



Centro Universitario de la Defensa en la Escuela Naval Militar

TRABAJO FIN DE GRADO

*Localización de imágenes subidas por usuarios de redes sociales
mediante utilización de herramientas de búsqueda visual de
imágenes*

Grado en Ingeniería Mecánica

ALUMNO: Pablo Duque Arce

DIRECTORES: Milagros Fernández Gavilanes

CURSO ACADÉMICO: 2021-2022

UniversidadeVigo



Centro Universitario de la Defensa en la Escuela Naval Militar

TRABAJO FIN DE GRADO

*Localización de imágenes subidas por usuarios de redes sociales
mediante utilización de herramientas de búsqueda visual de
imágenes*

Grado en Ingeniería Mecánica
Intensificación en Tecnología Naval
Cuerpo General

UniversidadeVigo

RESUMEN

Hoy en día, las redes sociales constituyen un pilar básico de la sociedad. Los usuarios publican imágenes de manera habitual, pero son muy pocos los que indican la localización de las mismas.

Conocer esa ubicación tiene un gran interés en muchos ámbitos, como por ejemplo, el social, el político, el económico, pero también el de seguridad.

Durante el desarrollo de este Trabajo de Fin de Grado, se explicarán las aplicaciones y buscadores que realizan búsqueda inversa de imágenes que están disponibles en el mercado actual y sobre las que nos apoyaremos para la obtención de la localización. Posteriormente, explicaremos las diferentes herramientas y librerías que utilizaremos para la confección de nuestros algoritmos.

Una vez definidas estas herramientas, introduciremos las tres arquitecturas en las que se basa GeoApi, nombre del sistema creado. En la primera arquitectura, tendremos tres algoritmos de búsqueda distintos que presentarán las diferentes localizaciones en función a diversos criterios. En la segunda, se ha desarrollado un cuarto algoritmo que presenta las posibles localizaciones en las que se ha ubicado el usuario durante un espacio temporal. En la tercera parte, introduciremos unas mejoras en el tratamiento de imágenes de forma a detectar, localizar y eliminar objetos e individuos.

Por último, someteremos a pruebas a nuestro sistema y presentaremos los resultados. Para las pruebas, utilizaremos imágenes extraídas de publicaciones realizadas por usuarios en la red social Twitter.

PALABRAS CLAVE

Geolocalización, Python, Geocodificación, API Google Vision, GeoApi.

ÍNDICE

Índice	1
Índice de Figuras	3
Índice de Tablas.....	6
1 Introducción y objetivos	7
1.1 Motivación	7
1.2 Objetivos	8
1.3 Organización de la memoria	9
2 Estado del arte	10
2.1 Redes sociales	10
2.1.1 Conceptos fundamentales de Twitter.....	11
2.1.2 Funcionalidades relacionadas con la geolocalización	12
2.1.3 El problema de la geolocalización	13
2.2 Métodos para determinar la ubicación de una imagen.....	13
2.3 Análisis de herramientas	13
2.3.1 Aplicaciones	14
2.3.2 Buscadores	19
2.3.3 Comparativa.....	21
3 Desarrollo del TFG.....	22
3.1 Herramientas necesarias.....	23
3.1.1 El lenguaje de programación Python.....	23
3.1.2 Entorno de desarrollo (IDLE).....	23
3.1.3 La API Google Vision	24
3.1.4 El formato JSON.....	27
3.1.5 La librería de extracción de mensaje Snsrape	28
3.1.6 La base de datos MongoDB.....	29
3.1.7 La librería metadata_parser	29
3.1.8 La librería GeoPy.....	30
3.2 Arquitectura general de GeoApi	30
3.2.1 Arquitectura general nº1	31
3.2.2 Arquitectura general nº2	40
3.2.3 Arquitectura general nº3	50
4 Resultados	55
4.1 Datos de prueba.....	55

4.2 Pruebas realizadas	57
4.2.1 Resultados obtenidos sobre imágenes con geolocalización.....	57
4.2.2 Resultados obtenidos sobre imágenes sin geolocalización.....	59
4.2.3 Comparativa entre resultados de conjuntos de datos con y sin geolocalización	61
4.2.4 Resultados obtenidos sobre imágenes de forma conjunta	62
4.2.5 Comparativa de ciudades	64
4.2.6 Detección y eliminación de objetos	66
4.2.7 Detección y eliminación de caras	68
5 Conclusiones y líneas futuras	71
5.1 Conclusiones	71
5.2 Líneas futuras	72
6 Bibliografía.....	73
Anexo I: Algoritmo 1	77
Anexo II: Algoritmo 2.....	80
Anexo III: Algoritmo 3.....	84
Anexo IV: Algoritmo Detección Usuario	88
Anexo V: Api Google Vision	100

ÍNDICE DE FIGURAS

Figura 1 - PitPicks (tomada de [12])	14
Figura 2 - Camfind (tomada de [6]).....	15
Figura 3 - Picturethis (tomada de [8])	15
Figura 4 - PlantSnap (tomada de [11])	16
Figura 5 - Search by Image (tomada de [19]).....	16
Figura 6 -PhotoSherlock (tomada de [21]).....	17
Figura 7 - Google Lens (tomada de [23]).....	18
Figura 8 - Creepy (tomada de [25])	19
Figura 9 - WeSee (tomada de [27])	20
Figura 10 - Yandex (tomada de [30])	20
Figura 11 - Google (tomada de [33])	21
Figura 12 - Python (tomada de [35])	23
Figura 13 - IDLE (tomada de [36])	24
Figura 14 - Google Vision (tomada de [39])	24
Figura 15 - Detección de productos (tomada de [40]).....	25
Figura 16 - Transcripción y clasificación de imágenes (tomada de [40])	26
Figura 17 - Búsqueda de imágenes similares (tomada de [40])	26
Figura 18 - Formato JSON (Edición propia)	28
Figura 19 - Librería Snsrape (tomada de [43])	28
Figura 20 - MongoDB (tomada de [45])	29
Figura 21 - Librería metadata_parser (tomada de [47])	29
Figura 22 - Librería GeoPy (tomada de [49]).....	30
Figura 23 - Arquitectura general 1 (Edición propia)	31
Figura 24 - Interfaz con el usuario (Edición propia)	31
Figura 25 - Detección de paisajes (Edición propia)	32
Figura 26 - Detección web (Edición propia)	33
Figura 27 - Metadatos (Edición propia)	33
Figura 28 - Almacenar información JSON (Edición propia)	34
Figura 29 - Guardar archivo JSON (Edición propia)	34
Figura 30 - Nominatim (Edición propia).....	34
Figura 31 - Geocodificación (Edición propia).....	34
Figura 32 - Extraer información de JSON (Edición propia)	35
Figura 33 - Selección coordenadas con mayor score (Edición propia)	35
Figura 34 - Representación de coordenadas (Edición propia).....	36
Figura 35 - Ciclo for (Edición propia).....	36

Figura 36 - Recuento palabras (Edición propia).....	37
Figura 37 - Palabra más repetida (Edición propia).....	37
Figura 38 - Palabra más repetida (Edición propia).....	37
Figura 39 - Lectura Formato JSON (Edición propia).....	38
Figura 40 - Búsqueda latitudes repetidas (Edición propia)	39
Figura 41 - Representación localización repetida máxima score (Edición propia).....	39
Figura 42 - No ubicaciones coincidentes (Edición propia)	40
Figura 43 - HTML / representación de coordenadas (Edición propia)	40
Figura 44 - Arquitectura general 2 (Edición propia).....	41
Figura 45 - Snsrape (Edición propia).....	41
Figura 46 - Almacenamiento información MongoDB (Edición propia)	42
Figura 47 - Base de Datos MongoDB	42
Figura 48 - Funciones os.listdir / isfile (Edición Propia)	43
Figura 49 - Función algoritmo1 (Edición propia)	43
Figura 50 - Función algoritmo2 (Edición propia)	44
Figura 51 - Función algoritmo2 / Recuento de palabras (Edición propia).....	44
Figura 52 - Función algoritmo 2 / Selección palabras (Edición propia)	44
Figura 53 - Función algoritmo2 / Devolución Lista2 (Edición propia)	45
Figura 54 - Función algoritmo3 / Almacenamiento Información (Edición propia)).....	45
Figura 55 - Función algoritmo3 / Búsqueda Localizaciones Coincidentes (Edición propia	46
Figura 56 - Función algoritmo3 / Devolución lista (Edición propia).....	46
Figura 57 - Lectura JSON / Llamada funciones (Edición propia).....	47
Figura 58 - Almacenamiento de información repetida (Edición propia).....	48
Figura 59 - Existe solo una Ubicación (Edición propia)	48
Figura 60 - Asignación variables elementos repetidos (Edición propia)	48
Figura 61 - Probabilidades para tres o más elementos repetidos (Edición propia)	49
Figura 62 - Representación no localizaciones coincidentes (Edición propia).....	49
Figura 63 - Representación 2 variables coincidentes (Edición propia).....	50
Figura 64 - Representación tres elementos coincidentes (Edición propia)	50
Figura 65 - Mejoras API Google Vision (Edición propia)	51
Figura 66 - Extraer Información API Google (Edición propia)	51
Figura 67 - Almacenamiento Información librería 'pandas' (Edición propia).....	51
Figura 68 - Ciclo for / Funciones (Edición propia)	52
Figura 69- Función draw_borders (Edición propia).....	52
Figura 70 - Función remove_borders (Edición propia).....	53
Figura 71 - Detección Caras (Edición propia).....	53

