cuales se intentaron lograr al final de este mismo. En este caso se dividieron los objetivos en dos partes, una para la investigación de los modelos y experimentos con estos, otro para el desarrollo de la aplicación web.

3.1. Objetivos de modelos

Para el objetivo de la investigación de los modelos, se seleccionaron y se pusieron a prueba distintos modelos que serán nombraron mas adelante a lo largo del proyecto. Para estos mismo se han marcado los siguiente objetivos que tenemos que cumplir para la continuidad del proyecto.

- Análisis de modelos.
- Extraer un vector característico para las imágenes.
- Obtener las imágenes más similares.
- Comparación de modelos.
- Selección del modelo final.
- Mejorar dicho modelo (fine-tuning).

3.2. Objetivos de la aplicación

Para el apartado de desarrollo se han marcado los objetivos teniendo en cuenta las dos aplicaciones, tanto para el backend como para el frontend, cada uno tendrá unos despliegues diferentes y contaran con un desarrollo a parte teniendo en cuenta diferentes objetivos.

3.2.1. Backend

- Modelado de la base de datos.
- Creación del proyecto en Django (Backend).
- Creación de la API.
- Búsqueda de imágenes similares.
- Búsqueda de impresores.

3.2.2. Frontend

- Creación del proyecto de Angular.
- Creación del servicio para comunicación con la API.
- Creación de la interfaz de usuario.
- Diseño de la interfaz.
- Usabilidad de la interfaz.

4. Planificación

4.1. Previa

Esta planificación se realizo al inicio del TFG en este caso la idea que se tuvo en mente era la de utilizar una red pre-entrenada para poder obtener las características de una imagen y luego obtener imágenes similares de la imagen de ejemplo, esta era la idea principal y a partir de aquí seguir mejorando el resultado

obtenido. Finalmente crear una aplicación web sencilla en la cual consultar las imágenes de impresores y mostrar las imágenes similares de una imagen seleccionada por el usuario.

Para esta primera parte, una vez se plateo la idea principal, la recomendación del tutor del TFG fue la de consultar diferente documentación acerca de las redes convolucionales para familiarizarnos con los términos y poder indagar sobre el problema planteado, poder buscar alguna otra solución al problema y tener una vista más general, también buscar información sobre Content-Image Retrieval que es el sistema objetivo logra al final de este proyecto. En este caso lo que se busco fue un sistema que realice una consulta de imágenes dada una imagen de ejemplo por el usuario, buscar imágenes similares a dicha imagen y recuperar las imágenes basándose en el contenido de la imagen de ejemplo.

Todo el tiempo a esta primera fase se dedico a la investigación de dichas tecnologías y como se podrían implementar para este objetivo en concreto, combinando cada una de ellas para así poder obtener una aplicación viable y con buenos resultado dentro de las limitaciones que tenemos con este dataset y con la validación del usuario final para la muestra de imágenes similares.

4.2. Inicial

Teniendo las ideas claras, en esta fase la idea principal fue la de investigación sobre sistemas de consultas de imágenes mediante ejemplo y también la investigación del Transfer Learning que es el método se usa en este proyecto y sirve para la extracción de características de las imágenes que fueron tratadas por las redes, para este primera parte el objetivo fue saber que se podría hacer con dichas redes, en este caso se opto por seleccionar tres redes:

- VGG
- ResNet
- MobileNet

Estos modelo fueron usados para la extracción las características de una una imagen y la obtención de un vector característico de dicha imagen, para poste-

riormente usar el algoritmo del vecino más cercano (KNN) mediante la métrica de distancia euclídea para encontrar imágenes que para este momento solo serán simples vectores, una vez tengamos estos vectores de las imágenes se procedió a buscar los que mas se parezcan a la imagen de ejemplo.

Una vez que se realizo la extracción de todos los vectores para cada imagen, y en cada una de las redes, se paso a comprobar que modelo pre-entrenado da mejores resultado para el dataset dado y comprobar que tal se comportaban como extractor de características de la imagen, para ello se comprobó para una imagen si alguno de los tag corresponden a la imagen de consulta, si los resultados obtenidos contienen algún tag de la imagen de consulta se trato de valido este resultado y diremos que tenemos una imagen parecida, esto seria del todo cierto si cada una de las imágenes de dicho tag se parecen mucho. Esta es la idea que se siguió para el planteamiento de imágenes similares como se vera mas adelante.

4.3. Media

Para esta parte la planificación que se tuvo en mente era la terminar de comparar y seleccionar un modelo el cual se usaría para la aplicación final de la web en la cual se podría consultar los datos y empezar con el desarrollo de la aplicación web. Antes de esto se buscaría alguna forma de mejorar los resultados de dicho modelo con el dataset que tenemos en este momento, que es bastante limitado en cuestión de imágenes y etiquetas. En este punto se tuvo en mente la aplicación de un fine-tuning al modelo seleccionado con los mejores resultados para poder aumentar la respuesta devuelta por el extractor característico del modelo para poder identificar mucho mejor imágenes que se parecen entre si para este dataset en concreto. Estos modelos se han entrenado mediante ImageNet en cual tiene millones de imágenes con mas de 1000 categorías, en este caso el extractor característico de dichos modelos es mucho mejor que cualquier modelo que vayamos a implementar para un dataset tan limitado como el nuestro. Siguiendo esta idea, una vez realizado el fine-tuning comprobamos que tal se comporta este nuevo modelo en cuestión del modelo original y una vez obtenida una mejora en cuestión de obtención de imágenes similares, nos quedamos con dicho modelos para aplicarlo a la extracción de características de las imágenes del dataset.

Para el desarrollo web se tuvo que dividir el trabajo en dos partes una para la parte de Backend con Django, con el cual ya se estaba familiarizado en la carrera, y así poder crear una API sencilla con las imágenes la cual devolvería por cada consulta N imágenes similares a la imagen consultada de ejemplo. También se tuvo en cuenta la creación de una basa de datos para los impresores a los cuales les corresponderá una o varias imágenes, dichas imágenes tienen uno o mas tags, todo a su vez con en una API creada para poder consultar a los impresores y obtener las imágenes de cada uno.

Para la parte del aplicativo Frontend el planteamiento fue la creación de una aplicación con Angular que es el framework que se uso en el proyecto de Ingeniería de Software, y la cual por ende era de mi interés aprenderlo, esta fue una de las razones por la cual me decante por este framework de javascript. Una vez con el modelo definido y con la API creada con Django, lo único que se tuvo que hacer en este apartado era la creación de una interfaz para el usuario final, por la cual el usuario consulta la base de datos de impresores y puede ver las imágenes parecidas a alguna que se haya seleccionado del impresor que se prefiera. Esta fase de desarrollo es la final después de todos los pasos mencionados anteriormente, el desarrollo de todo el TFG fue bastante largo y ha cubierto una gran cantidad de información nueva la cual estaba interesado en aprender durante todo el transcurso de este proyecto.

4.4. Diagrama de Gantt

Aquí podemos la distribución en mente de todo el desarrollo antes de su presentación, es lo me plantee en un inicio seguir, pero surgieron problemas que se comentan mas adelante en el apartado de Problemas 11.1.

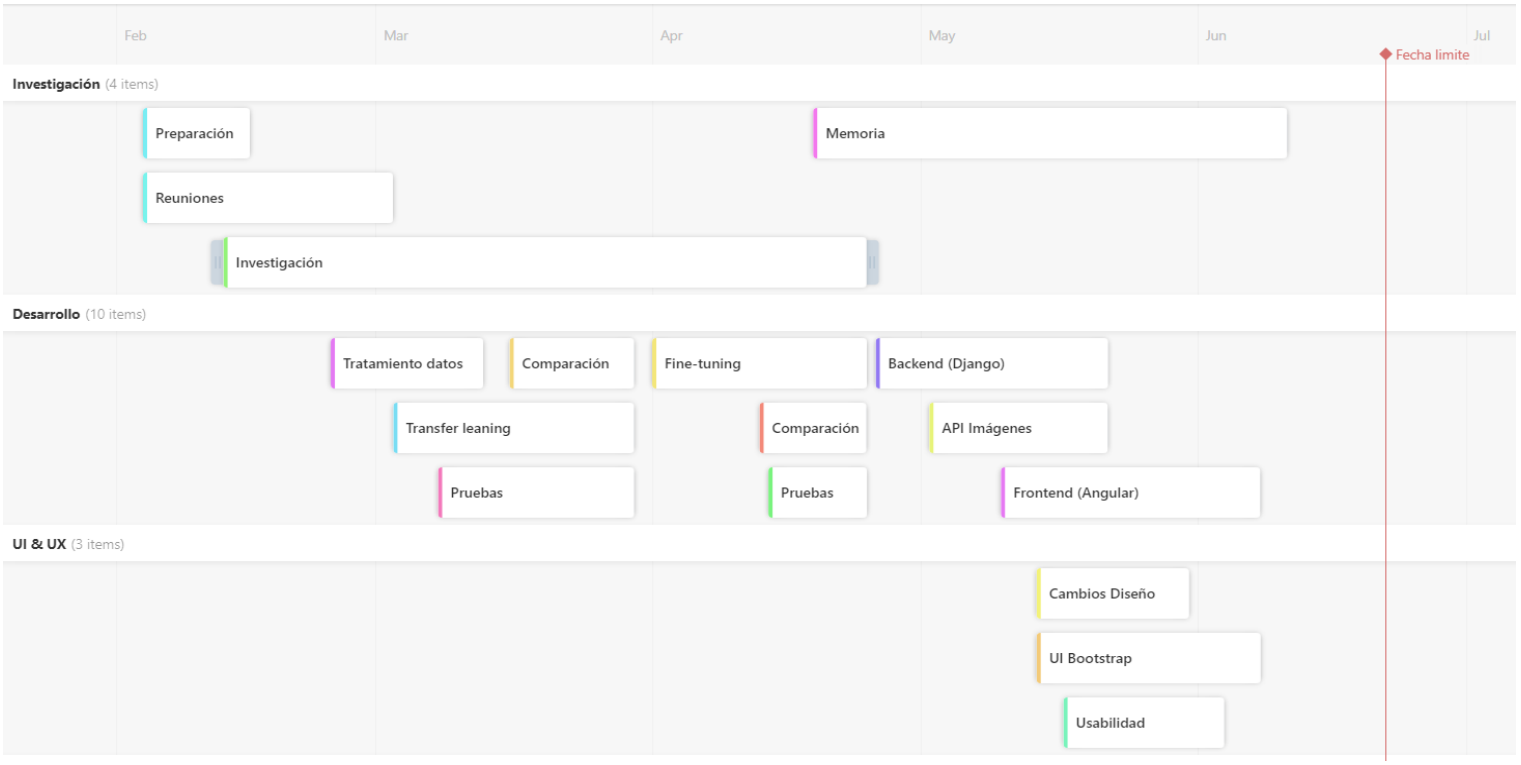


Figura 2: Planificación inicial

5. Conceptos

En esta apartado vamos a analizar los modelos seleccionados y por qué han sido seleccionados para un análisis posterior y para poder compararlos más adelante. También vamos a explicar conceptos básicos de las redes convolucionales para poder entender el funcionamiento de estas y proceder con las demás técnicas empleadas en este TFG.

5.1. Redes convolucionales

Muchos de los enfoques que vemos en Visión por ordenador explotan la propiedad de que píxeles cercanos están mas correlacionados entre ellos. Con esto podemos realizar la extracción de características locales en solo pequeñas subregiones de la imagen. Toda esta información de las características las podemos combinar

en múltiples etapas de procesamiento con el fin de detectar de características de alto nivel para poder obtener una información completa de la imagen y poder obtener el contexto de la misma. Además, las características que obtenemos en una región, probablemente sean útiles en otras regiones de la imagen.

Las redes convoluciones son similares a las redes neuronales, la diferencia es que las redes convolucionales esperan explícitamente que el conjuntos de entrada sean imágenes de X dimensión. Esto nos permite extraer ciertas propiedades de la imagen dentro de toda la red pasando por cada una de las capas convolucioanles. Están inspiradas en redes neuronales ya que incorporan operación no lineales como ReLU y otras operaciones de visión por ordenador como la manipulación de las imágenes mediantes filtros convolucionales para la extracción de características.

5.1.1. Convolución

La convolución es una técnica que modifica la imagen usando los valores de sus pixeles mediante la aplicación de un filtro el cual producirá una imagen resultante la cual puede contener información relevante para la red. Este calculo es bastante sencillo, el filtro que pasamos a la imagen determina el valor del píxel central que tendrá la nueva imagen en la posición a la que se le haya aplicado a la imagen y sumando el valor de todos sus vecinos multiplicado el valor del filtro.

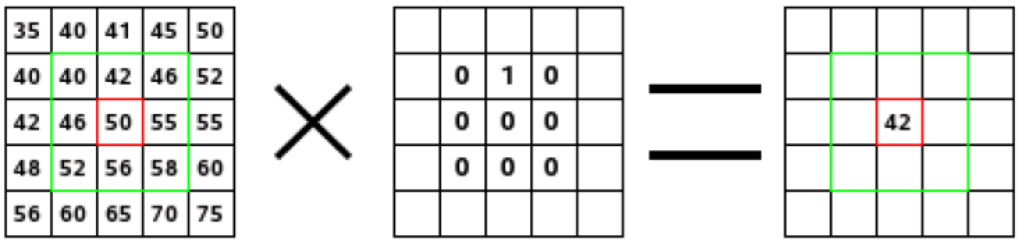


Figura 3: Convolución

Normalmente estos filtros se agrupan en una única capa dentro de la red convolucional para el procesado de extracción de características, para completar nuestra convolución, repetimos este proceso, alineando la función (filtro) con cada posible

parche de la imagen. Podemos tomar la respuesta de cada convolución y hacer una nueva matriz bidimensional a partir de ella, según el lugar de la imagen que se encuentre cada parche. El siguiente paso es repetir el proceso de convolución en su totalidad para cada una de las otras características. El resultado es un conjunto de imágenes filtradas, una para cada uno de nuestros filtros. Es conveniente pensar en toda esta colección de operaciones de convolución como un solo paso de procesamiento dentro de la capa de convolución en la red.

5.1.2. Pooling

Las capas de submuestreo o pooling ayudan a reducir la cantidad de parámetros a entrenar y por ende al numero de cálculos, reduce la representación espacial de la imagen, así como también permite una mejor generalización a la hora de poder predecir. Esta operación es bastante sencilla la cual equivale a quedarnos con el máximo o con un promedio de un área de una matriz resultante.

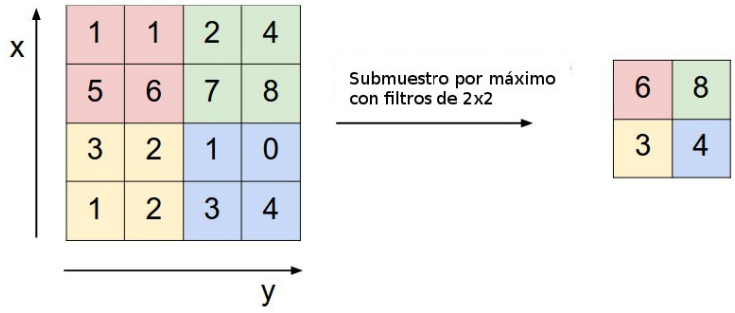


Figura 4: Pooling

Una capa de pooling es sólo la operación de realizar un pooling sobre una imagen o una colección de imágenes. La salida tendrá el mismo número de imágenes, pero cada una tendrá menos píxeles.