Figura 72 - faceAnnotation / Abrir Imagen / Ciclo for (Edición propia)	54
Figura 73 - Función remove_faces (Edición propia).....	54
Figura 74 - Mostar / Guardar la imagen (Edición propia).....	54
Figura 75 - Consulta imágenes con geo/sin geo (Edición propia).....	55
Figura 76 - Criterios de búsqueda imágenes Detección de usuario (Edición propia)	56
Figura 77 - Objetos No son Eliminados (Edición propia).....	67
Figura 78 - Antes de detección y eliminación de objetos (tomada de [50]).....	68
Figura 79 - Después de detección y eliminación de objetos (Edición propia)	68
Figura 80 - Antes de detección y eliminación de caras (tomada de [51])	69
Figura 81 - Después de detección y eliminación de caras (Edición propia).....	70

ÍNDICE DE TABLAS

Tabla 1 - N° imágenes válidas por ciudad con/sin geo (Edición propia).....	56
Tabla 2 - N° Usuarios e Imágenes por ciudad (Edición propia)	57
Tabla 3 - Resultados de Imágenes con Geolocalización (Edición propia)	58
Tabla 4 - Resultado Final Imágenes con Geolocalización (Edición propia)	59
Tabla 5 - Resultado de imágenes sin Geolocalización (Edición propia)	60
Tabla 6 - Resultado Final de Imágenes sin Geolocalización (Edición propia)	61
Tabla 7 - Resultados Detección de Usuario (Edición propia)	63
Tabla 8 - Resultado Final Detección Usuarios (Edición propia).....	63
Tabla 9 - Análisis Ciudades con Geolocalización (Edición propia).....	64
Tabla 10 - Análisis Ciudades sin Geolocalización (Edición propia).....	65
Tabla 11 - Análisis ciudades conjunto de imágenes (Edición propia)	66
Tabla 12 - Análisis Detección Eliminación Objetos (Edición propia)	67
Tabla 13 - Detección Eliminación Caras (Edición propia).....	69

1 INTRODUCCIÓN Y OBJETIVOS

1.1 Motivación

Las redes sociales, dentro del ámbito del mundo virtual, son aplicaciones que operan en múltiples ámbitos diferentes de la sociedad como puede ser: el profesional, el social, el laboral... y permiten el intercambio de información entre personas y/u organizaciones.

Al hablar de una red social, lo primero que nos viene a la mente son redes como *Instagram*¹, *Twitter*² o *Facebook*³, que son las más comunes en la actualidad. Sin embargo, las redes sociales tienen un concepto mucho más antiguo. Una red social es un concepto que analiza la interacción que hay entre unos individuos y este término de “red social” existe desde el siglo XIX [1].

Dentro de la red Internet, las redes sociales han tenido una gran controversia en algunos temas como puede ser la falta de privacidad, pero también han tenido un importante impacto positivo en otros temas, actuando como elemento de convocatoria de manifestaciones públicas. Estas plataformas, también han creado una nueva forma de entender las relaciones comerciales, abriendo nuevos caminos para el anuncio de productos o servicios.

En este sentido, algunas redes sociales, como bien pueden ser Instagram o Twitter, han visto cómo sus usuarios los usaban en gran medida para mostrar su estilo de vida. Por ejemplo, en Instagram la mayor parte de su contenido son fotografías o vídeos, pese a contener más aplicaciones. Twitter, también cuenta con una gran cantidad de usuarios que publican imágenes acerca de su día a día.

Al realizar fotografías creamos una imagen que inmortaliza un momento que hemos vivido. Pero, en la sociedad moderna tomar una foto no es suficiente. Ahora existe la necesidad de compartirla para validar esta experiencia que estamos viviendo, es decir, si no lo has compartido en tus redes sociales no lo has hecho.

Esta obsesión con compartir nuestras actividades puede tener distintos objetivos, como pueden ser: presumir acerca de un determinado estilo de vida, aumentar la autoestima con ‘me gustas’ o reacciones a nuestras publicaciones, proyectar una personalidad idónea, compartir momentos con otras personas, obtener aprobación social o llamar la atención de otras personas. Estos son algunos de los factores por los que los usuarios deciden mostrar su vida en las redes.

Debido a este aumento del uso de imágenes en las redes sociales, en muchas ocasiones es de gran necesidad poder conocer la localización de una persona a través de una determinada imagen. Esto

¹ <https://instagram.com/> [Disponible a 14/03/2022]

² <https://twitter.com/> [Disponible a 23/01/2022]

³ <https://facebook.com/> [Disponible a 14/03/2022]

puede tener una gran cantidad de aplicaciones como bien pueden ser seguridad, su uso por parte de otras aplicaciones para dispositivos móviles que tengan algún tipo de funcionalidad como, por ejemplo, detección e identificación de objetos o detección de localizaciones.

Debido a lo anterior, durante la última década el desarrollo de algoritmos de localización se ha ido incrementando exponencialmente. Se ha incorporado en los últimos tiempos del uso en tiempo real del escaneo de códigos QR en las imágenes, las cuales tienen localizaciones asociadas, identificando animales e incluso plantas, pero también permite clasificar tipos de comida o permite consultar valoraciones de restaurantes o tiendas. Estas son algunas de las muchas utilidades que presentan este tipo de aplicación.

Sin irnos más lejos, un claro ejemplo se puede ver con lo sucedido el pasado año en Barcelona, donde la policía nacional distribuyó en sus redes sociales la fotografía de la habitación de una menor de edad que posiblemente estaba siendo víctima de abusos sexuales, con el objetivo de localizar a la víctima. Todo lo anterior, dentro de una investigación abierta por la unidad de ciberdelincuencia sobre pornografía infantil.

En la imagen se podía ver una habitación con una cama; sobre la que se encontraba una considerable cantidad de peluches infantiles, y diferentes dibujos pegados en las paredes. Poco tiempo después, gracias a los correos y mensajes de diferentes usuarios se consiguió identificar en un tiempo record la habitación.

Consecuentemente la policía declaró en sus redes sociales: *“Vuestros correos electrónicos han permitido identificar la habitación (y en tiempo récord). MUCHÍSIMAS GRACIAS. Nuestros agentes ya se han puesto en contacto con los familiares de la menor y continúan las investigaciones.”* [2].

Esta no ha sido la primera vez que los Cuerpos de Seguridad del Estado han recurrido a la ayuda ciudadana para llegar a importantes conclusiones en diferentes investigaciones. Otro ejemplo, se dio en Alemania donde la policía consiguió detener a un violador tras difundir fotografías de su víctima. La policía accedió a las imágenes tras descubrirlas en la Deep Web [3].

Con estos dos ejemplos, podemos poner de manifiesto como la importancia de este tipo de aplicaciones ha ido aumentando exponencialmente hasta volverse imprescindible para una gran multitud de actividades.

Este tipo de aplicaciones permiten la detección de diferentes elementos como pueden ser objetos, edificios, estatuas o monumentos. A partir de dichos elementos las aplicaciones son capaces de extraer la localización de las imágenes.

Cada vez son más comunes las aplicaciones que desarrollan labores de búsqueda inversa de imágenes, pero, aún queda un largo proceso para conseguir que estas plataformas permitan obtener cualquier localización a partir de una imagen. Este proyecto constituye una ayuda más para seguir desarrollando este tipo de tecnologías.

1.2 Objetivos

Los objetivos que se pretenden alcanzar en el presente TFG se exponen a continuación:

- Familiarización con aplicaciones y buscadores que realizan búsqueda inversa de imágenes.
- Utilización de la API Google Vision para extraer información a partir de una serie de imágenes.
- Investigación, aprendizaje y uso de herramientas de Python para obtener imágenes de Twitter.
- Almacenamiento de las imágenes e información anexa a ellas una base de datos.

- Uso de Formatos JSON para almacenar la información extraída de las imágenes. Posteriormente, se deberán analizar dichos archivos JSON para la obtención de localizaciones.
- Desarrollo de varios sistemas de algoritmos que presentarán una localización a partir de la información proporcionada por la API Google Vision. Cada sistema deberá trabajar en base a unos criterios que lo hagan diferente al resto.
- Análisis de los resultados obtenidos. Para estos análisis utilizaremos las imágenes que habremos descargado previamente de Twitter.
- Sintetización de los resultados obtenidos y extracción de conclusiones.

1.3 Organización de la memoria

La memoria de este Trabajo de Fin de Grado se encuentra dividida en cinco capítulos, a los que se le añade un anexo.

En el primer capítulo se describe la introducción, objetivos y la organización de la memoria, que se desarrolla en este apartado. En el segundo capítulo se relata el estado del arte actual, donde comenzaremos con una introducción acerca de las redes sociales, para luego focalizar en la red social Twitter que usaremos para nuestro trabajo. Proseguiremos explicando métodos para determinar la ubicación de una imagen. A continuación, se introducirán diversas aplicaciones que hacen uso de imágenes y que permiten extraer información a partir de ellas, como por ejemplo su localización. Ejemplo de ello será la herramienta Creepy.