5.1.3. Unidades lineales rectificadas (ReLU)

Un pequeño pero importante función en este proceso es la Unidad Lineal Rectificada o ReLU. Es bastante sencilla de entender: cuando se produce un número negativo, se cambia por un 0. Es decir toma el máximo entre el valor y cero.

$$f(x) = max(0, x)$$

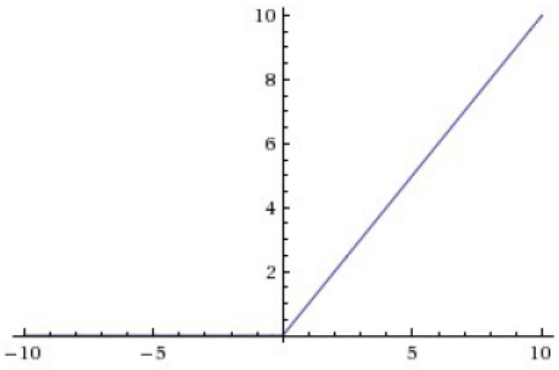


Figura 5: ReLU

La salida de una capa de ReLU es del mismo tamaño que el de la entrada, sólo que con todos los valores negativos eliminados.

5.1.4. Capas completamente conectadas

Capas completamente conectadas toman las imágenes filtradas de alto nivel y las traducen en votos para posteriormente seleccionar una como resultado. Cuando una nueva imagen se presenta a la CNN, se filtra a través de las capas inferiores hasta llegar a la capa totalmente conectada al final. Entonces se lleva a cabo una elección. La respuesta con más votos gana y se declara la categoría de la entrada. En lugar de tratar las entradas como un conjunto bidimensional, se tratan como una lista única y todas se tratan de forma idéntica. Cada valor tiene su propio voto sobre si la imagen actual.

5.1.5. Backpropagation

¿De dónde vienen los valores de los filtros? y ¿Cómo encontramos los pesos en nuestras capas completamente conectadas? Si todo esto tuviera que ser elegido a mano, las redes convolucionales no serian tan populares. No nos vamos a adentrar lo que ocurre dentro de la Backpropagation, pero se explicara un poco por

encima su funcionamiento de manera mas abstracta.

Cada imagen que la red convolucional se procesa y resulta en una votación. La cantidad de error en la votación nos dice cuán buenos son nuestras características extraídas y pesos. Las características y los pesos pueden ser ajustados para que el error sea menor. Cada valor se ajusta un poco más alto y un poco más bajo, y el nuevo error se calcula cada vez. Cualquier ajuste que haga que el error sea menor se mantiene. Después de hacer esto para cada característica en cada capa convolucional y cada peso en cada capa completamente conectada, los nuevos pesos dan una respuesta que funciona ligeramente mejor para esa imagen. Esto se repite con cada imagen siguiente en el conjunto de imágenes etiquetadas. Las interferencias que ocurren en una sola imagen se olvidan rápidamente, pero los patrones que ocurren en muchas imágenes se mantienen en las características y los pesos de las conexiones. Si tenemos suficientes imágenes etiquetadas, estos valores se estabilizan en un conjunto que funciona bastante bien en una amplia variedad de casos.

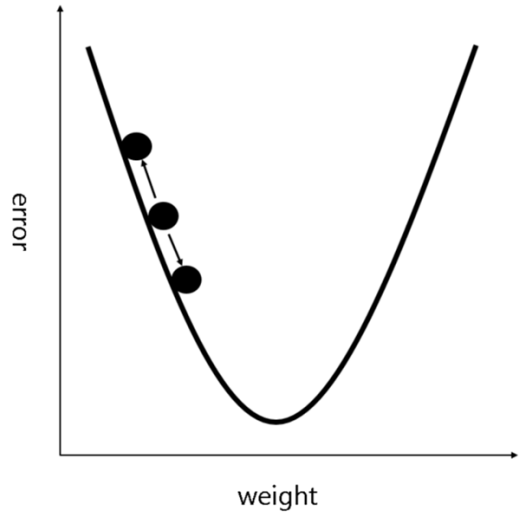


Figura 6: Backpropagation

5.1.6. Red completa

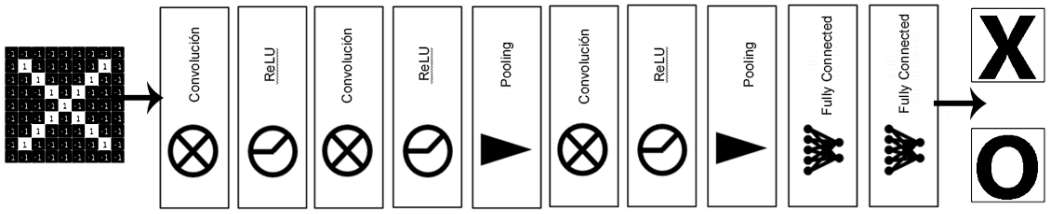


Figura 7: ReLU

5.1.7. Hiperparámetros

Desafortunadamente, no todos los aspectos de las redes convolucionales pueden aprenderse de una manera tan directa. Hay muchas decisiones que tomar y varias cosas a tener en cuenta a la hora de diseñar una red convolucional.

- Para cada capa convolucional, ¿Cuántos filtros? ¿Cuántos píxeles en cada filtro?
- Para cada capa de pooling, ¿Qué tamaño de ventana? ¿Qué salto?
- Por cada capa totalmente conectada, ¿Cuántas neuronas ocultas?

Además de estos también hay decisiones de arquitectura que hay que tomar ¿Cuántas de cada capa hay que incluir? ¿En qué orden? Algunas redes neuronales profundas pueden tener más de mil capas, lo que abre muchas posibilidades. Con tantas combinaciones y permutaciones, sólo se ha probado una pequeña fracción de las configuraciones posibles de una red convolucional. Los diseños de redes convolucionales tienden a ser impulsados por el conocimiento acumulado de la comunidad. Pero hay muchos otros ajustes que han sido probados y encontrados efectivos, como nuevos tipos de capas y formas más complejas de conectar las capas entre sí, en este caso no entraremos en detalles de estas.

5.2. Modelos

En esta apartado vamos a hablar sobre los modelos que hemos elegido para la comparación. Existen modelos de redes convolucionales que son populares por haber ganado competencias orientadas a la clasificación de imágenes, los cuales están disponibles usando diferentes librerías. Estos modelos suelen ser empleados como base pre-entrenada para múltiples áreas.

5.2.1. ImageNet

El proyecto ImageNet es una gran base de datos visual diseñada para su uso en la investigación de software de reconocimiento de objetos visuales. El proyecto ha anotado a mano más de 14 millones de imágenes y contiene más de 20.000 categorías. Para este proyecto se han usado modelos que se han entrenado sobre esta base de datos de imágenes.

5.2.2. VGG16

Este modelo quedo en el segundo puesto del 2014 de la competición de ImageNet ILSVRC (ImageNet Large Scale Visual Recognition Challenge). Es una de las primeras redes profundas mas conocidas. Su contribución principal fue demostrar que la profundidad de la red es un componente critico para un desempeño óptimo y la obtención de mejores resultados. Contienen relativamente pocas capas convolucionales: 13 capas convolucionales y 3 densas, de ahí que en sus nombre incluya el 16. Una desventaja de la VGG16 es que es muy cara para evaluar y consume mucha memoria y requiere de muchos parámetros 140 millones aproximadamente.

A continuación podemos ver la arquitectura de este modelo.

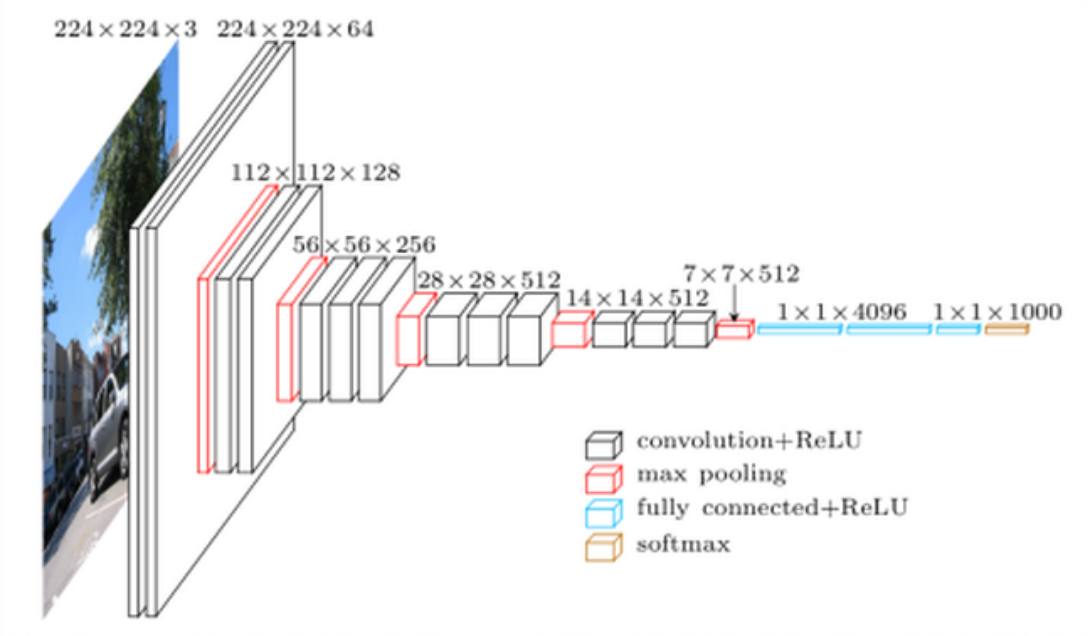


Figura 8: VGG16

5.2.3. ResNet50

ResNet fue introducida por Microsoft, ganando la competición ILSVRC (ImageNet Large Scale Visual Recognition Challenge) en el año 2015. Es una red que intenta solucionar el problema en redes muy profunda con múltiples capas. En este caso llamados bloques residuales, el modelo que vamos a usar tiene 50 bloques residuales.

A medida que diseñamos redes cada vez más profundas se hace imperativo entender cómo la adición de capas puede aumentar la complejidad y la expresividad de la red. Aún más importante es la capacidad de diseñar redes en las que la adición de capas hace que las redes sean estrictamente más expresivas y no sólo diferentes. Para resolver el problema del gradiente de desaparición en el que se pierde información en capas más profundas, esta arquitectura introdujo el concepto llamado Red Residual. En esta red utilizamos una técnica llamada conexiones de salto. La conexión de salto salta el entrenamiento de unas pocas capas y se conecta directamente a la salida.

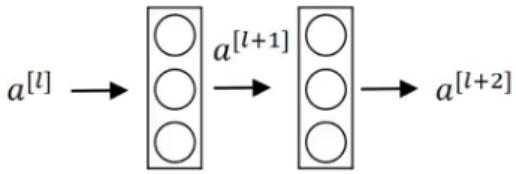


Figura 9: Residual block

$$
\begin{aligned}
z^{[l+1]} &= W^{[l+1]}a^{[l]} + b^{[l+1]} \\
a^{[l+1]} &= g(z^{[l+1]}) \\
z^{[l+2]} &= W^{[l+2]}a^{[l+1]} + b^{[l+2]} \\
a^{[l+2]} &= g(z^{[l+2]})
\end{aligned}
$$

Después de aplicar el short cut, lo que hacemos es pasar el valor de $a^{[l]}$ directamente al final para sumar su valor, así mantenemos este dato residual.

$$
\begin{aligned}
z^{[l+1]} &= W^{[l+1]}a^{[l]} + b^{[l+1]} \\
a^{[l+1]} &= g(z^{[l+1]}) \\
z^{[l+2]} &= W^{[l+2]}a^{[l+1]} + b^{[l+2]} \\
a^{[l+2]} &= g(z^{[l+2]} + a^{[l]})
\end{aligned}
$$

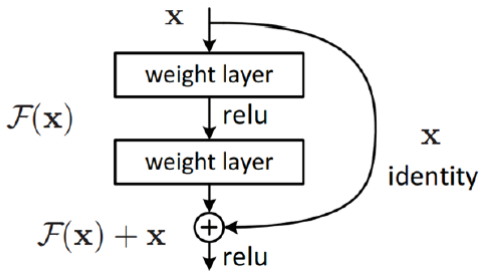


Figura 10: Residual block

5.2.4. MobileNet

MobileNet es un modelo de arquitectura de una red convolucional para la clasificación de imágenes y la visión por móvil. Existen otros modelos, pero lo que hace especial a MobileNet es que tiene menos potencia de cálculo para ejecutar o aplicar el aprendizaje por transferencia. Esto lo hace perfecto para los dispositivos móviles, los sistemas incorporados y las computadoras sin GPU o con una baja eficiencia de cálculo. También es más adecuado para los navegadores web, ya que los navegadores tienen limitaciones en cuanto a la computación, el procesamiento gráfico y el almacenamiento.

La capa central de MobileNet son los filtros separables en profundidad, denominados Convolución Separable en Profundidad (Depthwise Separable Convolution). La estructura de la red es otro factor para aumentar el rendimiento. Por último, el ancho y la resolución pueden ajustarse para compensar la latencia y la precisión.

Las convoluciones separables son mas eficientes a la hora de calcularse que la convoluciones normales ahorrándonos un numero significativo de multiplicaciones. Las convolución separable de realiza en dos partes una Depthwise convolution y una Pointwise convolution. Un ejemplo seria el siguiente:

Si tenemos una imagen de entrada de 32x32x3 y queremos una imagen de salida de 28x28x64 aplicando un filtro de 5x5x3 sin padding y con stride de 1, en una convolución normal el calculo seria el siguiente:

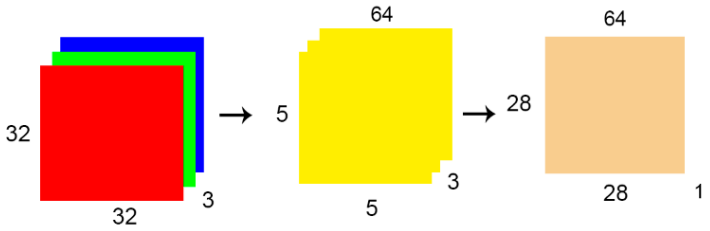


Figura 11: Convolution

$$64 \times 5 \times 5 \times 3 \times 28 \times 28 = 3763200$$

Obtenemos un total de 3 763 200 multiplicaciones

Si aplicamos lo mismo usando una Depthwise convolution y una Pointwise convolution obtendremos el siguiente número de multiplicaciones:

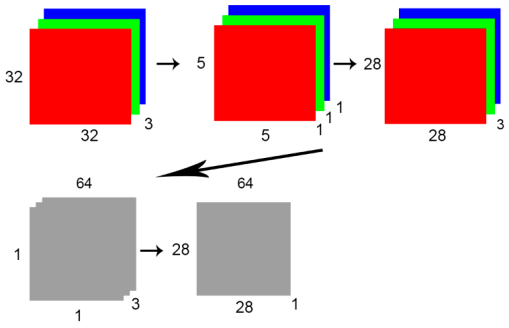


Figura 12: Depthwise Separable Convolution

$$3 \times 5 \times 5 \times 1 \times 28 \times 28 = 58800$$
$$64 \times 1 \times 1 \times 3 \times 28 \times 28 = 150528$$

Obtenemos un total de 209 328 multiplicaciones

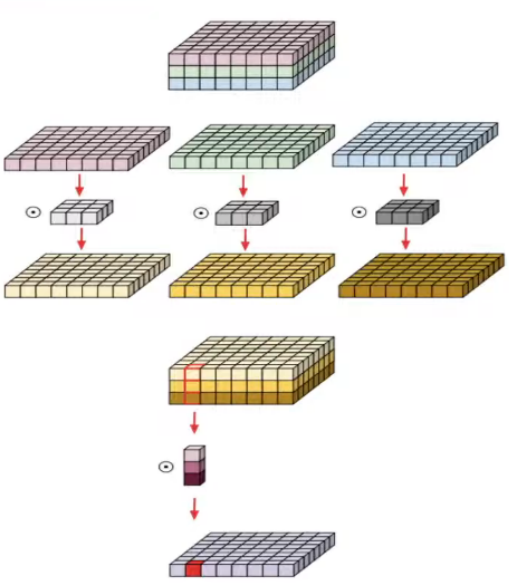


Figura 13: Depthwise Separable Convolution

Una vez definidas todas las redes que vamos a utilizar, vamos a comparar cada una para ver que podemos hacer usando el método de Tranfer Learning.