En el Capítulo 3 se realizará el desarrollo del TFG, en el que comenzaremos introduciendo el lenguaje de programación Python. Proseguiremos explicando la API de Google Vision. Una vez hayamos realizado la introducción a dicha API, explicaremos que es un archivo JSON, su funcionamiento y aplicación en Python. Posteriormente, explicaremos el concepto general que fundamenta este proyecto denominado GeoApi, cómo funciona y las herramientas que se usarán. Continuando, en el siguiente apartado realizaremos un esquema general, donde podremos ver las diferentes partes de las que consta GeoApi e iremos explicando una a una las diferentes piezas y su funcionamiento. Por último, explicaremos unas mejoras implementadas para el tratamiento de imágenes.

En el capítulo 4 analizaremos los resultados obtenidos por GeoApi en función del tipo de imagen y del algoritmo utilizado. Empezaremos explicando el modo de obtener imágenes partiendo de publicaciones en Twitter sin y con geolocalización. Una vez realizado el análisis, sacaremos conclusiones en base a los resultados obtenidos, comparando la relación que existe entre el tipo de imagen, la ciudad y la localización que nos ha proporcionado el algoritmo utilizado. Finalizando el análisis, comprobaremos el funcionamiento de las mejoras implementadas.

En el capítulo 5 extraeremos las diferentes conclusiones que hemos extraído con la realización del proyecto y plantearemos las líneas futuras para continuar trabajando y mejorando los algoritmos de detección de localizaciones a partir de imágenes.

2 ESTADO DEL ARTE

2.1 Redes sociales

Una red social es un servicio que permite a un individuo construir un perfil público o semipúblico dentro de un entorno delimitado, confeccionando una lista de “amigos” con los que comparte contenido y conexión. La naturaleza y el modo de nombrar a estas conexiones pueden variar de un sitio a otro. De este modo, las redes sociales proporcionan interacción a miles de personas en prácticamente cualquier ubicación y en cualquier momento, y estas se pueden clasificar en cuanto a diversos factores que a su vez no son excluyentes entre sí. Estos son:

- Por su público y temática
 - Redes sociales horizontales: para todo tipo de usuario y sin temática definida. Ejemplos: Facebook, Twitter...
 - Redes sociales verticales: en base a un eje temático y su objetivo es congrega gente alrededor de ese eje. Ejemplo: *Meetic*⁴.
- Según como sea la interacción entre los usuarios:
 - Redes sociales humanas: fomentan las relaciones entre personas en función de sus gustos, aficiones u actividades. Ejemplos: *Tuenti*⁵, *Youare*⁶...
 - Redes sociales de contenido: unen perfiles a través de contenido publicado. Ejemplos: *TikTok*⁷, reels de Instagram...
 - Redes sociales inertes: unir marcas, automóviles y lugares. Ejemplo: *Respectance*⁸.
- Por su localización geográfica
 - Redes sociales sedentarias: este tipo de red varía en función de las relaciones entre personas, contenidos compartidos o eventos creados.
 - Redes sociales nómadas: además de las características previas de las redes sociales sedentarias habría que sumarle la localización del sujeto.

En nuestro caso nos interesan las redes sociales horizontales, porque son las más usadas a nivel social y es donde los usuarios comparten más fotografías. Además, las redes sociales de contenido; son las que agrupan a las personas en función de sus preferencias y las imágenes son uno de los recursos más usados para compartir y, en último lugar las redes sociales nómadas que utilizan la localización de los usuarios para prestarles determinados servicios [4].

⁴ <https://www.meetic.es/> [Disponible a 14/03/2022]

⁵ <https://tuenti.es/> [Disponible a 14/03/2022]

⁶ <https://www.youare.com/> [Disponible a 14/03/2022]

⁷ <https://www.tiktok.com/es/> [Disponible a 14/03/2022]

⁸ <https://es.respectance.com/> [Disponible a 14/03/2022]

El desarrollo de los dispositivos móviles y de los servicios de telefonía ha aumentado de forma exponencial durante esta década. Consecuentemente, la utilización de las redes sociales en estos dispositivos también ha visto un gran auge. Hoy en día, es extraño que un ciudadano adulto no sea usuario de alguna red social. Además, la edad con la que las personas acceden a estas redes cada vez es menor debido en parte a la facilidad de acceso de los terminales móviles y a la falta de control en este determinado ámbito.

Muchas de estas redes sociales son redes horizontales en las que los usuarios comparten todo tipo de imágenes: vacaciones, mascotas, paisajes, vehículos, deportes, etc. Pero muchas de estas imágenes podrían ser engañosas o simplemente artificiales, debido a la necesidad social que ha venido desarrollándose donde si no compartes las actividades que realizas es como si no las hubieras realizado.

El aumento en el número de usuarios de estas redes ha acarreado la aparición de nuevas actividades delictivas relacionadas con ciertos fraudes que tratan de engañar a los usuarios más despistados. Por ello es necesario la creación de ciertas aplicaciones o funcionalidades que traten de combatir este tipo de actividades.

Para detectar este tipo de actividades delictivas o engañosas que utilizan fotografías se han ido desarrollando diferentes softwares durante los últimos años, que permiten conocer la localización aproximada de donde fue tomada la imagen en solo unos segundos. Este tipo de aplicaciones tienen una gran cantidad de funcionalidades como pueden ser: clasificación de objetos, lectura de texto o detección de logos. Adicionalmente, se están desarrollando aplicaciones que permiten la identificación de personas a partir de imágenes.

2.1.1 Conceptos fundamentales de Twitter

Twitter⁹ fue creado en el año 2006 por Jack Dorsey. Es una red social que tiene un total de 400 millones de usuarios. Dichos usuarios publican ‘tweets’ que pueden ser leídos por el resto de los usuarios. Los tweets estuvieron limitados a 140 caracteres, pero después del 2017 dicho número aumentó hasta llegar a un total de 280, es decir, el doble. Los usuarios pueden adjuntar imágenes, vídeos e, incluso, hipervínculos.

Dentro del ámbito de la red social, se denomina seguidor(‘*follower*’) a aquellos usuarios que se encuentran suscritos a otro usuario y reciben notificación de las publicaciones de este realice. Se denominan amigos (‘*followees*’) a aquellos a los que sigue.

Los ‘tweets’ que son publicados por un usuario son enviados a sus seguidores. El propósito es la compartición de mensajes de longitud breve de manera dinámica. También, Twitter es utilizado por celebridades u organismos públicos y privadas para divulgar noticias u opiniones. Este tipo de usuarios son los que poseen un mayor número de seguidores.

Los usuarios valoran los ‘tweets’ dando me gusta(‘*like*’), o compartiendo dicho mensaje con sus seguidores(‘*retweet*’). También es posible responder a un ‘tweet’, mencionar a otros usuarios o compartirlo con otra red social. La forma más habitual para crear interrelaciones es a través de etiquetas(‘*hashtags*’), todos los ‘tweets’ que tengan dicha etiqueta se vincularán entre sí. Si una etiqueta aparecerá un gran número de veces, esta se incluirá dentro de los temas del momento (‘*trending topic*’). Los usuarios podrán realizar búsquedas en dichas listas, para averiguar de qué se está hablando y poder participar en dicha interacción [5].

Acerca de la influencia global de Twitter, en un análisis global de las redes sociales más influyentes y populares del mundo, Twitter aparece en el puesto número dieciséis con aproximadamente 400 millones de usuarios en la primavera del 2021 [6]. Si centramos nuestro foco en

⁹ <https://twitter.com/> [Disponible a 23/01/2022]

el mundo occidental, la relevancia de esta aplicación aumenta de manera considerable. Concretamente en España, Twitter ocupa el tercer lugar en el ranking de redes sociales más populares [7].

Twitter resulta una plataforma abierta fácil de analizar y que presta un gran volumen de datos. Actualmente los usuarios publican aproximadamente 500 millones de ‘*tweets*’ diarios [8].

Uno de los datos que resultan de especial interés a la hora del análisis es la ubicación de los usuarios. De esta manera, si conocemos la opinión que tienen los usuarios de un determinado producto esta será de gran ayuda a las empresas para implementar mejoras, pero esto no sirve de nada si no conocemos donde se ubican estos usuarios. Es aquí donde cobran gran importancia las aplicaciones de geolocalización.

Por otro lado, los usuarios cada vez están más concienciados con su privacidad. Pese a que Twitter permite a los usuarios geolocalizar sus tweets, son muy pocos los que lo hacen. De hecho, se llegaron a tener niveles tan bajos del uso de esta herramienta, que años atrás Twitter se planteó su eliminación [9].

2.1.2 Funcionalidades relacionadas con la geolocalización

Como hemos destacado anteriormente, en Twitter es posible modificar la configuración para que el usuario incluya texto en el campo ‘*localización*’. Aquí, puede escribir la ubicación que considere oportuna. Es posible dentro de este campo encontrar todo tipo de localizaciones desde direcciones postales hasta palabras del estilo Marte, pasando por nombres de ciudades, o bien el nombre del país. Además, son pocos los usuarios que rellenan la información de este campo. Consecuentemente, podemos extraer la localización a partir de estos campos, pero no podemos saber hasta qué punto dicha información es fiable.

Hoy en día, al darnos de alta en Twitter, se rellena en nuestro perfil el campo ‘*país*’. Esto sirve para que la aplicación personalice o establezca relación con los distintos campos que nos podrían interesar. El usuario, si posteriormente quiere, podrá modificar dichos datos. Dicha información no está disponible en la API de Twitter, sino que se considera como privada.

También, los ‘*tweets*’ pueden contener una localización asociada. Anteriormente, podían contener una latitud y una longitud desde donde se realizaba la publicación y era extraída de la ubicación GPS del dispositivo. Debido al escaso uso, esta funcionalidad fue eliminada en el año 2019.

A pesar de lo anterior, los usuarios pueden incluir la localización al redactar un ‘*tweet*’, seleccionando la ciudad o el país de una lista que se encuentra preestablecida. La lista de datos es generada por la aplicación *FourSquare*¹⁰, que permite crear a los usuarios ubicaciones y dar su opinión acerca de ellas. FourSquare fue creada por Dennis Crowley, Naveen Selvadurai, Harry Heymann, Nathan Folkman y Mike Singleton en el 2009 [10].

Los tweets que tienen asociada una localización contienen metadatos que son accesibles a través de la API de Twitter. También existen aplicaciones como la mencionada anteriormente FourSquare, que permiten a los usuarios realizar una publicación cada vez que adjunten una nueva ubicación.

Otra información que nos puede resultar de gran interés a la hora de obtener la ubicación de un usuario, son los seguidores o a aquellas personas a las que sigue dicho usuario. En muchos casos, estos usuarios son personas que viven en una zona anexa al usuario.

La información temporal también puede ser relevante. En una secuencia de tweets si la diferencia de tiempo con el siguiente no es demasiado grande, podremos localizar un conjunto de publicaciones que hayan sido realizadas en un período de tiempo determinado. Continuando con lo anterior, en el perfil del usuario se puede guardar información sobre el uso horario que estamos utilizando.

¹⁰ <https://foursquare.com/> [Disponible a 11/03/2022]

2.1.3 El problema de la geolocalización

Analizando lo visto en el anterior apartado, el principal problema es que los usuarios deciden no localizar sus publicaciones. Esto lo podemos ver, con el ejemplo de que en 2019 Twitter decidiera retirar geolocalización de sus ‘tweets’. Esto se ve reforzado con un aumento de la privacidad por parte de los usuarios, lo que hace que cada vez sea más raro que los usuarios publiquen su localización.

También, gran parte de la información que Twitter almacena de sus usuarios es privada, con lo cual no podemos tener acceso a dicha información. Consecuentemente, la localización se dificulta al tener más datos relacionados con dichas publicaciones.

Debido a lo anterior, tendremos que hacer uso de otras herramientas, como bien pueden ser aplicaciones de geolocalización inversa, buscadores que trabajan a partir de imágenes o buscadores que realizan búsquedas a partir de metadatos. Durante este trabajo presentaremos varias de las herramientas más útiles que existen hoy en día.

2.2 Métodos para determinar la ubicación de una imagen

Cualquier fotografía en formato digital cuenta con dos apartados: la primera parte la imagen en sí y por otra parte los metadatos [11]. Estos últimos se suelen encontrar dentro de las propiedades del archivo, pero es un tipo de información que se recoge en las fotografías digitales que bien pueden haber sido tomadas desde una cámara réflex o desde un dispositivo móvil.

Los metadatos guardan información sobre el tipo de imagen, el tiempo o la ubicación donde se realizó esta fotografía. En el caso de las cámaras digitales, estos también almacenan información acerca de la marca, el modelo de la cámara, tipo de compresión, flash, apertura del diafragma o el ISO, entre otros. Sin embargo, los dispositivos móviles, permiten añadir otro tipo de información como puede ser servicio de ubicación o GPS, esto nos permitirá plasmar la ubicación geográfica exacta desde donde fue plasmada la imagen.

A parte de la localización a partir de los metadatos de la imagen, podemos realizar lo que se denomina como búsqueda inversa de imágenes, es decir, utilizar aplicaciones que dada una imagen buscan a través de algún mecanismo imágenes similares y a partir de ellas se determina su localización.

Herramientas con estas características nos permiten establecer relación entre la imagen de la cual queremos obtener la localización, con otras imágenes las cuales se encuentren publicadas en internet, y sobre las que podremos obtener información acerca de la imagen. Ejemplos de esto son buscadores como Google o Yandex¹¹.

En nuestro trabajo, nos centraremos en este segundo tipo de herramientas, pese a que en algún determinado momento trabajaremos con los metadatos de determinadas páginas web. A continuación, presentaremos una serie de herramientas para la localización de imágenes, realizando una distinción entre aplicaciones y buscadores [11].

2.3 Análisis de herramientas

En este apartado analizaremos aquellas herramientas que se encuentran disponibles hoy en día y que permiten realizar diversas tareas consideradas de interés para este TFG partiendo de imágenes,

¹¹ <https://yandex.com/> [Disponible a 14/03/2022]

como por ejemplo la búsqueda inversa de imágenes. Como recalcamos en el apartado anterior, haremos una distinción entre aplicaciones y buscadores.

2.3.1 Aplicaciones

En las aplicaciones que pasaremos a describir, se detallarán aquellas que hemos considerado de interés en relación con este TFG. Algunas de estas aplicaciones entre otras cosas permiten obtener imágenes similares.

2.3.1.1 PictPicks

Conocido anteriormente como Image Search Viewer, *PictPicks*¹² es una aplicación que permite realizar búsquedas de imágenes a partir de una consulta textual o de una foto (almacenada en la galería o bien tomada desde un dispositivo móvil). Esas búsquedas se hacen a través del buscador de Google quien devuelve sus resultados y se visualizan en su interfaz de usuario. Estos resultados se suelen corresponder con imágenes, aunque la aplicación también nos permite acceder a fuentes de imágenes e interactuar con otras aplicaciones.

Un inconveniente que tiene es el hecho de que presenta demasiada publicidad, pudiendo ser incluso molesta para el usuario. La aplicación devuelve imágenes, no localizaciones por lo que no nos resultaría de interés [12]. Además, no dispone de API para su utilización.



Figura 1 - PictPicks (tomada de [12])

2.3.1.2 Camfind

*Camfind*¹³ fue creada en el año 2013 por Dominik Mazur, es una aplicación móvil que realiza búsqueda visual y reconocimiento de imágenes. Permite a los usuarios identificar cualquier elemento tomando una foto con su dispositivo móvil [13]. Para ello, utiliza su propia API de reconocimiento de imagen de CloudSight.

Camfind permite realizar el escaneo de códigos QR, asociándolos con una determinada URL. La aplicación tiene distintas funciones, entre otras, puede utilizarse como una aplicación social para la compartición de imágenes. También posee un algoritmo que nutre a los usuarios con publicaciones basadas en sus intereses [14]. Además, la aplicación reconoce objetos, pero no establece relación con imágenes similares.

¹² Descarga en: <https://play.google.com/store/apps/details?id=jp.mydns.usagigoya.imagesearchviewer&hl=es&gl=US>
[Disponible a 11/03/2022]

¹³ Descarga en: https://play.google.com/store/apps/details?id=com.msearcher.camfind&hl=es_419&gl=US
[Disponible a 11/03/2022]



Figura 2 - Camfind (tomada de [6])

2.3.1.3 Picturethis

*Picturethis*¹⁴ fue creada por la empresa Sony. Esta aplicación ha sido desarrollada para los amantes de las plantas, permitiendo al usuario reconocer casi cualquier tipo. Al igual que las aplicaciones que hemos visto anteriormente, permite trabajar tanto con fotos que son tomadas en el momento desde un dispositivo móvil o bien con fotos que se encuentran almacenadas en la galería de imágenes del mismo.

No solo permite identificar las plantas, sino que también permite realizar un diagnóstico cuando la planta parece estar enferma. También permite a los usuarios compartir en el momento la localización de un determinado tipo de planta para que el resto de los usuarios puedan conocerlas también.

Pese a permitir publicar imágenes geolocalizadas, la aplicación no nos permite obtener ubicaciones relacionadas con las plantas [15].

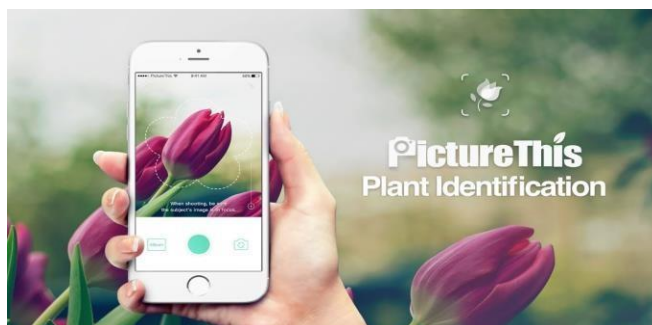


Figura 3 - Picturethis (tomada de [8])

2.3.1.4 Plant Snap

La aplicación fue creada en el año 2012 por Eric Ralls [16]. *Plant Snap*¹⁵ nos permite identificar diferentes tipos de plantas a través de una imagen de la galería o realizando una fotografía en tiempo real.

Se estima que la aplicación es capaz de detectar el 90% de los árboles y plantas existentes. También nos permite clasificar nuestras fotografías y crear nuestras propias colecciones [17].