5.3. Transfer Learning

Hablemos de la manera más rápida y fácil de construir un modelo de aprendizaje profundo, sin preocuparse demasiado por la cantidad de datos que tengamos. El entrenamiento de un modelo profundo puede requerir muchos datos y recursos computacionales, pero por suerte tenemos el aprendizaje por transferencia (Transfer Learning).

El aprendizaje por transferencia es particularmente frecuente en las tareas relacionadas con la visión por ordenador. Los estudios han demostrado que las características aprendidas de conjuntos de imágenes muy grandes, como el de ImageNet, son altamente transferibles a una variedad de tareas de reconocimiento de imágenes. Hay varias maneras de transferir el conocimiento de un modelo a otro. Quizás la forma más fácil es cortar la capa superior del modelo ya entre-

nado y reemplazarla por una inicializada al azar o utilizar estas características para otras tareas. Se puede pensar que este método utiliza el modelo transferido como un extractor de características, ya que la parte fija actúa como un extractor de características y la capa superior actúa como una red neural tradicional totalmente conectada.

¿Por qué la transferencia de aprendizaje funciona? ¿Qué tipo de información útil puede ofrecer un modelo para realizar una tarea en la que no ha sido entrenado? Una propiedad interesante de las redes convolucionales profundas es que cuando se entrenan en un gran conjunto de imágenes, los primeros parámetros de la capa se asemejan entre sí, independientemente de la tarea específica en la que se hayan entrenado. Por ejemplo, las redes convolucionales tienden a aprender bordes, texturas y patrones en las primeras capas. Estas capas parecen capturar las características que son ampliamente útiles para el análisis de las imágenes. Las características que detectan bordes, esquinas, formas, texturas y diferentes tipos de iluminadores pueden considerarse como extractores genéricos de características y ser utilizados en muchos tipos diferentes de entornos, como es el caso de este TFG como veremos más adelante. Cuanto más nos acercamos a la salida, más características específicas tienden a aprender las capas, como las partes y objetos de los objetos.

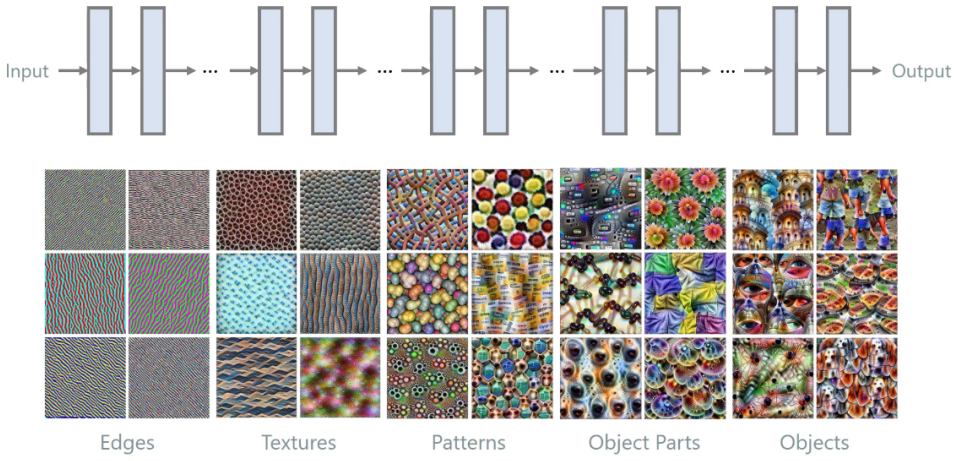


Figura 14: Características

5.4. Fine Tuning

Siguiente con el tema anterior, en el ajuste fino (Fine tuning), primero cambiamos la última capa para que coincida con las clases de nuestro conjunto de datos, como hemos hecho antes con el aprendizaje por transferencia. Pero además, también re-entrenamos las capas de la red que queremos. Lo que hicimos en el ejemplo anterior fue cambiar sólo las capas de la etapa de clasificación, manteniendo el conocimiento que la red obtuvo al extraer las características (patrones), de la cual estamos cargando los pesos (ImageNet). Con el ajuste fino no nos limitamos a reciclar sólo la etapa de clasificación (es decir, las capas totalmente conectadas), sino que lo que haremos es reciclar también la etapa de extracción de características, es decir, las capas de convolución y de agrupación.

¿Cuándo hago el ajuste y la transferencia de aprendizaje? ¿Cómo elijo de qué capa me voy a re-entrenar? Bueno, como regla general, lo primero que haremos es transferir el aprendizaje, es decir, no volveremos a entrenar a nuestra red. Eso nos dará una línea de base que tendremos que podemos ir mejorando y también haciendo pruebas. Luego, sólo re-entrenaremos la etapa de clasificación, y luego podemos intentar re-entrenar algún bloque convolucional, para ver si hemos mejorado o no.

También depende del tipo de problema que tengamos. Si:

- El nuevo conjunto de datos es pequeño y similar al original: Tal vez sea mejor elegir las características de la última capa de la etapa convolucional y usar un clasificador lineal.
- El nuevo conjunto de datos es grande y similar al original: Al tener más datos probablemente no nos sobreajustaremos con los resultados, por lo que podemos o no afinar más capas.
- El nuevo conjunto de datos es pequeño y muy diferente del original: Sería mejor utilizar características de una capa anterior de la etapa convolucional, ya que ésta se establecerá en patrones más generales que las capas posteriores, y luego utilizar un clasificador lineal.
- El nuevo conjunto de datos es grande y muy diferente del original: Lo

entrenaremos desde cero. Sin embargo, aún se recomienda inicializar los pesos con los de la ImageNet.

6. Tecnologías

En este apartado vamos a explicar cada una de las tecnologías usadas y la manera en la que la vamos a utilizar, tanto para la parte de investigación como para la parte de desarrollo, tanto de el Backend como la del Frontend. En este aspecto se han utilizado diversas tecnologías y librerías para cada apartado, nombraremos solo las más importantes.

6.1. Tecnologías usadas

6.1.1. Keras

Para la parte de los modelos se ha usado Keras. Keras es una biblioteca de Redes Neuronales de Código Abierto escrita en Python. Se ejecuta sobre TensorFlow y está especialmente diseñada para posibilitar la experimentación en más o menos poco tiempo con redes de deep learning (aprendizaje profundo). Sus ventajas son:

- Amigable para el usuario (Documentación)
- Modular
- Extensible

Keras sigue las mejores prácticas para ser usable por cualquier desarrollador: ofrece APIs consistentes y simples, minimiza el número de acciones de usuario requeridas para casos de uso común, y proporciona mensajes de error que son comprensibles por el usuario. También cuenta con una amplia documentación y guías para el desarrollador. Keras es el framework de aprendizaje profundo más utilizado en Kaggle. Debido a que Keras facilita la realización de nuevos experimentos, te permite probar más ideas en menos tiempo.

De este framework se ha usado los modelos que se han mencionado anteriormente, y con estos hemos experimentado la extracción de características con cada uno de dichos modelos sobre las imágenes de nuestro dataset.

6.1.2. OpenCV

OpenCV (Open Source Computer Vision) es una librería open source de visión por computador, análisis de imagen y aprendizaje automático. Es una librería originalmente desarrollada por Intel. Dispone de infinitud de algoritmos que permiten, con sólo unas pocos líneas de código, identificar rostros, reconocer objetos, clasificar objetos, detectar movimientos de manos, etc. Nos permite realizar lo siguiente y mucho más:

- Leer y escribir imágenes.
- Detección de rostros y sus características.
- Detección de formas como círculos, rectángulos, etc. en una imagen.
- Reconocimiento de texto en imágenes.
- Modificar la calidad de la imagen y los colores.
- Desarrollando aplicaciones de realidad aumentada.

Para este trabajo se ha usado OpenCV para el procesamiento de las imágenes previo a la extracción de las características y así poder aplicar cada uno de los modelos a dichas imágenes procesadas. Así mantenemos la homogeneidad de las imágenes y para que se vean mucho mejor que la imagen original, todas estas imágenes son procesadas en cada modelo para obtener mejores resultados.

6.1.3. Annoy

Hay algunas otras bibliotecas para hacer una búsqueda de los vecinos más cercanos. Annoy es bastante rápido en este aspecto, pero en realidad hay otra característica que realmente distingue a Annoy: tiene la capacidad de usar archivos estáticos como índices. En particular, esto significa que puede compartir el índice a través de los procesos. Annoy también desacopla la creación de índices de

la carga de los mismos, de modo que puede pasar los índices como archivos y asignarlos a la memoria rápidamente. Otra cosa buena de Annoy es que trata de minimizar la huella de la memoria para que los índices sean bastante pequeños. Annoy funcionan usando proyecciones aleatorias y construyendo un árbol. En cada nodo intermedio del árbol se elige un hiperplano aleatorio que divide el espacio en dos subespacios. Este hiperplano se elige tomando una muestra de dos puntos del subconjunto y tomando el hiperplano equidistante de ellos. Esto lo hacemos k veces para obtener un bosque de árboles. k tiene que ser ajustado mirando qué equilibrio tiene mejor precisión y rendimiento.

Para resumir el funcionamiento de Annoy:

- Empezamos con un conjunto de puntos
- Partimos el conjunto en 2 subconjuntos aleatorios eligiendo dos puntos
- Construimos un árbol binario con cada división
- Buscamos en todos los arboles para encontrar K items
- Removemos los duplicados
- Miramos la distancia entre estos items
- Devolvemos los K items mas cercanos

6.1.4. Pandas

Pandas es una herramienta de manipulación de datos de alto nivel. Es construido con el paquete Numpy y su estructura de datos clave se denomina DataFrame. El DataFrame te permite almacenar y manipular datos tabulados en filas de observaciones y columnas de variables. Pandas es un paquete de Python que proporciona estructuras de datos rápidas, flexibles y expresivas diseñadas para hacer que el trabajo con datos relacionales.º .etiquetados"sea fácil e intuitivo. Además, tiene el objetivo más amplio de convertirse en la herramienta de análisis y manipulación de datos de código abierto más potente y flexible disponible en cualquier idioma.

Para este proyecto se ha usado Pandas para ir trabajando con los datos de las imágenes como los tags y el impresor de dicha imagen. También para poder usar este DataFrame como punto de partida para la extracción de las características de los pasos posteriores y poder almacenar los vectores característicos de dichas imágenes directamente en este DataFrame.

6.1.5. Django

Django es un framework de alto nivel para la web hecho con el lenguaje de Python que fomenta un rápido desarrollo y un diseño limpio y pragmático. Construido por desarrolladores experimentados, se encarga de gran parte de las molestias del desarrollo web, de modo que puedes concentrarte en escribir tu aplicación sin necesidad de reinventar la rueda. Es libre y de código abierto. Django tienes las siguiente ventajas:

- Django fue diseñado para ayudar a los desarrolladores a llevar las aplicaciones desde el concepto hasta su finalización lo más rápidamente posible.
- Django se toma la seguridad muy en serio y ayuda a los desarrolladores a evitar muchos errores de seguridad comunes.
- Algunos de los sitios más concurridos de la web aprovechan la capacidad de Django para escalar de forma rápida y flexible.

En este proyecto se ha usado Django ya que es un framework que se ha visto en la carrera y también porque usa el lenguaje de Python y a parte porque es uno de los lenguajes que se complementa con Keras y Pandas para poder integrar todo de manera más fácil. Por otro lado, es tenemos un framework bastante potente para poder crear una API de manera sencilla y que sea escalable para poder conectarla a cualquier aplicación Frontend.

6.1.6. Angular

Angular es un framework Javascript potente, muy adecuado para el desarrollo de aplicaciones frontend modernas, de complejidad media o elevada. El tipo de aplicación Javascript que se desarrolla con Angular es del estilo SPA (Single Page Application) o también las denominadas PWA (Progressive Web App).

El framework Angular ofrece una base para el desarrollo de aplicaciones robustas, escalables y optimizadas, que promueve además las mejores prácticas y un estilo de codificación homogéneo y de gran modularidad. Aunque ofrece principalmente una base para el desarrollo de la parte de Frontend, la programación Javascript del lado del cliente, también aborda técnicas de desarrollo de la parte del Backend. Las ventajas de Angular son las siguiente:

- Construye aplicaciones móviles nativas con estrategias de Cordova, Ionic o NativeScript.
- Herramientas de la línea de comandos.
- Sistema de plantillas propio.
- Ofrece división automática del código para que los usuarios sólo carguen el código necesario para renderizar la vista que solicitan.

Para este proyecto se ha elegido Angular para el desarrollo de la parte de Frontend de la web por el simple hecho de que lo he empezado a aprender en Ingenieria de Software y porque se quería seguir aprendiendo un poco mas sobre este framework y por ende ya estaba familiarizado con el lenguaje de Javascript.

7. Análisis

7.1. Análisis de requisitos

Se tiene un en cuenta una serie de requisitos que se han de cumplir para la aplicación. Estos requisitos marcaran la linea de trabajo por la cual seguir con todo el proyecto para llevar a cabo la implementación del mismo. Y saber que es lo que se ha de tener como mínimo en nuestra aplicación y proyecto en general.

Análisis de datos e imágenes: Queremos tener una idea global de los datos que estamos manejando, tanto para los datos que nos suministran en formato de texto así como el de las imágenes para su posterior procesamiento. Se irán analizando tanto antes como durante el desarrollo de todo el proyecto como iremos

viendo a lo largo de la implementación.

Base de datos de imágenes e impresores: La creación de una base de datos sencilla con todos los impresores y todas la imágenes con su correspondientes etiquetas. Esta base de datos servirá como base para el posterior desarrollo de la aplicación. Al no tener acceso directo a la base de datos original tenemos que crear esta base de datos a partir de los datos que nos suministran para poder realizar una demo en condiciones.

Extracción de imágenes similares: Tener una solución al problema de devolver al usuario imágenes similares a una imagen seleccionada, teniendo el cuenta el parecido visual con las otras imágenes del conjunto de imágenes, en este caso almacenadas en la base de datos. Tenemos que tener un cuenta la respuesta del usuario final para la comprobación de que el trabajo de obtención de imágenes es el correcto.

Interfaz para interactuar con la imágenes e impresores: Tenemos que tener en cuenta la interfaz que manejara el usuario y como obtendrá dichas imágenes de cada uno de los impresores, en este aspecto queremos una interfaz sencilla que nos permita ver los resultados obtenidos de la extracción de la características y la de las imágenes similares obtenidas. Como en el punto anterior esta vendrá dada por la valoración del usuario final a dicho trabajo.