¹⁴ Descarga en: https://play.google.com/store/apps/details?id=com.msearcher.camfind&hl=es_419&gl=US
[Disponible a 11/03/2022]

¹⁵ <https://www.plantsnap.com/> [Disponible a 11/03/2022]

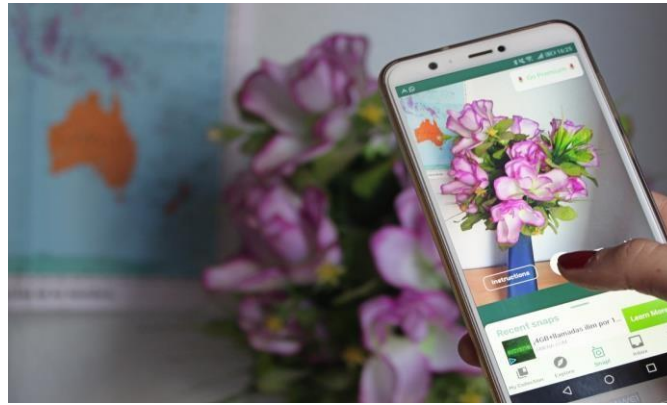


Figura 4 - PlantSnap (tomada de [11])

2.3.1.5 Search by Image

*Search by Image*¹⁶ es una herramienta que consta de un buscador que nos permite utilizar imágenes bien desde la galería de fotos, utilizando la cámara del dispositivo o adquiriéndola desde las redes sociales. Esto nos permite conocer; por ejemplo, si las fotos del perfil de un usuario son reales o, en verdad, no pertenecen a esa persona. También realiza la búsqueda de imágenes similares o la búsqueda con pequeñas porciones de la imagen. La principal ventaja de esta aplicación con respecto a otras es que Search By Image funciona mejor con retratos de personas [18].

El buscador puede resultar de ayuda para la búsqueda de imágenes similares. De esta manera, nos podría ayudar a extraer localizaciones de páginas web que contengan imágenes con un cierto parecido.



Figura 5 - Search by Image (tomada de [19])

2.3.1.6 Photo Sherlock

*Photo Sherlock*¹⁷ fue creada en el año 2011 por Yhor Prysazhnjuk. Esta aplicación nos permite conocer información acerca de una fotografía. No solo te muestra si un individuo está utilizando fotografías de otras personas, sino que la aplicación es capaz de mostrar información acerca de las personas que aparecen en la foto y llegar a mostrar; incluso, las redes sociales o búsquedas relacionadas con esas personas [20].

¹⁶ Descarga en: <https://play.google.com/store/apps/details?id=com.palmteam.imagesearch&hl=es&gl=US> [Disponible a 11/03/2022]

¹⁷ Descarga en: <https://play.google.com/store/apps/details?id=hippeis.com.photochecker&hl=es&gl=US> [Disponible a 11/03/2022]

Esta aplicación está más centrada en trabajar con personas que con localizaciones. Nos podría resultar útil para obtener información relacionada con un usuario, y extraer a partir de imágenes o sus redes sociales las posibles localizaciones donde podría haber sido tomada una imagen.



Figura 6 -PhotoSherlock (tomada de [21])

2.3.1.7 Google Lens

Google es una empresa que posee un gran número de servicios como bien pueden ser *Drive*¹⁸, *Maps*¹⁹... los cuales generan ingresos dando soporte a aplicaciones de terceros. A mediados de 2015 Google desarrolló una API que fue conocida con Google Vision API. Esta API supuso las bases para la creación de *Google Lens*²⁰ en 2017.

Se trata de una herramienta que se puede descargar bien como aplicación de Android e iOS, pero también se encuentra integrada en *Google Fotos*²¹ y en el buscador *Google Chrome*²². Así, la herramienta posee una gran cantidad de aplicaciones como bien pueden ser:

- Traducción de textos en tiempo real, enfocando el texto con tu cámara desde la aplicación. De esta manera detectará el idioma y realizará la traducción. También se podrá realizar traducciones parciales seleccionando la parte del texto en la que estamos interesados. Tras realizar la traducción, esta se podrá almacenar en el portapapeles. La aplicación también te permite interactuar con *Google Translate*²³.
- Análisis de textos, tanto escritos a mano como con cualquier dispositivo, permitiendo su almacenamiento y su posterior narración. Además, aporta una serie de links en los cuales se han detectado fragmentos del texto o palabras relacionadas con el texto. También permite analizar fechas e interactuar con el calendario del móvil. Lo mismo ocurre con los números de teléfono, con los que podremos realizar llamadas a costa de relación que ocurre entre ambas aplicaciones.
- Identificación de animales, plantas u objetos, simplemente apuntando el objetivo con la cámara, aportará la información que haya encontrado que esté relacionada. En el caso en el que haya más de un objeto en el visor, tendremos que seleccionar alguno de los puntos a ser analizados. Adicionalmente, en el caso de objetos también nos mostrará ofertas para su compra.

¹⁸ https://www.google.com/intl/es_es/drive/ [Disponible a 16/03/2022]

¹⁹ <https://www.google.es/maps/?hl=es> [Disponible a 16/03/2022]

²⁰ <https://lens.google/intl/es-419/> [Disponible a 11/03/2022]

²¹ <https://www.google.com/intl/es/photos/about/> [Disponible a 11/03/2022]

²² Descarga en: <https://play.google.com/store/apps/details?id=com.android.chrome&hl=es&gl=US> [Disponible a 11/03/2022]

²³ <https://translate.google.es/?hl=es> [Disponible a 14/03/2022]

- Obtención de información acerca de libros y películas.
- Ayuda en la realización de tareas. Esto está enfocado en ayudar a estudiantes. Analiza tanto operaciones matemáticas como cualquier tipo de pregunta. En el caso de que la aplicación no encuentre ninguna respuesta adecuada, buscará contenido relacionado en el buscador.
- Escaneo de códigos de barras, relacionándolo con tiendas que ofrezcan el mismo producto. Esto nos permitirá conocer mejor el mercado.
- Identificación de lugares, esta es la función de la aplicación la cual nos interesa puesto que es la que guarda una mayor relación con el trabajo. De esta manera, enfocando la cámara a un determinado edificio, objeto, monumento... la aplicación mostrará el lugar donde se encuentra ubicado o bien mostrará búsquedas webs relacionadas.
- Identificación de platos de comida. Enfocando con la cámara a un plato, indica qué tipo de comida se trata y en la parte inferior mostrará una serie de recetas que nos ayudan a poder cocinarlo posteriormente. Además, nos dará más información adicional sobre los alimentos que estamos ingiriendo. Opuestamente, nos mostrará imágenes sobre un determinado plato de comida, analizando la carta y destacando el plato en el que estemos interesados [22].



Figura 7 - Google Lens (tomada de [23])

Google Lens supone la aplicación más completa de todas las analizadas en parte debido a la gran cantidad de servicios que ofrece.

2.3.1.8 Creepy

La herramienta *Creepy*²⁴ permite buscar y seleccionar información de una cuenta de Twitter u otras redes sociales como Flickr, para posteriormente extraer la información GPS y temporal. Para ello, se genera una base de datos que permite la visualización de dichas coordenadas, filtrando previamente la información hasta llegar a conseguir la ubicación exacta. De este modo, y una vez exportada esa información a un archivo, se puede representar en un mapa a través de Google Maps para su análisis.

En principio, solo se utiliza para extraer información que el usuario ha puesto en internet, esta información generará un fichero el cual puede ser complementado con otras fuentes de información para obtener la pesquisa deseada.

²⁴ <https://www.geocreepy.com/> [Disponible a 11/03/2022]

Como hemos reseñado anteriormente, la aplicación recoge información de posicionamiento que se obtiene de redes sociales o fotos que publica un usuario en internet. La información de posicionamiento que recoge Creepy de Twitter es [24]:

- Información GPS (cuando se hace desde dispositivos móviles).
- Tweets asociados a una ubicación.
- Triangulación basada en la IP desde la que se hizo el tweet.
- Metadatos.

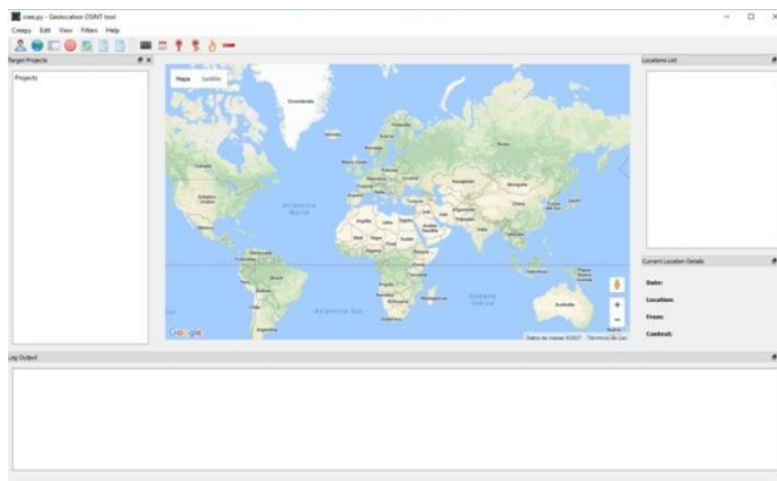


Figura 8 - Creepy (tomada de [25])

Esta herramienta nos puede resultar de gran ayuda a la hora de extraer datos de imágenes que estén en la web, que posteriormente nos permitan obtener y representar dicha localización en un mapa. La representación en un mapa es un punto importante a tener en cuenta, puesto que no muchas aplicaciones lo integran dentro de sus funcionalidades.

Creepy no proporciona datos acerca de la ubicación de un ‘tweet’ en cuestión, sino que extrae la información pública de geolocalización del usuario, así como la información de geolocalización extraída de las etiquetas EXIF de las fotografías publicadas. Así si el usuario no geocaliza sus publicaciones ni sus imágenes, no recuperará dicha ubicación.

2.3.2 Buscadores

En los buscadores que pasaremos a describir, se detallarán aquellos que hemos considerado de interés en relación con este TFG. Algunos de ellos se centran en realizar búsquedas usando imágenes y otros partiendo de una consulta textual. A continuación, presentaremos algunos de los buscadores que hemos seleccionado

2.3.2.1 WeSee

WeSee²⁵ es un buscador que utiliza imágenes para conseguir establecer relaciones con las fotografías que ya se encontraban almacenadas previamente en la aplicación. WeSee mostrará las imágenes con una mayor similitud, arrojando una gran cantidad de resultados diversos.

Adicionalmente, WeSee permite obtener resultados a partir de palabras clave. Esto nos complementará con la búsqueda que estamos realizando a través de imágenes [26].

²⁵ <https://www.wesee.com/> [Disponible a 11/03/2022]

El buscador nos puede resultar útil al trabajar con una gran multitud de imágenes que ya tiene almacenadas. Consecuentemente, al introducir nuestra fotografía el buscador nos arrojará un número alto de imágenes relacionadas, de donde podremos obtener la localización que más concuerde.

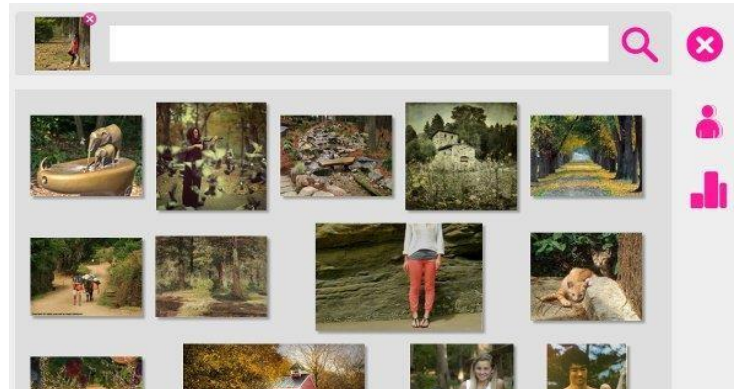


Figura 9 - WeSee (tomada de [27])

2.3.2.2 Yandex

*Yandex*²⁶ es un buscador ruso perteneciente a la compañía homónima que fue fundada en el año 1997 por Arkady Volozh e Ilya Segalovich. Se trata del buscador más utilizado en Rusia y en los antiguos países pertenecientes a la Unión Soviética, tiene más de 65 millones de visitas diarias.

El éxito en Europa del este se debe a que el algoritmo de su buscador comprende el idioma ruso, tarea que otros buscadores no priorizaron hasta años más tarde [28].

El buscador Yandex es muy útil a la hora de encontrar imágenes similares a las que aparece una determinada persona, los resultados son bastante exactos. Por lo que este buscador nos podrá resultar interesante a la hora de buscar imágenes relacionadas con un usuario, tratando de conseguir las localizaciones más frecuentes de las fotografías de dicho usuario [29]

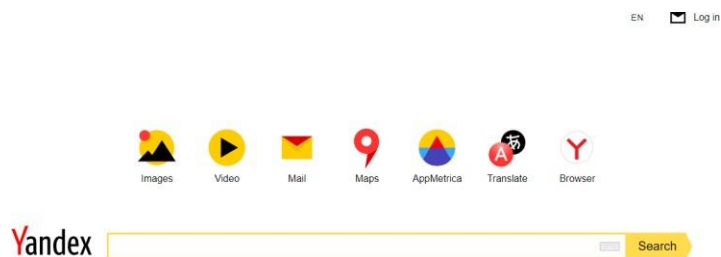


Figura 10 - Yandex (tomada de [30])

2.3.2.3 Google

El buscador de *Google*²⁷ es el motor de búsqueda más utilizado en la web, por ello recibe millones de consultas diarias. Fue desarrollado por Larry Page y Sergey Brin en 1997 [31].

El buscador de Google incluye distintas herramientas las cuales lo hacen más versátil a la hora de realizar búsqueda inversa de imágenes. Entre ellas cabe destacar, la API Google Vision que sirve de

²⁶ <https://yandex.com/> [Disponible a 12/03/2022]

²⁷ <https://www.google.es/> [Disponible a 12/03/2022]

base para la aplicación Google Lens, integrada dentro del buscador de Google (explicado en 2.3.1.7). En segundo lugar, está la API Google Geocoding que trabaja con tres componentes principales: nombres, coordenadas e identificador único, esta API trata de mostrar al Usuario datos legibles para el ser humano como nombres de países, localidades o calles. También transforma las coordenadas a direcciones a través de un proceso conocido como geocodificación inversa [32].

Hoy en día, el buscador de Google es el más completo para realizar búsqueda a partir de imágenes, puesto que es el que más herramientas ofrece al usuario. Se tendrá en cuenta para el trabajo, especialmente la API de Google Vision, y el concepto de geocodificación inversa.

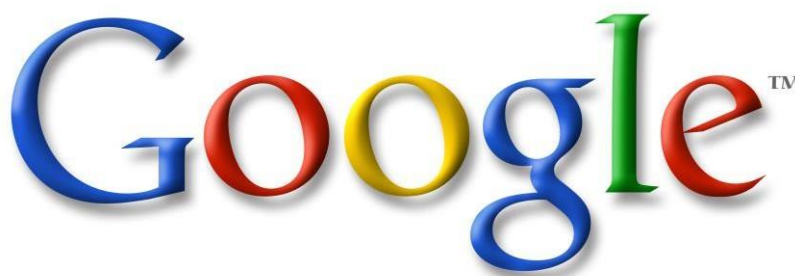


Figura 11 - Google (tomada de [33])

2.3.3 Comparativa

A lo largo de este apartado, hemos podido visualizar una serie de aplicaciones y buscadores que trabajan con imágenes. Extrayendo conclusiones de ambos grupos, observamos que pocas aplicaciones realizan labores de obtención de localizaciones a partir de imágenes, es decir, las aplicaciones se encuentran más enfocadas en otras tareas como pueden ser identificación de plantas u objetos dentro la imagen (PlantSnap o Camfind). Sin embargo, dentro de los buscadores, el número de estos que realizan este tipo de búsqueda es mayor que el de las aplicaciones que realizan la misma tarea.

Dentro de las aplicaciones, vemos que la mayoría de ellas tienen un enfoque más relacionado con el concepto de red social. Lo anterior, es debido a que la mayoría de las aplicaciones se centran en compartir imágenes entre los usuarios.

Tenemos diferentes tipos de aplicaciones como pueden ser Picturesthis o PlantSnap, que se encuentran más enfocadas en la identificación y el trabajo con plantas, otras como PictPicks, PhotoSherlock o SearchByImage, utilizan un buscador por debajo de la aplicación. Otras aplicaciones como Camfind, son usadas para detectar elementos de imágenes, para ello hacen uso de una API. En este caso, Camfind usa la API *Cloudsight*²⁸.

En cuanto a los buscadores, observamos como existe una mayor tendencia a que estos realicen búsqueda a través de imágenes. Si bien todos los buscadores estudiados nos podrían servir de ayuda, los que presentan mejores prestaciones son el buscador de Google y Yandex. Yandex; en especial, por los buenos resultados que cosecha a la hora de trabajar con rostros y rasgos faciales, faceta en la cual el buscador de Google no es tan sofisticado. Pese a lo anterior, nos interesa más localizar la imagen que determinar quien aparece en la fotografía, por lo que nos interesan más los servicios que ofrece el buscador de Google.

De las aplicaciones que hemos estudiado anteriormente, la única que nos podría resultar de ayuda sería *Google Lens*, la cual se puede usar de forma simultánea al buscador de Google o, bien, como aplicación de escritorio. Pero, lo que realmente nos interesa es acceder a las herramientas que utiliza Google Lens para realizar sus labores de búsqueda, para ello accederemos a la API Google Vision a través de Google Cloud, esta API será estudiada en el siguiente apartado de la memoria

²⁸ <https://cloudsight.ai/> [Disponible a 15/03/2022]

3 DESARROLLO DEL TFG

Para el desarrollo de este apartado comenzaremos explicando las diferentes herramientas que necesitamos para la confección de GeoApi, el sistema creado. Empezaremos explicando el lenguaje de programación que utilizaremos para desarrollarlo, en nuestro caso Python. Seguiremos introduciendo el entorno de desarrollo integrado (IDLE) que hemos utilizado para compilar nuestro código. Después, haremos una introducción a la API Google Vision, la cual nos permitirá buscar imágenes similares y obtener información de ellas. Esa información será proporcionada en un formato manejable, como es el caso del formato JSON.

Con respecto al modo de obtención de las imágenes, este se puede hacer partiendo de archivos elegidos por el usuario o ser descargadas directamente de la red social Twitter a partir de una librería de Python denominada *Snsrape*, la cual explicaremos brevemente. Teniendo esto en cuenta, introduciremos para familiarizarnos la base de datos MongoDB sobre la que nos apoyaremos para el almacenamiento de la información de la publicación que contenga la/s imagen/es procedente de Twitter. En este sentido, de nuevo recurriremos al formato JSON. A parte de las herramientas anteriores, también nos ayudaremos de las librerías de Python *metadata_parser*, para la obtención de los metadatos asociados a una URL, y de *GeoPy* que nos permitirá realizar las operaciones de geocodificación²⁹ y geocodificación inversa³⁰, permitiendo así representar las coordenadas en un mapa.