7.2. Análisis de datos

En primer lugar, antes de la extracción de las características de las imágenes con alguno de los modelos se tiene que procesar y analizar los datos que se nos ha entregado. En este caso no tenemos un acceso directo a las imágenes ni a la base de datos. Con lo cual en primera instancia se estaba viendo si se podía tener acceso a los mismos, pero con resultados inciertos. Por parte del usuario se nos suministro de un TXT con los datos de los impresores y las URLs de las imágenes en un formato no muy amigable, con lo cual con la ayuda del usuario se ha podido entender dicho formato y a partir de aquí crear un script de Python para poder procesar dicho fichero de texto.

Como podemos ver en la siguiente imagen las filas que empiezan con un valor de 100 son impresores y las filas que siguen abajo son marcas de dichos impresores con sus respectivos URLs de la imagen. En este fichero de texto tenemos varios símbolos de los cuales tenemos que tener cuidado de eliminar.

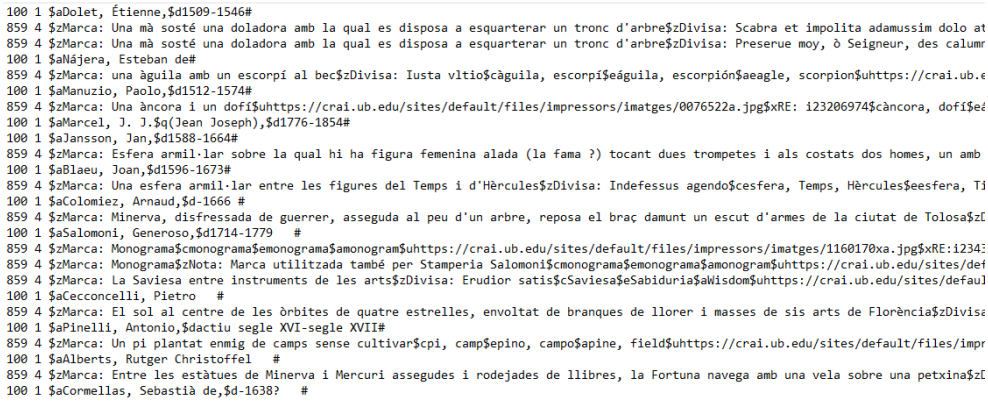


Figura 15: Fichero de texto

Una vez procesado dicho archivo se nos genera un DataFrame con el siguiente formato. Una columna con una lista de tags, una columna con una la lista de URLs de las imágenes y otra para el impresor. En este DataFrame también hemos eliminado a los impresores que no tenían imágenes y también las imágenes sin tags, ya que para la parte de validación y métricas no nos servirán.

	tag	filename	author
0	[destral, tronc]	[https://crai.ub.edu/sites/default/files/impre...	Dolet, Étienne 1509-1546
1	[aguila, escorpi]	[https://crai.ub.edu/sites/default/files/impre...	Nájera, Esteban de
2	[ancora, dofi]	[https://crai.ub.edu/sites/default/files/impre...	Manuzio, Paolo 1512-1574
3	[esfera, hercules]	[https://crai.ub.edu/sites/default/files/impre...	Blaeu, Joan 1596-1673
4	[minerva, tolosa]	[https://crai.ub.edu/sites/default/files/impre...	Colomiez, Arnaud -1666

Figura 16: DataFrame inicial

Al no tener acceso a las imágenes se tuvo que descargar cada una haciendo una petición directamente a cada URL con un tiempo de espera entre descargas para

evitar un posible bloqueo en la IP. Una vez descargada cada una, con el mismo nombre del archivo original que esta en la web, se guardo cada imagen en una carpeta para tenerlas directamente en el ordenador para poder trabajar mejor con ellas. Después de esto se cambio la columna de URLs para que tuviera el nombre de los archivos en vez de la URL.

	tag	filename	author
0	[destral, tronc]	[10247993b.jpg]	Dolet, Étienne 1509-1546
1	[aguila, escorpi]	[0127615a.jpg]	Nájera, Esteban de
2	[ancora, dofi]	[0076522a.jpg, 0076522b.jpg, 0076522c.jpg]	Manuzio, Paolo 1512-1574
3	[esfera, hercules]	[0009492a.jpg, 0009492b.jpg]	Blaeu, Joan 1596-1673
4	[minerva, tolosa]	[0000005a.jpg, 11601693a.jpg]	Colomiez, Arnaud -1666

Figura 17: DataFrame con nombre de archivos

Después de esto se aplicaron mas transformaciones al DataFrame para separar cada archivo de imagen con su respectivo impresor y sus tags.

	tag	filename	author
0	[destral, tronc]	10247993b.jpg	Dolet, Étienne 1509-1546
1	[aguila, escorpi]	0127615a.jpg	Nájera, Esteban de
2	[ancora, dofi]	0076522a.jpg	Manuzio, Paolo 1512-1574
3	[ancora, dofi]	0076522b.jpg	Manuzio, Paolo 1512-1574
4	[ancora, dofi]	0076522c.jpg	Manuzio, Paolo 1512-1574

Figura 18: DataFrame con tags por imagen

Ahora tenemos que analizar cada tag, para esto se realizo una exploración de los mismo y para poder ver que tags solo aparecen una solo vez. En este caso obtenemos un DataFrame con un total de 2189 imágenes.

	tag	count
0	destral	1
1	arada	1
2	llebrer	1
3	reina	1
4	jordi	1

Figura 19: Tags

Como podemos ver tenemos algunos tags que solo aparecen una única vez, con lo cual no queremos tener estos tags, ya que para el apartado de para métricas no servirían de nada y al tener un método para comprobar que hemos obtenido imágenes que contengan el mismo tag, no obtendríamos los resultados deseados. Si analizamos todos los tags podemos ver lo siguiente:

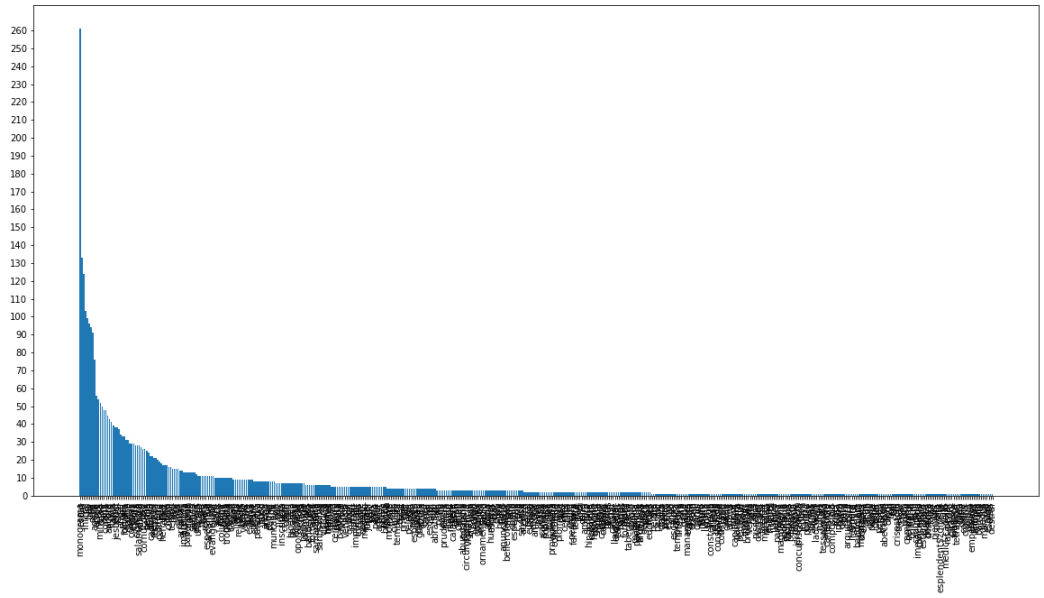


Figura 20: Gráfico de tags

Hay muchos tags los cuales se repiten muy poco. Lo que se ha hecho es trabajar con los tags que se repitan más veces, en este caso los tags que se repitan menos

de 20 veces fueron eliminados y solo se trabajo con el resto, esto para temas de validación y para luego que en el fine tuning tengamos un número considerable de imágenes con tags parecidos, en el caso de que tengamos imágenes con tags que aparecen 1 o 2 veces no podemos asegurarnos de que obtengamos K imágenes parecidas y validar todas de manera homogénea o tengan al menos un tag valido en las respuestas devueltas. Una vez eliminado dichos tags nos quedamos con un total de 1577 imágenes.

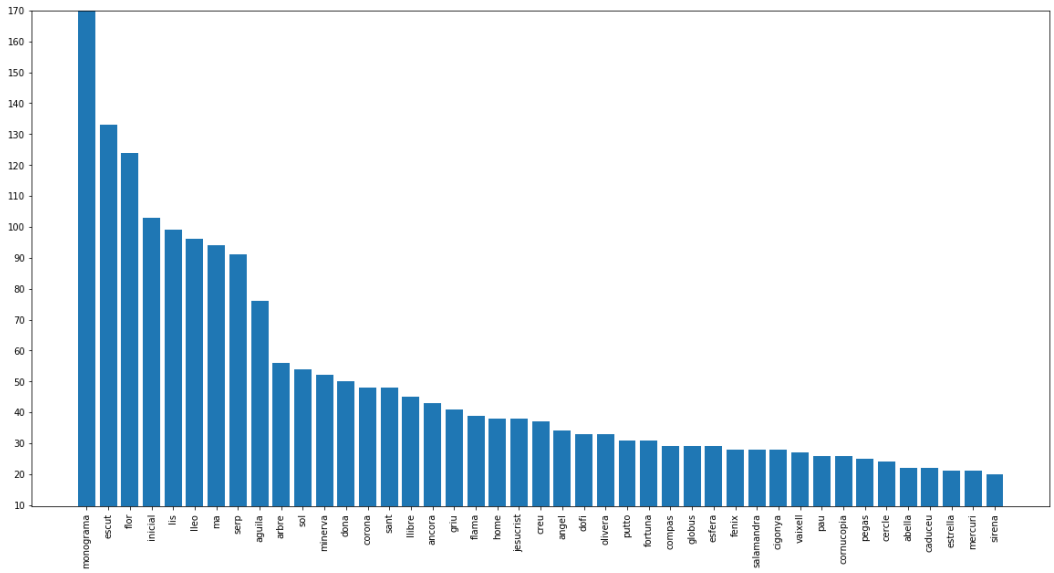


Figura 21: Gráfico de tags

Ahora se puede analizar el gráfico de tags y como podemos ver tenemos 43 tags validos los cuales tiene 20 o mas imágenes con ese tag. También podemos observar que el tag Monograma aparece muchas más veces que el resto, es uno de los tags donde más imágenes tenemos. Lo que haremos ahora sera guardar este DataFrame con los tags e imágenes para poder usarlos con los modelos, el resultado del DataFrame final es el siguiente:

:

	tag	filename	author
0	(aguila,)	0127615a.jpg	Nájera, Esteban de
1	(ancora, dofi)	0076522a.jpg	Manuzio, Paolo 1512-1574
2	(ancora, dofi)	0076522b.jpg	Manuzio, Paolo 1512-1574
3	(ancora, dofi)	0076522c.jpg	Manuzio, Paolo 1512-1574
4	(esfera,)	0009492a.jpg	Blaeu, Joan 1596-1673
5	(esfera,)	0009492b.jpg	Blaeu, Joan 1596-1673

Figura 22: DataFrame final

7.3. Análisis de capas convolucionales

Como un una primera instancia se tenia pensado solo usar MobileNet vamos a ver como se comporta sus redes convolucionales dada una imagen y también vamos a visualizar si lo que hemos dicho es verdad, de que en capas tempranas tenemos solo características básicas y en capas más profundas cosas más especificas. En capas mucho más profundas la visualización es más complicada ya que los valores mostrados no son comprensibles para el ojo humano.

La imagen que vamos a analizar sera la siguiente:

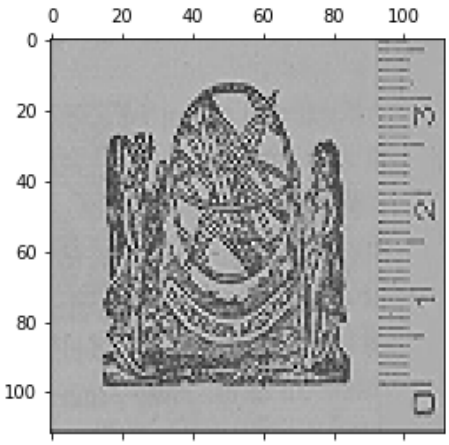


Figura 23: Imagen a analizar

Como podemos ver en las primeras capas podemos ver perfectamente la imagen, en este caso se aplican filtros sencillos.

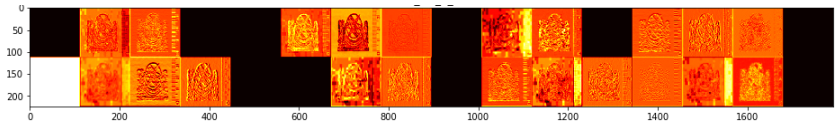


Figura 24: Capa 1

En la capa 4 podemos ver aun la imagen, pero también zonas de interés bastante amplias donde se encuentra el objeto con mayor dimensión.

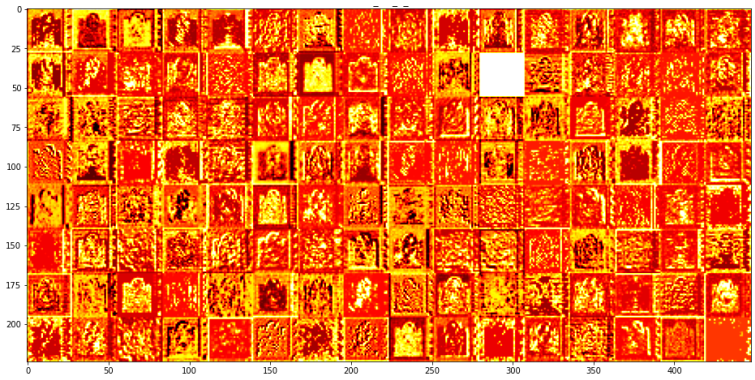


Figura 25: Capa 4

En la capa 6 ya nos cuesta más distinguir la imagen, pero aun podemos ver zona importantes con un valor mayor que detectan estos filtros.

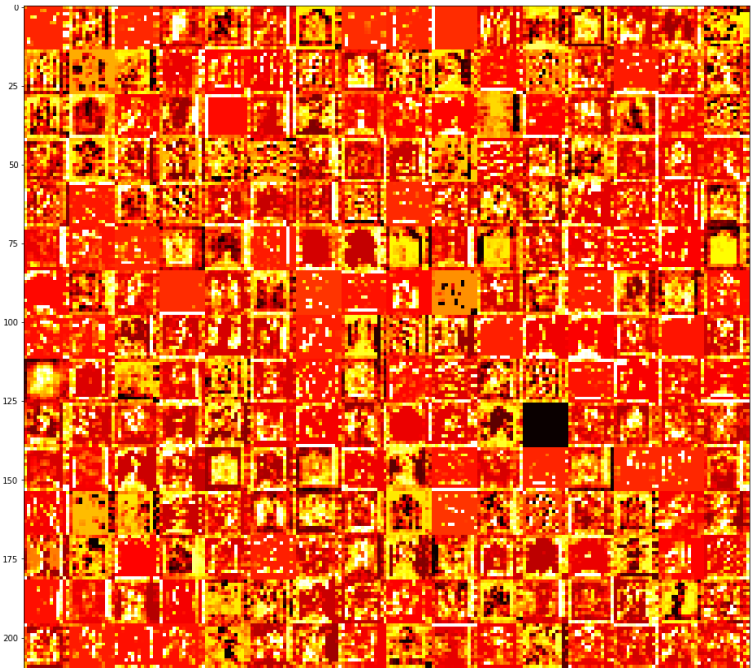


Figura 26: Capa 6

En la capa 12 ya se hace mucho más difícil distinguir alguna cosa de la imagen original. Y en capas más profundas la interpretación visual es casi imposible.

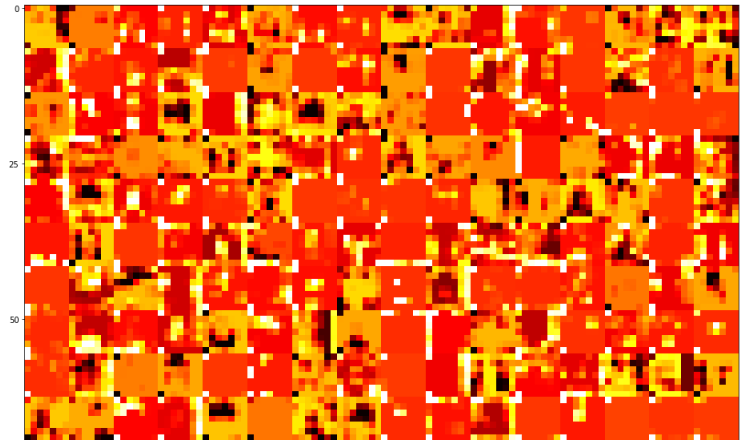


Figura 27: Capa 6

8. Implementación

En este apartado vamos a ver en detalle todo el desarrollo que se ha llevado a cabo, en este caso lo vamos a dividir en diferentes partes en las cuales se explicara lo que se ha hecho en cada uno.

8.1. Procesamiento de imágenes

Antes de trabajar con las redes tenemos que procesar las imágenes, ya que las imágenes tiene artefactos a eliminar y objetos que no son de interés para la obtención de características y que pueden ser mal interpretadas por la red como veremos a continuación. Con lo cual, lo que se ha hecho a continuación es procesar la imagen con tal de obtener una imagen mucho más definida y recortada. La imagen que queremos procesar en la siguiente:

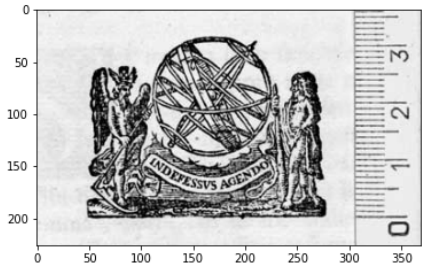


Figura 28: Imagen antes de procesarla

En este caso lo que se ha hecho es usando la librería de OpenCV buscar el objeto mas grande dentro de la imagen y recortar basándonos en este objeto. OpenCV nos brida con funciones para este cometido. Una vez que la imagen sea recortada lo que haremos es hacer un resize manteniendo el aspect ratio de la imagen y convirtiéndola en una imagen de dimensión (224,224) que es el tamaño del input de las redes.

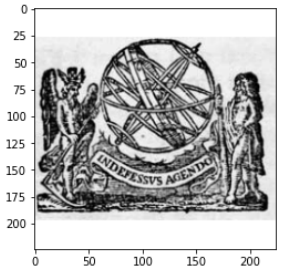


Figura 29: Imagen después de procesarla

Dicha nueva imagen se guarda por separado y el nombre del archivo es diferente al original, este nuevo nombre de archivo es guardado en una nueva columna del DataFrame con el cual estamos trabajando.

	tag	filename	author	processed_image
0	(aguila,)	0127615a.jpg	Nájera, Esteban de	0127615a0.jpg
1	(ancora, dofi)	0076522a.jpg	Manuzio, Paolo 1512-1574	0076522a1.jpg
2	(ancora, dofi)	0076522b.jpg	Manuzio, Paolo 1512-1574	0076522b2.jpg
3	(ancora, dofi)	0076522c.jpg	Manuzio, Paolo 1512-1574	0076522c3.jpg
4	(esfera,)	0009492a.jpg	Blaeu, Joan 1596-1673	0009492a4.jpg
5	(esfera,)	0009492b.jpg	Blaeu, Joan 1596-1673	0009492b5.jpg
6	(minerva,)	0000005a.jpg	Colomiez, Arnaud -1666	0000005a6.jpg

Figura 30: Imagen procesadas

8.2. Transfer Learning

Para este apartado vamos a poner a prueba el conocimiento de las redes convolucionales. En este caso queremos extraer las características de cada imagen y obtener un vector de N dimensiones para poder procesarlas mediante un algoritmo de KNN así poder obtener las K imágenes parecidas y comprobar que sean correctas. En este caso se han comparado las 3 redes mencionadas anteriormente, que son modelos ya pre-entrenamos: VGG16, ResNet y MobileNet.

Para esto se han procesado las imágenes y extraído los vectores característicos de cada imagen que tenemos en el DataFrame, con lo cual tenemos 3 columnas en el DataFrame para cada modelo con el vector característico correspondiente. En

este caso por cada modelo lo que se ha hecho es leer dicho modelo de la librería Keras e inicializar el modelo teniendo en cuenta una serie de parámetros:

```
applications.mobilenet.MobileNet(
    weights='imagenet',
    include_top=False,
    pooling='avg')
```

Figura 31: Ejemplo con MobileNet

- weights: Cargamos los pesos de ImageNet
- include top: Incluimos la parte de clasificación
- pooling: Tipo de pooling de salida

Queremos solo el extractor de características de cada modelo, los parámetros son iguales para los 3, con esto solo nos quedamos con un vector único que es el descriptor de la imagen. En este caso cada red nos devuelve un vector de diferente dimensión para cada imagen, dicho vector es el descriptor de la imagen.

- VGG16: Vector de 512 valores.
- ResNet: Vector de 2048 valores.
- MobileNet: Vector de 1024 valores.

Con lo cual obtendremos el siguiente DataFrame:

	tag	filename	author	processed_image	vgg_features	resnet_features	mobilenet_features
0	(aguila,)	0127615a.jpg	Nájera, Esteban de	0127615a0.jpg	[2.566443, 3.376925, 4.0354877, 0.0, 0.6963096...	[0.0, 0.040823843, 0.0, 0.0, 0.000846392, 0.04...	[0.1495454, 0.29908353, 0.004312407, 0.0355722...
1	(ancora, dofi)	0076522a.jpg	Manuzio, Paolo 1512-1574	0076522a1.jpg	[0.0, 2.167516, 0.0588582, 0.0, 4.356243, 0.0,...	[0.0, 0.0, 0.0, 0.0, 0.011631879, 0.0, 0.08793...	[0.03204904, 0.71795064, 0.16496207, 0.8374542...
2	(ancora, dofi)	0076522b.jpg	Manuzio, Paolo 1512-1574	0076522b2.jpg	[0.593719, 0.9669736, 0.0, 0.0, 0.13504304, 3....	[0.00066297204, 0.0, 0.0, 0.0, 0.0005833664, 0...	[0.049377967, 0.69437176, 0.01733708, 0.0, 3.2...
3	(ancora, dofi)	0076522c.jpg	Manuzio, Paolo 1512-1574	0076522c3.jpg	[10.875512, 2.3147213, 0.0, 0.0, 0.0, 0.733971...	[0.0, 0.0, 0.0, 0.0, 0.0039984463, 0.0, 0.1292...	[0.07315579, 0.0, 0.006026902, 0.0, 1.4178928,...
4	(esfera,)	0009492a.jpg	Blaeu, Joan 1596-1673	0009492a4.jpg	[5.6829286, 5.6957893, 0.033854265, 0.33120558...	[0.71807986, 0.0, 0.0, 0.0, 0.06354012, 0.3178...	[0.6132733, 0.122206554, 0.3569663, 0.0, 0.113...
5	(esfera,)	0009492b.jpg	Blaeu, Joan 1596-1673	0009492b5.jpg	[2.8403537, 13.236951, 0.8516058, 1.1844817, 0...	[0.0002672961, 0.21222597, 0.0, 0.014192433, ...	[0.43555793, 0.6272333, 1.555637, 0.28966957, ...
6	(minerva,)	0000005a.jpg	Colomiez, Arnaud -1666	0000005a6.jpg	[1.7734082, 14.466338, 0.13315396, 0.0, 0.0, 0...	[0.2856313, 0.0, 0.019643085, 0.0, 0.10856461,...	[0.82734376, 2.4228055, 0.29782745, 0.00106083...

Figura 32: Vectores descriptores

Una vez que tenemos todos los vectores para todas las imágenes lo que se ha hecho es normalizar los vectores ya que se ha demostrado que dan mejores resultados. La normalización se usa para modificar los valores en el vector característico y así poder medirlos en una escala común. Existen varios métodos para esto, así que vamos a presentarles los dos más comunes.

- Normalización L1: Busca las menores desviaciones absolutas, y funciona asegurándose que la suma de los valores absolutos en cada fila sea 1.
- Normalización L2: Busca los menores cuadrados, y funciona asegurándose que la suma de los cuadrados sea 1.

Se ha demostrado que la normalización L2 da mejores resultados para trabajar directamente con los vectores característicos.

$$\sum_{j=1}^{p} ||x_{ij}||^2 = 1, \qquad \forall i = 1, \ldots, n$$

Una vez normalizado los vectores se ha agregado una nueva columna para estos datos normalizados.

	tag	filename	author	processed_image	vgg_features	resnet_features	mobilenet_features	vgg_features_norm	resnet_features_norm	mobilenet_features_norm
0	(aguila,)	0127615a.jpg	Nájera, Esteban de	0127615a0.jpg	[2.566443, 3.376925, 4.0354877, 0.0, 0.6963096...	[0.0, 0.040823843, 0.0, 0.0, 0.000846392, 0.04...	[0.1495454, 0.29908353, 0.004312407, 0.0355722...	[0.028017194615612265, 0.036850174309654 5, 0...	[0.0, 0.0014012941186315956, 0.0, 0.0, 0.0, 2.90527...	[0.0046605023684726 5, 0.0093207781162304 44, 0....
1	(ancora, dofi)	0076522a.jpg	Manuzio, Paolo 1512-1574	0076522a1.jpg	[0.0, 2.167516, 0.0588582, 0.0, 4.356243, 0.0,...	[0.0, 0.0, 0.0, 0.0, 0.011631879, 0.0, 0.08793...	[0.0320490 4, 0.71795064, 0.16496207, 0.8374542...	[0.0, 0.022685602563512 01, 0.000616020254353 30...	[0.0, 0.0, 0.0, 0.0, 0.0, 0.0004412368225094 0363, 0...	[0.0011868891589065294, 0.026588247284230 94, 0...
2	(ancora, dofi)	0076522b.jpg	Manuzio, Paolo 1512-1574	0076522b2.jpg	[0.593719, 0.9669736, 0.0, 0.0, 0.13504304, 3....	[0.00066297204, 0.0, 0.0, 0.0, 0.0005833664, 0...	[0.049377967, 0.69437176, 0.01733708, 0.0, 3.2...	[0.005543153540862692, 0.009027979734026 23, 0....	[2.1772126802722854e-05, 0.0, 0.0, 0.0, 1.9157...	[0.0015717581981214723, 0.022102661970655132, ...
3	(ancora, dofi)	0076522c.jpg	Manuzio, Paolo 1512-1574	0076522c3.jpg	[10.875512, 2.3147213, 0.0, 0.0, 0.0, 0.733971...	[0.0, 0.0, 0.0, 0.0, 0.0039984463, 0.0, 0.1292...	[0.07315579, 0.0, 0.006026902, 0.0, 1.4178928,...	[0.10746015731441784, 0.022871596036484594, 0....	[0.0, 0.0, 0.0, 0.0, 0.0001333441647893396 4, 0...	[0.0026603194537203126, 0.0, 0.0002191690350 28...
4	(esfera,)	0009492a.jpg	Blaeu, Joan 1596-1673	0009492a4.jpg	[5.6829286, 5.6957893, 0.033854265, 0.33120558...	[0.71807986, 0.0, 0.0, 0.0, 0.06354012, 0.3178...	[0.6132733, 0.122206554, 0.3569663, 0.0, 0.113...	[0.06311187163518903, 0.063254697217398 13, 0.0...	[0.019165864602818 83, 0.0, 0.0, 0.0, 0.0016959...	[0.022084586479476 65, 0.004400780380861736, 0....

Figura 33: Vectores descriptores normalizados

8.3. Métricas

Para medir cuan bien obtenemos las imágenes similares nos vamos a basar en el numero de imágenes obtenidas y si contiene al menos un tag correcto, si contiene al menos uno lo contaremos como correcto (hit). Para este caso usaremos la métrica de MAP (Mean average precision). Antes tenemos que tener en cuenta la posición de la imagen dentro de las imágenes devueltas, para ello se usa Precision at K. Por defecto, la precisión tiene en cuenta todos los documentos recuperados, pero también puede evaluarse en un número determinado de documentos

recuperados, lo que se conoce comúnmente como rango de corte, en el que el modelo sólo se evalúa considerando únicamente sus consultas más importantes. La medida se denomina precisión en k o P@K.

$$\text{precision} = \frac{|\{\text{relevant documents}\} \cap \{\text{retrieved documents}\}|}{|\{\text{retrieved documents}\}|}$$

Estando familiarizados con la P@K, podemos ahora pasar a calcular la P@K media. Esto nos daría una mejor medición de nuestro modelo en su capacidad de ordenar los resultados de la consulta.

$$AP@K = \frac{1}{retrieved} \sum_{k}^{n} P@k \times rel@k$$

rel@k es una función de relevancia. Es una función indicadora que es igual a 1 si el documento en el rango k es correcto y es igual a 0 en caso contrario.

Por ultimo la media para un numero Q de consultas para hacer la medición global de como se comportan nuestros modelos.

$$\text{MAP} = \frac{\sum_{q=1}^{Q} \text{AveP(q)}}{Q}$$

Para el re-entrenamiento usaremos el F1-score para medir la precisión del modelo a la hora de clasificar. Tiene en cuenta la precisión y el recall. Un valor de 1 nos dice que el modelo es perfecto y un valor de 0 es el valor mas bajo posible.

$$F1 = \frac{2 \cdot precision \cdot recall}{precision + recall}$$

8.4. Vecino mas cercado

Una vez tengamos todos estos vectores para cada imagen pasaremos al apartado de KNN. K-Nearest-Neighbor es un algoritmo basado en instancia de tipo supervisado en el ámbito de Machine Learning. Puede usarse para clasificar nuevas muestras o para predecir. Sirve esencialmente para clasificar valores buscando

los puntos de datos similares por cercanía aprendidos en la etapa de entrenamiento. Este algoritmo nos permitirá medir el grado de similitud de cada vector característico. Para esto se ha usado Annoy que como hemos mencionado anteriormente es una librería que nos ayuda con esta labor. En este caso Annoy tiene varias métricas/distancias para medir la similitud de los vectores así como el numero de arboles que se usaran para dicha medición, cuantos mas arboles mayor precisión pero mayor tiempo de procesado, cuanto menos arboles menor tiempo de procesado pero menor precisión.

Si miramos los vecinos mas cercanos de los vectores sin normalizar obtenemos los siguiente resultados. En este caso Annoy se ha inicializado con 500 arboles, sin los vectores normalizados y con distancia Euclídea.