Una vez definidas las herramientas que vayamos a utilizar, explicaremos las tres arquitecturas generales desarrolladas que compondrán nuestro trabajo. Todas ellas estarán constituidas por un sistema que incluirá un algoritmo, que analizará la información obtenida de las imágenes por la API de Google Vision, que será otra de las partes que estará presente en las diferentes arquitecturas. Por último, tendremos un sistema para la representación de coordenadas en un mapa a partir de las localizaciones proporcionadas por el algoritmo seleccionado.

Por tanto, primero se procederá a explicar la primera arquitectura en la que supondremos que las imágenes serán proporcionadas por el usuario. A continuación, se detallará la segunda arquitectura, donde las imágenes serán obtenidas de Twitter a partir de la librería de Python *Snsrape* y toda la información asociada a la publicación será almacenada en la base de datos MongoDB. Por último, introduciremos una serie de mejoras implementadas en el tratamiento de imágenes.

Comencemos entonces por describir las herramientas necesarias para desarrollar el trabajo.

²⁹ Transformación de una dirección postal a coordenadas.

³⁰ Transformación de coordenadas a una dirección postal.

3.1 Herramientas necesarias

3.1.1 El lenguaje de programación Python

Los lenguajes de programación sirven de herramienta para la construcción de programas. Concretamente, el lenguaje de programación Python³¹ fue creado por Guido Van Rossum entorno a los años 90 y ha ido consiguiendo popularidad debido a la sencillez que presenta para la resolución de problemas.

Python se encuentra basado en el lenguaje ABC (un lenguaje de programación de propósito general desarrollado en el Centrum Wiskunde & Informatica) y también influenciado por otros lenguajes como C. Este lenguaje de programación se puede clasificar como imperativo (instrucciones se dan paso a paso, de forma secuencial) y funcional (en él, el programador especifica lo que quiere hacer, las funciones van en la primera parte y utilizaremos variables a los que se le pueden asignar diferentes valores). Es un lenguaje de alto nivel (no expresa algoritmos teniendo en cuenta la capacidad de las máquinas, tiene en cuenta las capacidades cognitivas de los seres humanos) que permite realizar acciones con pocas líneas de código.

La sintaxis es simple, llegándose a considerar un pseudocódigo (es un falso lenguaje que permite escribir los pasos que va a realizar el programa de una manera similar al lenguaje de programación que utilizaremos en el código). [34]



Figura 12 - Python (tomada de [35])

3.1.2 Entorno de desarrollo (IDLE)

Debido al éxito que ha tenido Python, se han desarrollado varios entornos de desarrollo integrado, entre los que cabe destacar IDLE³² que se constituye como uno de los más importantes. IDLE fue desarrollado por el mismo creador de Python Guido Van Rossum en 1998.

IDLE es un entorno de desarrollo sencillo desarrollado para Python. Es un entorno idóneo para pequeñas aplicaciones y se encuentra distribuido entre los principales sistemas operativos, como bien pueden ser: Windows, Linux o MacOS.

También es un entorno interactivo en el que se pueden ejecutar instrucciones sueltas de Python. Además, posee otras funciones como pueden ser: editores de texto, sangría inteligente o resaltado de sintaxis [36].

³¹ <https://www.python.org/downloads/> [Disponible a 12/03/2022]

³² <https://docs.python.org/3/library/idle.html> [Disponible a 12/03/2022]

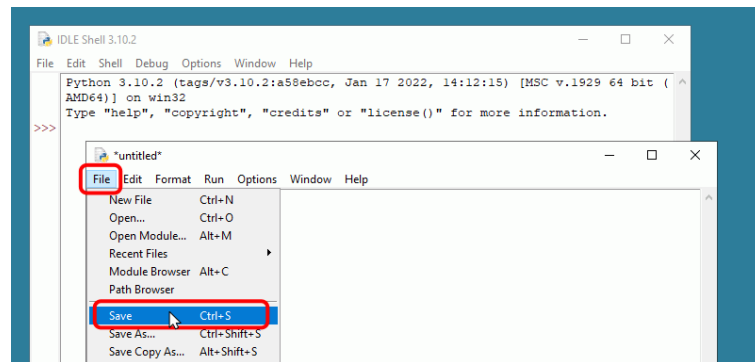


Figura 13 - IDLE (tomada de [36])

3.1.3 La API Google Vision

Una interfaz de programación de Aplicaciones (API) está diseñada para que otros desarrolladores creen aplicaciones directamente con la información que les proveen, como por ejemplo, los servicios de Google. En este caso, estas APIs. son desarrolladas por Google y ofrecen los servicios de la compañía a otros desarrolladores. Así, aplicaciones de terceros están habilitadas para utilizar las API para mejorar sus servicios. Algunos ejemplos de estas APIs pueden ser Vision³³, Maps³⁴ o Geocoding³⁵. Un ejemplo de ello es integrar un mapa dentro de una determinada web a través de Google Maps. Es importante destacar que el uso de algunas APIs de Google requiere autorización y autenticación para su uso [37].

A este respecto, si nos centramos en la API Google Vision, esta permite el reconocimiento de imágenes, lo cual sirve de gran ayuda a otros desarrolladores para la confección de aplicaciones de búsqueda inversa de imágenes como es el caso del sistema creado, GeoApi. Google Vision API se encuentra disponible en *Google Cloud*³⁶ y aprovecha la inteligencia artificial; que Google ha estado desarrollando durante los últimos años, para analizar imágenes. La API no presta servicio a usuarios, sino que está enfocada en los desarrolladores, prestándoles diferentes funcionalidades [38].



Figura 14 - Google Vision (tomada de [39])

La API Google Vision nos permite obtener información de nuestras imágenes en la nube y para ello nos presenta dos modelos:

- AutoML Vision: permite el desarrollo de modelos propios de aprendizaje automático. Para ello se subirán fotografías, que servirán para el desarrollo de nuestro modelo a través de la interfaz gráfica AutoML Vision.

³³ <https://cloud.google.com/vision> [Disponible a 12/03/2022]

³⁴ <https://developers.google.com/maps?hl=es-419> [Disponible a 12/03/2022]

³⁵ <https://developers.google.com/maps/documentation/geocoding/overview> [Disponible a 12/03/2022]

³⁶ <https://cloud.google.com/?hl=es> [Disponible a 12/03/2022]

- API de Vision: ofrece modelos de aprendizaje que ya se encuentran preentrenados. Clasifica las imágenes entre diferentes categorías ya predefinidas. Además, puede realizar diferentes funciones las cuales son de gran utilidad como, por ejemplo, detección de objetos o detección de paisajes.

A continuación, se presentarán una serie de esquemas que explican cómo funciona la API en diferentes situaciones.

1. Búsqueda de productos: encuentra productos a través de imágenes.

El usuario es el encargado de cargar la imagen dentro la API, la cual será la que detecte las partes más importantes de la imagen. El usuario podrá seleccionar los productos que desee que se encuentren presente dentro de la imagen y la API le mostrará los resultados relacionados que encuentre dentro de la web.

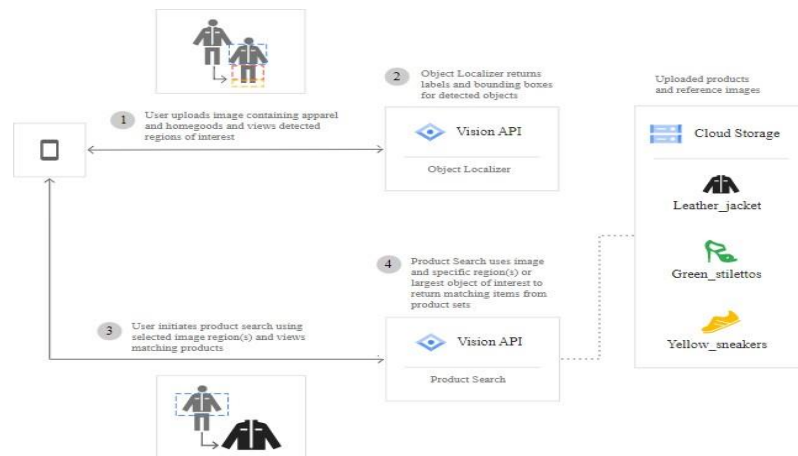


Figura 15 - Detección de productos (tomada de [40])

2. Transcripción y clasificación de documentos: clasifica y accede a la información de manera eficaz.

Documentos PDF y TIFF son cargados en Google Cloud. Posteriormente, la API analizará el documento, lo transcribirá y volverá a almacenarlo en Google Cloud. A continuación, o bien a través de la API clasificaremos, analizaremos las sintaxis del documento, o bien, a través de AutoML clasificaremos el texto a través de lo que ha ido aprendido previamente nuestro modelo, retroalimentándose de forma continua. Sea cual fuere la opción escogida, la aplicación tendrá acceso al documento mejorado.