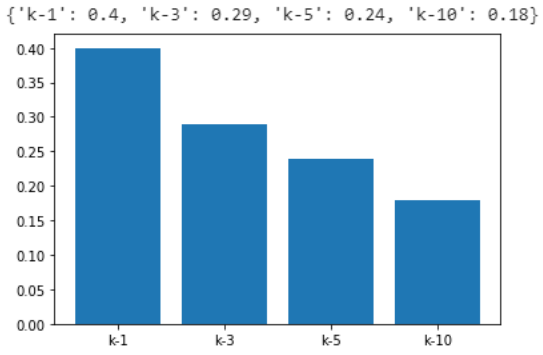


Figura 34: VGG

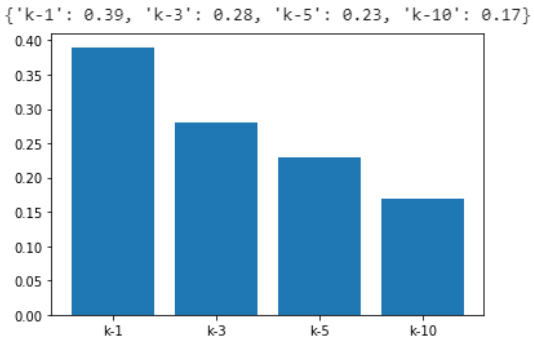


Figura 35: ResNet

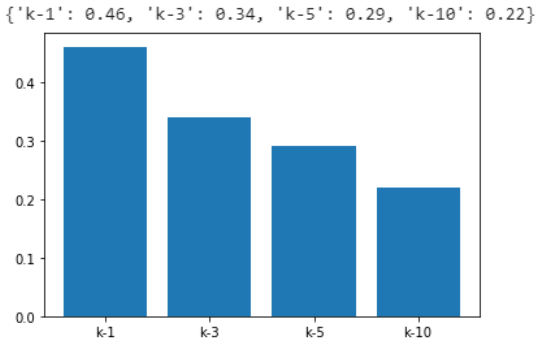


Figura 36: MobileNet

Pero si normalizamos los vectores obtenemos los siguientes resultados. Dependerá de si queremos trabajar con los vectores normalizados o no. En este caso aumentamos la precisión de VGG y ResNet, pero MobileNet sigue siendo superior. Así que nos quedaremos con este modelo para poder seguir analizándolo.

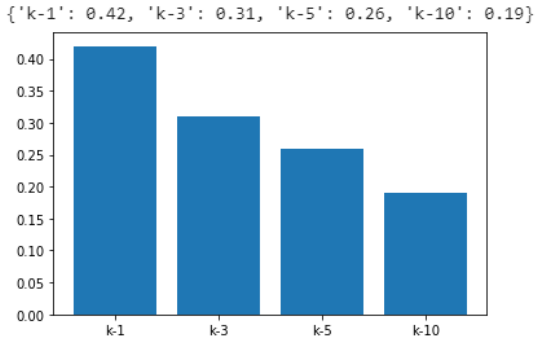


Figura 37: VGG

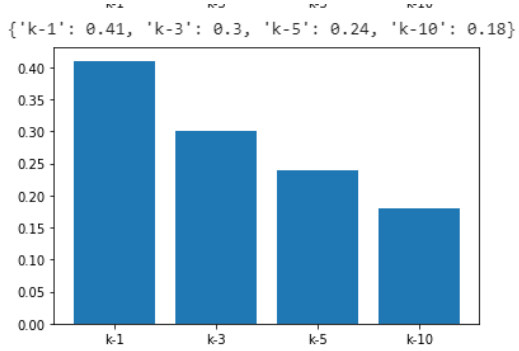


Figura 38: ResNet

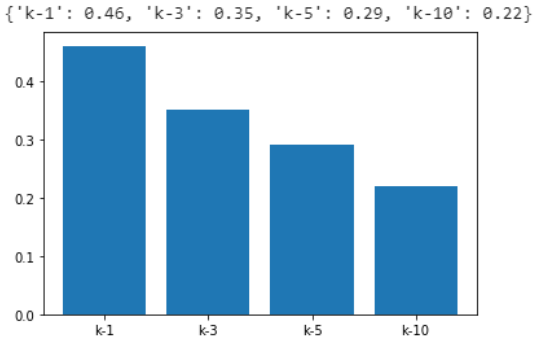


Figura 39: MobileNet

8.5. Filtrando tags

Una vez tenemos el modelo que queremos, analizamos el por qué de esta baja respuesta cuando obtenemos K imágenes mayores que uno. En este caso analizando los tags en los cuales los modelos tienen una baja respuesta, en este caso los tags con una precisión menor de 0.4, podemos observar lo siguiente:

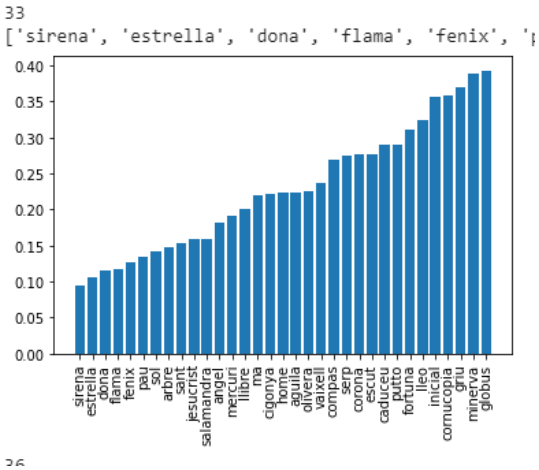


Figura 40: VGG

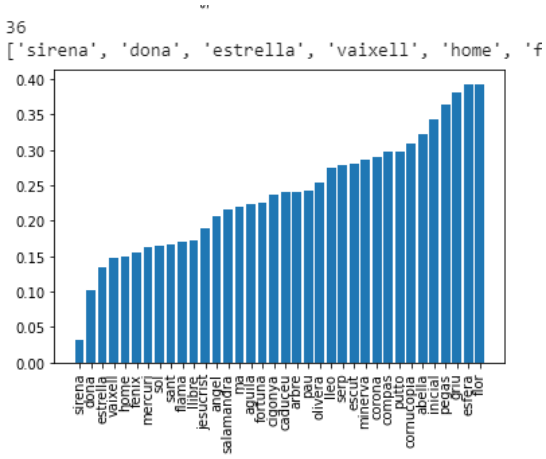


Figura 41: ResNet

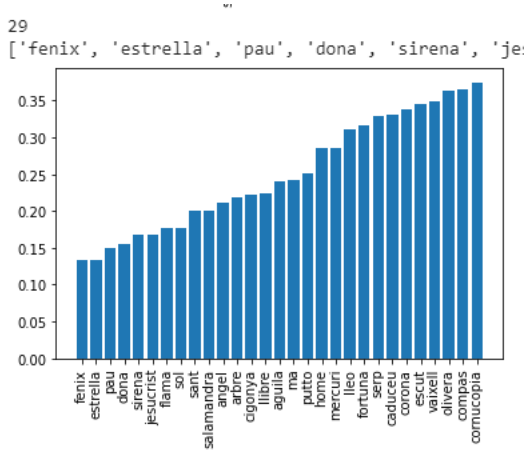


Figura 42: MobileNet

MobileNet tiene menos tags en los cuales se equivoca y ResNet tiene mas tags donde se equivoca. Si analizamos los tags donde tienen mas aciertos los modelos, en este caso donde la precisión sea mayor que 0.4, podemos observar que MobileNet tienes mas tags con alta precisión comparado a los otros modelos.

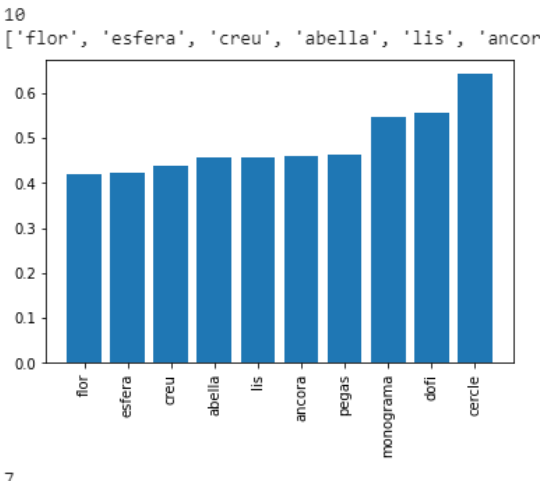


Figura 43: VGG

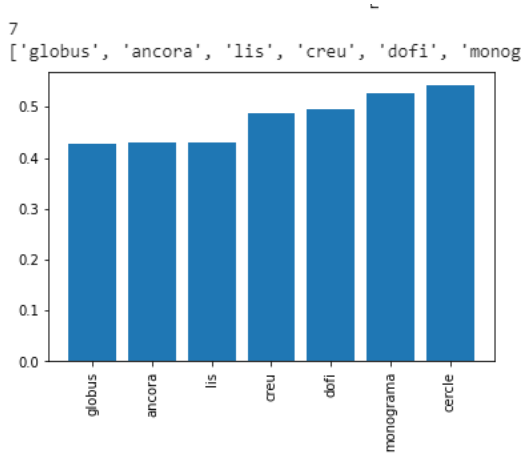


Figura 44: ResNet

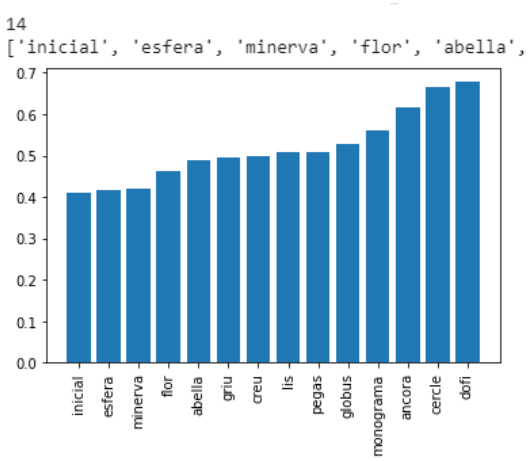


Figura 45: MobileNet

Si analizamos las imágenes que tienen los tags de baja precisión y los tags de alta precisión podemos llegar a una conclusión. Y es que los tags con baja respuesta tienen mas imágenes que no se parecen entre si aun siendo el tag mas usado. Y las imágenes con tags de alta respuesta son imágenes muy parecidas entre si y también tenemos varios ejemplos de estas mismas.

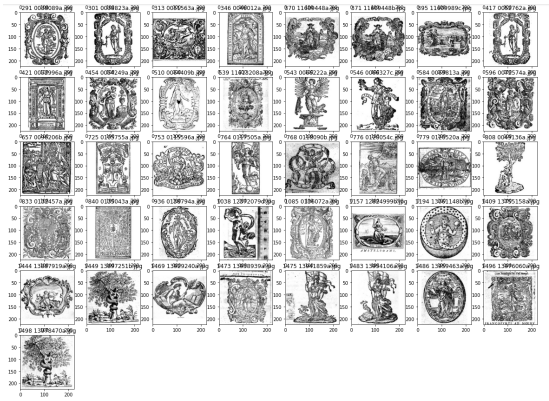


Figura 46: Tag Dona: baja respuesta

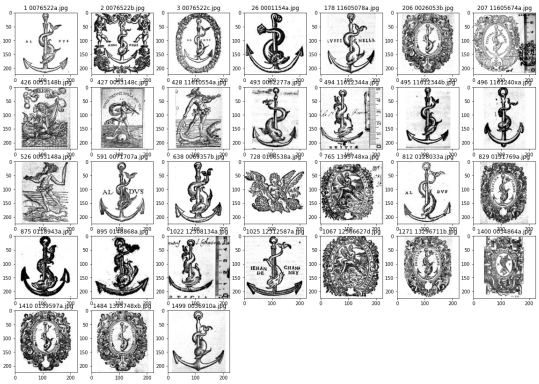


Figura 47: Tag Dofi: alta respuesta

Si queremos hacer un Fine tuning con el modelo de MobileNet necesitamos tener imágenes que se parezcan entre si y tengan tags parecidos, ya que si no el re-entrenamiento no funcionaria, ya que la clasificación final seria errónea. En esta caso lo que haremos sera eliminar los tags con baja respuesta y quedarnos con los tags de alta respuesta, solo para realizar el Fine tuning. Esto no quiere decir que el modelo no extraiga bien las características de una imagen, si tomamos el ejemplo de una imagen con el tag Dona en la que se ve un árbol y obtenemos las K imágenes mas similares, podemos observar que el modelo se comporta de manera correcta aun sabiendo que los tags son incorrectos.

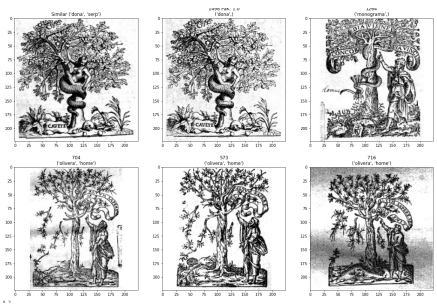


Figura 48: Tag Dona: ejemplo

8.6. Fine tuning

Una vez eliminados los tags con imágenes sin un parecido razonable, vamos a pasar a hacer un Fine tuning del modelo ganador, en este caso MobileNet y comprobar si tenemos una mejoría importante. Para este caso, teniendo los conocimientos mencionados anteriormente en el apartado de Fine tuning vamos a re-entrenar la red agregando un nuevo clasificador para los tags con alta respuesta y re-entrenando el modelo con dichas imágenes.

Podemos usar el modelo de MobileNet sin la capa de clasificación, como la hemos estado usando, y agregarle una capa de clasificación para nuestros tags. Podemos usar una o varias capas, en este caso usaremos una capa Flatten que lo único que hace es aplanar la entrada, una capa oculta con 128 neuronas y una salida de 14, que son los tags con mayor respuesta. Aquí se puede jugar con el número de neuronas y capas, en este caso es recomendable seguir con las indicaciones de la comunidad y ver que es lo que le funciona a la mayoría, así que usaremos una capa oculta con 128 neuronas, para los pocos datos que tenemos con una capa oculta es más que suficiente y también para los recursos disponibles.

```
model = MobileNet(weights='imagenet', include_top=False, input_shape=in_shape)
flat1 = Flatten()(model.layers[-1].output)
class1 = Dense(128, activation='relu', kernel_initializer='he_uniform')(flat1)
output = Dense(out_shape, activation='sigmoid')(class1)
```

Figura 49: Nuevas capas

Como podemos observar en la capa final de salida de 14 valores tenemos una función de activación sigmoide, cuyos valores de salida estarán siempre entre 0

y 1, esta función se aplica independientemente a cada uno de los elementos de salida de la red, lo cual quiere decir que la salida de la sigmoide para uno de esos valores no dependerá del resto de salidas de la red.

$$f(s_i) = \frac{1}{1 + e^{-s_i}}$$

También para la función de perdida hemos usado Binary Cross-Entropy Loss ya que tenemos un problema de clasificación multi-etiqueta. Para esto hay que entender Cross-Entropy loss que no es mas que una función para saber si el target de una clase dado el valor predicho de la red son parecidos o no, lo que hacemos es comparar vectores de probabilidades.

$$CE = -\sum_{i}^{C} t_i log(s_i)$$

Donde t_i y s_i son el valor real de salida (vector real) y el score (vector de probabilidad) predicho por la red para cada una de las clases i en C.

Esta función es independiente para cada componente vectorial (clase), lo que significa que la pérdida calculada para cada componente vectorial de salida de la CNN no se ve afectada por los valores de otros componentes. Por eso se utiliza para la clasificación de múltiples etiquetas, donde la percepción de un elemento perteneciente a una determinada clase no debe influir en la decisión para otra clase. En este caso tenemos C problemas binarios para cada clase.

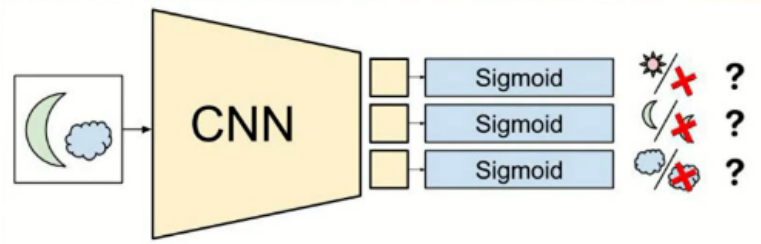


Figura 50: Binary Cross-Entropy Loss

Antes de entrenar vamos a agregar una nueva columna al DataFrame de datos, que serán los tags codificados, lo pasamos por un binarizador y obtenemos un

vector de 0 y 1 según la posición en donde se encuentre el tag, este sera el valor Y que predicará nuestro modelo. A este método se le conoce como one hot encoding, lo que hace one hot encoding es tomar una columna que tiene datos categóricos, que ha sido codificada con etiquetas, y luego dividir la columna en múltiples columnas. Los números se reemplazan por 1 y 0, dependiendo de qué columna tiene qué valor. Para nuestro DataFrame obtenemos el siguiente resultados.

	index	tag	filename	author	processed_image	tag_bina
0	1	(ancora, dofi)	0076522a.jpg	Manuzio, Paolo 1512-1574	0076522a1.jpg	[0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0]
1	2	(ancora, dofi)	0076522b.jpg	Manuzio, Paolo 1512-1574	0076522b2.jpg	[0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0]
2	3	(ancora, dofi)	0076522c.jpg	Manuzio, Paolo 1512-1574	0076522c3.jpg	[0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0]
3	4	(esfera,)	0009492a.jpg	Blaeu, Joan 1596-1673	0009492a4.jpg	[0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0]
4	5	(esfera,)	0009492b.jpg	Blaeu, Joan 1596-1673	0009492b5.jpg	[0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0]

Figura 51: Tags binarizados

Con nuestro DataFrame ya listo, lo único que queda en re-entrenar el modelo de MobileNet con los nuevos datos. Esto lo veremos mas adelante en el apartado de Pruebas y resultados, en el cual compararemos varias técnicas para el aumento de precisión del modelo final.

9. Desarrollo Web

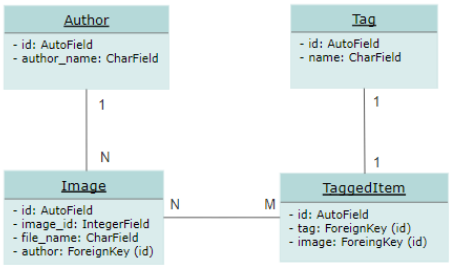
Continuando con el desarrollo del proyecto vamos a desarrollar un aplicativo web en el cual se podrá consultar los impresores y obtener las imágenes de dicho impresor, a su vez poder obtener las imágenes más similares a una imagen seleccionada por el usuario final y así poder valorar cuan bien obtenemos imágenes parecidas y poder enseñar al usuario una interfaz sencilla en la cual poder ver dichos resultados.. Por otra parte, veremos como se ha desarrollado cada parte y su posterior puesta en marcha en diferentes servicios.

9.1. Backend

Con el índice ya definido, lo que se ha realizado es la creación de una API con Django la cual tendrá el índice de Annoy guardado y la base de datos creada con todos los datos de imágenes, impresores y tags. Con lo cual lo único que tenemos que hacer es exportar el índice de Annoy, en este caso contiene todos los vectores

de las imágenes y también el método para devolver los mas parecidos dado un índice, este archivo pesa muy poco. Dicho archivo lo agregaremos a Django para poder realizar las consultas de las imágenes más parecidas.

Antes de el desarrollo de la API se analizo el esquema que tendrá nuestra base de datos, en este caso queremos tener impresores, los cuales pueden tener N imágenes con M tags. Estos tags pueden contener N imágenes de diferentes impresores. En este caso para los datos que tenemos la base de datos es bastante sencilla como podemos ver a continuación.



La creación de la API se ha realizado con el modulo REST framework que es bastante sencillo de usar y nos permite crear endpoints para poder realizar las consultas, no tenemos muchos endpoints mas que los de hacer consultas.

9.1.1. Endpoints

A continuación podemos ver los endpoints creados para nuestra aplicación.

GET /images

Obtener todas las imágenes paginadas

Responses

200 OK

Schema

```
object (5) optional
links object (2) optional
    next string optional
    previous string optional
total number optional
page number optional
page_size number optional
results array[object] (5) optional
    pk number optional
    image_id number optional
    file_name string optional
    author array[object] (2) optional
    tags array[object] (1) optional
```

GET /authors

Obtener todos los impresores paginados

Responses

200 OK

Schema

```
object (5) optional
links object (2) optional
    next string optional
    previous string optional
total number optional
page number optional
page_size number optional
results array[object] (3) optional
    pk number optional
    author_name string optional
    image array[object] (4) optional
        pk number optional
        image_id number optional
        file_name string optional
        tags array[object] (1) optional
            name string optional
```

POST /authors/{id}

Obtener un impresor dado un id

Path Parameters

`id` number
id del impresor a obtener

Responses

200 OK
404

Schema

```
object (3) optional
  pk number optional
  author_name string optional
  image array[object] (4) optional
    pk string optional
    image_id number optional
    file_name string optional
    tags array[object] (1) optional
```

POST /images/{id}

Obtener una imagen dado un id

Path Parameters

`id` number
id de la imagen a obtener

Responses

200 OK
404

Schema

```
object (5) optional
  pk number optional
  image_id number optional
  file_name string optional
  author object (2) optional
    pk number optional
    author_name string optional
  tags array[object] (1) optional
    string optional
```

POST /images/similar

Obtener imágenes similares

Request Body

object {2}	optional
id string or number id de la imagen	optional
num string or number número de imagenes a obtener	optional

Responses

200 OK

400

Schema

array[object] {5}	optional
pk number	optional
image_id number	optional
file_name string	optional
author object {2}	optional
pk number	optional
author_name string	optional
tags array[object] {1}	optional
name string	optional

POST /authors/search

Obtener imágenes similares

Request Body

object {1}	optional
name string nombre del impresor	optional

Responses

200 OK

400

Schema

object {5}	optional
links object {2}	optional
next string	optional
previous string	optional
total number	optional
page number	optional
page_size number	optional
results array[object] {3}	optional
pk number	optional
author_name string	optional
image array[object] {4}	optional
pk number	optional
image_id number	optional
file_name string	optional
tags array[object] {1}	optional
name string	optional

9.2. Frontend

Para la parte de frontend hemos trabajado con Angular para conectarnos a los endpoints que necesitamos nombrados anteriormente en la parte de backend. Dados estos endpoints de la API podemos obtener una lista paginada de impresores y de cada uno obtener las imágenes correspondientes a dicho impresor. También podemos seleccionar una imagen y obtener las K imágenes similares a dicha imagen. Es este caso tenemos una interfaz bastante sencilla pero a la vez bastante intuitiva.

9.2.1. Heurísticas de Nielsen

Se ha intento seguir algunas de las heurísticas de nielsen para poder centrarnos en la usabilidad de la interfaz de esta web. En esta web nos centramos en las siguiente:

- Visibilidad del estado del sistema: Podemos ver que impresor hemos seleccionado y que imagen.
- Consistencia y estándares: Tenemos animaciones para poder saber que imagen vamos a seleccionar o que impresor.
- Prevención de errores: Al tener pocos parámetros que cambiar los errores están solventados.
- Estética y diseño minimalista: Al tener una web con poca funcionalidad tenemos una diseño y estética minimalista.
- Flexbilidad y eficiencia de uso: Tenemos paginación y búsqueda de autores para facilitar al usuario la usabilidad.

9.2.2. Patrones de Diseño

Siguiendo con la usabilidad de la web, se ha optado por diferentes patrones de diseño, los cuales son:

- Cards: Para mostrar las imágenes.
- Local Search: Búsqueda de impresores.

- Numbered pagination: Paginación para los autores y la búsqueda.
- Local settings: Para cambiar el numero de imágenes obtenidas.
- Filter: Para filtrar los resultados de las imagenes obtenidas.
- Breadcrumbs: Para mostrar al usuario donde se encuentra.

9.3. Deploy

Para realizar el deploy de los aplicativos se han utilizado dos servicios bastante conocidos y gratuitos para nuestro proyecto.

9.3.1. Heroku

Para realizar el deploy del proyecto de Django se ha optado por usar Heroku que es una plataforma como servicio de computación en la Nube que soporta distintos lenguajes de programación, entre ellos Python. En esta caso al ser gratuito podemos tener un servicio activo a internet de manera rápida sin gasto alguno.

9.3.2. Firebase

Para realizar el deploy del proyecto de Angular se ha optado por usar Firebase que es un servicio que ofrece Google, es una plataforma para el desarrollo de aplicaciones web y móviles. Firebase proporciona funciones como estadísticas, bases de datos, informes de fallas y mensajería, de manera que puedas ser más eficiente y permite a los desarrolladores enfocarse en los usuarios. En este caso es un servicio gratuito con lo cual el despliegue es bastante sencillo para una aplicación basada en Javascript.

10. Pruebas y resultados

En este apartado vamos a valorar las pruebas que se han realizado para todo lo que se ha visto de Fine-tuning. Por otra parte podremos ver el resultado obtenido aplicando diferentes técnicas para aumentar la precisión de los resultados.

10.1. Fine-tuning

Una vez hemos obtenido el modelo "ganador"se ha re-entrenado con los nuevos datos, en este caso, las imágenes con tags relevantes y que se parecen entre si. Para este apartado no se ha tocado la red, solo se han agregado las nuevas capas, y como podemos observar hay un sobreajuste temprano bastante grande en las primeras etapas del entrenamiento y la precisión de la validación es bastante baja.

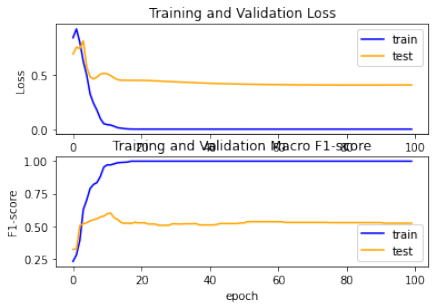


Figura 52: Sin generador ni tuning de capas

10.2. Data Augmentation

En este caso se ha agregado un generador de datos, lo que se conoce como data augmentation, que es básicamente agregar datos nuevos modificando las imágenes de entrenamiento y aplicando rotaciones, ampliaciones, etc. Podemos obtener mejores resultados, esto es debido que aumentando el número de datos retrasamos el sobreajuste y reducimos el loss, que es lo que estamos buscando.

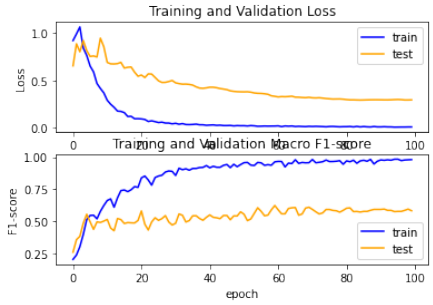


Figura 53: Con generador sin tuning de capas

10.3. Layer tuning

En este caso se ha aplicado un tuning de las capas para que se modifiquen los pesos de las capas convolucionales y así puedan identificar mejor los objetos de nuestro dataset, podemos obtener aun mejores resultados como se muestra a continuación.

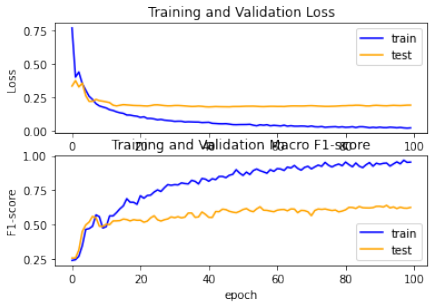


Figura 54: Con generador y tuning de capas

Este es el mejor modelo para extraer características y para clasificar las imágenes dadas, lo que se realizara a continuación es comprobar que tal son los resultados de este nuevo modelo y si obtenemos mejores resultados o no. Observado en general los resultados podemos decir que el generador de datos y el tuning de capas ayuda muchísimo al re-entrenamiento de una red.

	Loss	F1-score
No gen No tuning	0.475	0.515
Si gen No tuning	0.295	0.584
Si gen Si tuning	0.193	0.626

10.4. Resultados

En esta apartado vamos a probar que tal se comporta este modelo, tanto para los tags con una respuesta alta y también para todos los tags analizamos en una primera instancia, así podremos comprobar si hemos obtenido mejores resultados o no.

En este caso obtenemos los siguientes resultados, como podemos ver la mejoría es del 4 % tanto para todas la imágenes como para los tags de alta respuesta. No es un gran aumento en cuanto a la precisión pero definitivamente podemos usar este modelo para extraer las características y obtener imágenes similares.

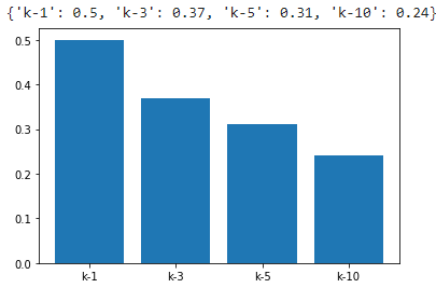


Figura 55: Todas las imágenes

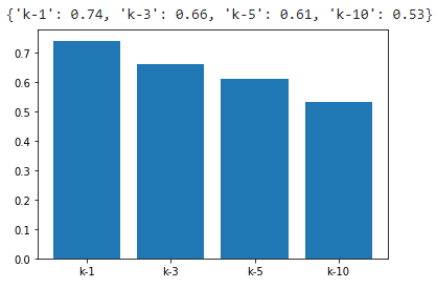


Figura 56: Imágenes con tags de alta respuesta

11. Conclusiones

Este proyecto me ha servido como estudiante a explorar muchísimas más tecnologías de las que se han visto en la carrera y poder aprender alguna tecnología nueva. Se ha logrado tener una web en fase "beta"para poder consultar la base de datos de impresores y obtener imágenes similares de una imagen.

He podido aprender bastante sobre el redes convolucionales y a entender mejor el mundo de tecnologías que hay detrás y saber como usarlas, en este caso con

Keras que nos agrega una capa superior a la que tenemos por defecto con TenserFlow podemos obtener una vista mas general de las redes convolucionales como desarrolladores y no como arquitectos de redes. En este caso siguiendo la documentación podemos entender el funcionamiento de todos los métodos utilizados sin indagar mucho en como funcionan por dentro o en conceptos matemáticos. Con lo cual para desarrolladores Keras es una buena opción para empezar en el mundo del deep learning y jugar con muchos modelos de los cuales están disponibles de manera gratuita.

En el aspecto de la investigación también he podido aprender a ver como funcionan las redes convulucionales mas en profundidad de los que se ve en la carrera. Como he podido ver tenemos una infinidad de métodos diferentes y distintos tipos de redes, a su vez tenemos muchos nuevos métodos que podemos usar para resolver un problema en concreto que tengamos. En el caso del Fine-tuning podemos abordar distintos problemas de clasificación de imágenes y extracción de características con dichos modelos, y como hemos podido ver llevarlos a nuestro problema para encontrar una solución viable.

En la parte de desarrollo, me ha gustado la idea de dividir el proyecto de dos subproyectos diferentes y asi poder tener una aplicación escalable en tecnologías diferentes auto-gestionadas independientemente. Para la parte de código, ambos proyecto se han subido a un repositorio privado de Github, divididos para facilitar el deploy de las aplicaciones a sus distintos servicios. En este aspecto me ha ayudado a poder tener control de dichas aplicaciones por separado.

En el frontend he aprendido Angular, he podido familiarizarme con sus componentes y poder gestionar una API de manera básica, se ha valorado también la organización del código del proyecto de Angular, siguiendo patrones visto en diferentes repositorios de expertos que trabajan con dicha tecnología. También centrarme en el diseño de la web y en la usabilidad de la misma, en este caso siguiendo lo visto en Factores Humanos.

Para la parte del backend con Django no ha resultado muy difícil al empezar a preparar los endpoints ya que estaba familiarizado con los visto en la carrera sobre este framework.

11.1. Problemas

Una de los problemas que surgieron al inicio fue la de los datos, ya que no se tenia acceso directo a los datos o las imágenes. Con lo cual tener la manera de poder obtener estos datos fue una complicación de inicio, por se pude obtener los datos en un formato que se pudo lograr a entender para poder a partir de estos obtener las imágenes.

La parte complicada del backend ha sido el incorporar Annoy, ya que la instalación en Heroku en un inicio dio bastante problemas, pero se pudieron solucionar. Con esto ultimo también tenemos una limitación en cuanto a memoria, sabemos que Annoy usa ficheros estáticos para sus índices para ahorrar memoria, pero aun así usa parte de ella para otras cálculos, esto seria un problema para un entorno real con millones de imágenes en un servidor básico de Heroku.

Por motivos personales con lo vivido estos últimos meses con el covid, el proyecto se retraso un poco y no pude avanzar todo lo que quería ni hacer más pruebas en la parte de investigación, al final pude terminar toda esta parte y mejorar el apartado de la web que quedaba por terminar. Quizá queden cosas por avanzar en el apartado web que seguramente me hubieran gustando terminar y mejorar para una experiencia de usuario mas completa. El proyecto empezó bien en este aspecto pero en la curva final se alargo demasiado. Aquí se puede ver la planificación final después del covid.

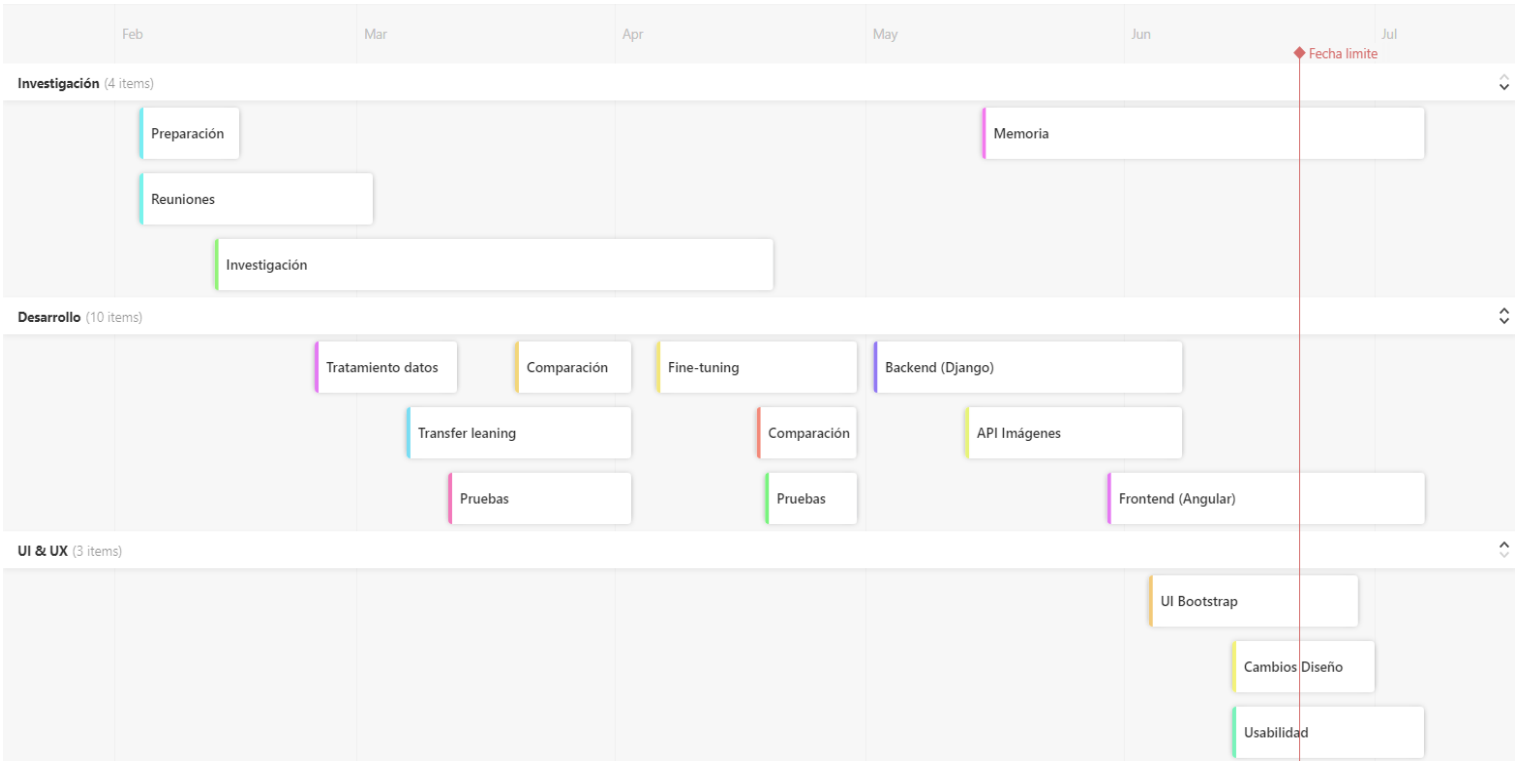


Figura 57: Planificación final post covid

11.2. Trabajo futuro

Como agregación se quieren definir lineas de trabajo que se podrían seguir para el futuro de la aplicación, en este caso para el apartado web. Quizá también, en el aspecto de errores de imágenes etiquetadas y al aumento de datos de las mismas, esto vendría dado por el acceso directo a la base de datos y quizá la creación de una base alternativa.

- Características de la aplicación: Agregar alguna característica nueva a la aplicación web como la subida de imágenes y el tratamiento de tags seguido también del re-entrenamiento por parte del servidor con las nuevas imágenes y la creación de un nuevo índice de Annoy. En este aspecto, al no tener una conexión directa se podría tener otra base de datos solo para imágenes y quizá un sistema de migración de datos a otro servidor con mayor capacidad para la actualización de los datos de las imágenes y la

extracción de características. En este aspecto la parte mas costosa seria la de la extracción de características y la actualización de los índices de Annoy, ya que se tendría que volver a crear de nuevo todo el índice, Annoy no permite agregar nuevos datos una ves tengamos el modelo construido.

- Errores: Se podría mejorara los resultados obtenidos agregando nuevas imágenes y arreglando los tags de dichas imágenes para que tuvieran imágenes parecidas al resto de imágenes. En este aspecto poco más se puede hacer al tener un dataset de este tipo, ya que es un trabajo de etiquetado. Si por el contrario tuviéramos imágenes muy parecidas con sus etiquetas y quizá un gran numero de imágenes, posiblemente los resultados del re-entrenamiento serian mucho mejores a los vistos en este proyecto.
- Diseño de la aplicación: Quizá se podría mejorar el diseño de a aplicación con algún otro framework de CSS o con algún tema definido, así también a su vez tener otra gama de colores diferente al actual.

Se podría seguir hablando de mejoras a futuro pero probablemente no terminaríamos nunca. Solo nos queda seguir con estos puntos y trazar una linea de desarrollo para poder seguir mejorando la aplicación, pero la parte fundamental la tenemos lista para continuar por este camino.

12. Bibliografía

[1] pandas - Python Data Analysis Library
`https://pandas.pydata.org/`

[2] Approximate nearest neighbors and vector models, introduction to Annoy
`https://www.youtube.com/watch?v=QkCCyLW0ehU`

[3] spotify/annoy: Approximate Nearest Neighbors in C++/Python optimized for memory usage and loading/saving to disk
`https://github.com/spotify/annoy`

[4] Keras: the Python deep learning API
`https://keras.io/`

[5] OpenCV: OpenCV modules
https://docs.opencv.org/master/index.html

[6] Django documentation — Django documentation — Django
https://docs.djangoproject.com/en/3.1/

[7] Angular - Introduction to the Angular Docs
https://angular.io/docs

[8] Sign language image classification - Fine-tuning MobileNet with Keras - deeplizard
https://deeplizard.com/learn/video/FNqp4ZY0wDY

[9] How to Visualize Filters and Feature Maps in Convolutional Neural Networks
https://machinelearningmastery.com/

[10] Building a Reverse Image Search Engine: Understanding Embeddings
https://www.oreilly.com/

[11] Similar Images Recommendations using FastAi and Annoy,
https://towardsdatascience.com/

[12] Breaking Down Mean Average Precision (mAP)
https://towardsdatascience.com/

[13] Multi-label Remote Sensing Image Retrieval based on Deep Features, Michele Compri, https://imatge.upc.edu/web/publications/multi-label-remote-sensing-image-retrieval-based-deep-features

[14] Image visual similarity with deep learning: application to a fashion e-commerce company, Pedro Rui, Silva Da, Machado R (2017)

[15] Find more like this product using Transfer Learning, https://mc.ai/find-more-like-this-product-using-transfer-learning/

[16] Product Recommendation by Image Similarity,
http://www.luccasperone.com/2018/10/04/product-recommendation-by-image-similarity/

[17] How Transfer Learning Really Works,
https://blog.paperspace.com/art-with-neural-networks/

[18] Analysis of Different Similarity Measures in Image Retrieval Based on Texture and Shape, https://www.researchgate.net/publication/324039262_Analysis_of_Different_Similarity_Measures_in_Image_Retrieval_Based_on_Texture_and_Shape

[19] Stoplight, Design, Document Build Quality APIs Faster,
https://stoplight.io/

[20] Shahla J. Hassen, Ahmed Taha and Mazen M. Selim, 2019. Trademark Image Retrieval using Transfer Learning. Journal of Engineering and Applied Sciences, 14: 6897-6905,
https://medwelljournals.com/abstract/?doi=jeasci.2019.6897.6905

[21] Specialist training and resources for Angular, https://codecraft.tv/

[22] Fine-tuning with Keras and Deep Learning,
https://www.pyimagesearch.com/

[23] Fine-tuning using pre-trained models, https://www.learnopencv.com/

[24] Masonry, https://masonry.desandro.com/

[25] BBC GEL — Foundations,
https://www.bbc.co.uk/gel/guidelines/category/foundations

[26] Django REST framework, https://www.django-rest-framework.org/

[27] Django Tagging,
https://django-tagging.readthedocs.io/en/develop/

[28] router.navigateByUrl and router.navigate, https://www.it-swarm.dev/

[29] Specialist training and resources for Angular, https://codecraft.tv/

13. Anexos

13.1. Web demo

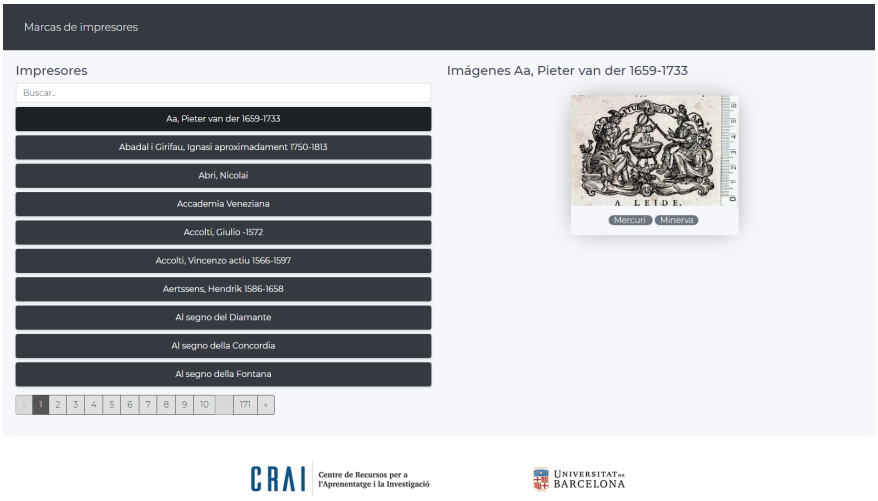


Figura 58: Inicio

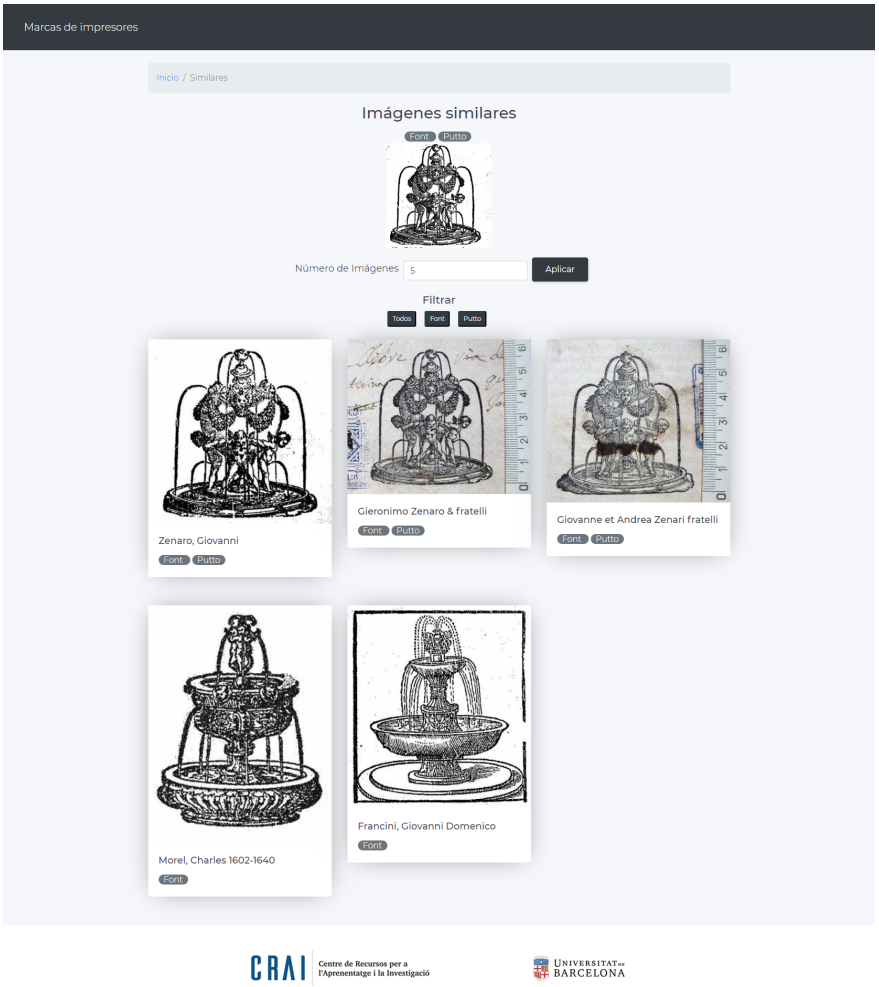


Figura 59: Imagen similar

Figura 60: Búsqueda