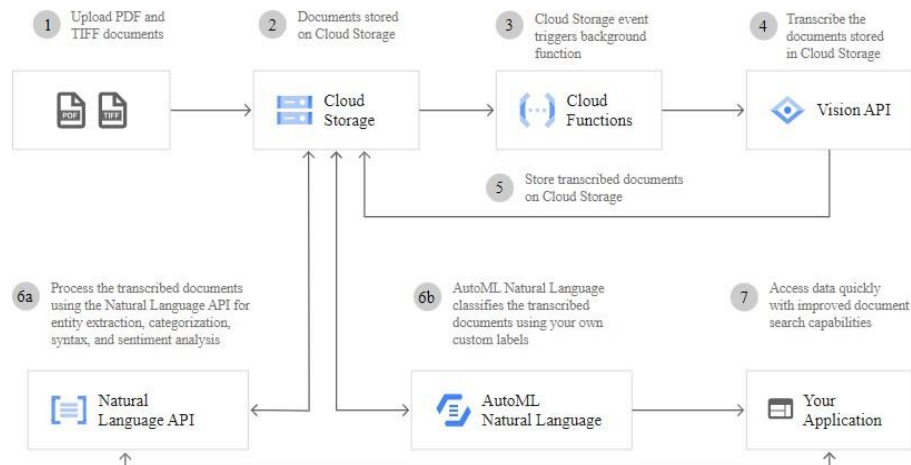


Figura 16 - Transcripción y clasificación de imágenes (tomada de [40])

3. Búsqueda de imágenes similares: utiliza la API de Vision y AutoML Vision para poder clasificar las imágenes.

El usuario será el que cargue las imágenes dentro de la API y estas serán almacenadas en Google Cloud, también podrán ser usadas imágenes tomadas desde un dispositivo móvil. La API Google Vision detecta objetos, personas, monumentos... dentro de la imagen a través de las consultas que realiza por imagen o por palabras. Por otro lado, AutoML Vision asignará categorías y clasificará las imágenes en función a como haya sido entrenada.

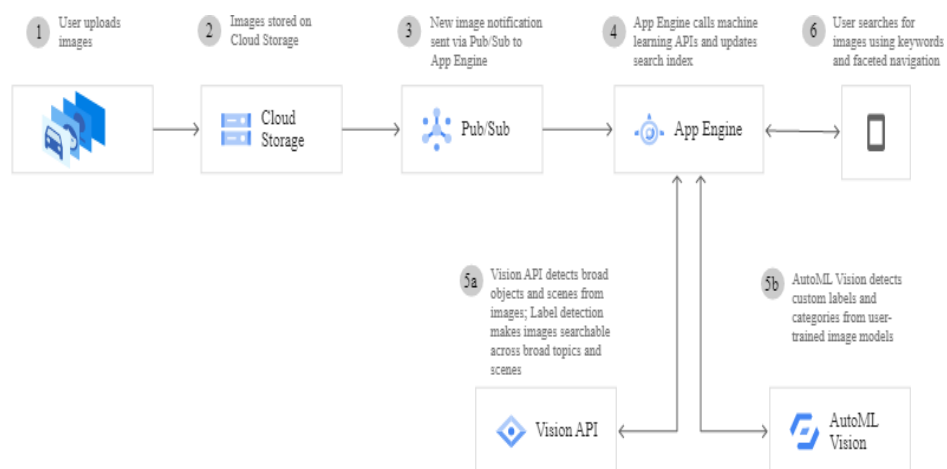


Figura 17 - Búsqueda de imágenes similares (tomada de [40])

Resumiendo, las principales ventajas del uso de la API Google Vision son detectar objetos automáticamente, y ser capaz de clasificar y extraer información de cada uno de los elementos de la fotografía. En segundo lugar, AutoML Vision permite entrenar modelos personalizados para la clasificación automática de imágenes.

Las diferentes funcionalidades que ofrece la API Google Vision son:

1. Detección de páginas webs: a partir de una imagen la API nos permite obtener las diferentes URLs que poseen imágenes relacionadas con dicha fotografía, así como los metadatos asociados a cada URL.

2. Detección de etiquetas: nos permite detectar los diferentes objetos que aparecen en la imagen por sus nombres y nos da la probabilidad con la que la API estima de que ese objeto sea realmente lo que está indicando.
3. Detección de extractos textuales: permite analizar archivos PDF, plasmando todo el texto que es capaz de detectar.
4. Detección de caras: detecta las diferentes caras que aparecen en una imagen y expresa los estados de ánimo de cada uno de los rostros en base a distintas emociones.
5. Detección de contenido explícito: la API reconoce diferentes actividades no permitidas para usuarios menores de edad, como puede ser violencia o racismo. Relacionado con el control parental.
6. Detección de paisajes: permite la detección de estatuas, edificios, calles, etc. En función de lo que hay en la parte posterior de la imagen.
7. Detectar logos: reconoce y detecta diferentes logos de marcas, e indica la probabilidad que asigna la API a que el resultado que ha dado sea realmente correcto [40].

En nuestro trabajo, la API Google Vision, a partir de las funciones de detección de webs y detección de paisajes nos ayudarán a extraer información de las imágenes. Posteriormente, para el análisis de dicha información necesitaremos almacenarla en algún archivo, para ello utilizaremos el formato JSON.

3.1.4 El formato JSON

Surgió en la década de los 90 y fue creado por Douglas Crockford con el objetivo de resolver el problema de que las máquinas se puedan entender entre sí. Los programas utilizaban diferentes sistemas operativos y los sistemas operativos utilizaban diferentes lenguajes de programación.

La primera solución fue XML, pero tenía problemas puesto que trabajaba con una amplia cantidad de datos (procesamiento lento). Consecuentemente, se buscaron formatos menos pesados y veloces para el intercambio de información. A principios de la década de los 2000 comenzó a ser popularizado los formatos JSON³⁷.

El formato JSON es un formato utilizado para el almacenamiento e intercambio de datos de una manera sencilla. Sus siglas significan *Java Script Object Notation* o traducido al español Notación de Objetos de JavaScript. Es un tipo de formato ligero, que resulta sencillo leer y escribir, y además es fácil de interpretar y generar por los programas.

Es un formato de texto que es independiente del lenguaje de programación que estemos utilizando, entre otros podemos utilizar: C, C++, Java, JavaScript o Python. Este tipo de formatos es ideal para intercambiar datos con una API REST (comunicación a través de la web).

JSON define hasta 6 tipos de valores: nulo, números, cadenas, booleanos, matrices y objetos. Pese a sus siglas, no tiene por qué estar en lenguaje JavaScript. Debido a lo anterior, deducimos que los servicios que comparten la información no tienen por qué estar escritos en el mismo lenguaje. Por ejemplo, el emisor puede ser Python y el receptor JavaScript. Cada lenguaje tiene su propia librería para codificar y decodificar este tipo de archivos.

³⁷ <https://www.json.org/json-en.html> [Disponible a 12/03/2022]

Las principales ventajas de JSON son: facilidad para ser comprendido, sencillez, ocupa pocos bytes y tiene una alta velocidad de procesamiento. Por el contrario, tenemos una serie de desventajas que son: falta de extensibilidad, no soporta grandes cargas de datos y requiere mecanismos de seguridad externa [41].

A continuación, en la Figura 18 - Formato JSON (Edición propia) mostramos un ejemplo de un simple formato JSON.

```
{
  "Pedro": {
    "Edad": 34,
    "Hobbie": ["Hockey", "Comer"]
  },
  "Gato": {
    "Edad": 25,
    "Hobbie": ["Comer", "Fútbol"]
  },
}
```

Figura 18 - Formato JSON (Edición propia)

3.1.5 La librería de extracción de mensaje Snsrape

Antes de realizar un análisis de la ubicación de una fotografía, lo primero que necesitamos es la fotografía en sí. Considerando que nuestras imágenes proceden de Twitter, utilizaremos para ello la librería *Snsrape*³⁸.

Snsrape es una librería de Python, que nos permite extraer de una red social. Para ello, Snsrape analiza perfiles, hashtags o búsquedas dirigidas por un usuario, y devuelve la información que ha obtenido.

La librería Snsrape nos permite trabajar con una gran multitud de datos de Twitter como pueden ser: URLs, fechas, contenido de los mensajes, ids, nombres de usuarios, links, 'rtweets' de una publicación, recuento de los me gustas o 'likes' de una publicación, lenguaje, dispositivo desde donde fue realizada la publicación, si la publicación es nueva, los usuarios mencionados y longitud del mensaje, entre otros [42].



Figura 19 - Librería Snsrape (tomada de [43])

³⁸ <https://es.python.org/> [Disponible a 12/03/2022]

3.1.6 La base de datos MongoDB

Una vez hayamos obtenido nuestras imágenes de Twitter y tengamos la información asociada a dichas imágenes en archivos JSON, necesitaremos almacenar dicha información. Para ello utilizaremos la base de datos *MongoDB*³⁹.

MongoDB fue creada en el 2007 por la compañía 10genInc, ahora conocida como MongoDBInc. Es una de las bases de datos más famosas a nivel mundial y principalmente se utiliza para almacenar los datos procedentes de aplicaciones. La base de datos está orientada a documentos, consecuentemente los datos se almacenarán en documentos. Además, no es necesario seguir un esquema preestablecido. Es lo que se conoce como base de datos no estructurada [44].

MongoDB está escrito en C++. Tiene una consola desde la que podemos ejecutar los comandos. En la consola también podemos definir variables, funciones o utilizar bucles. Además, se permite el uso de otros lenguajes de programación, como en nuestro caso Python.



Figura 20 - MongoDB (tomada de [45])

3.1.7 La librería metadata_parser

La librería *metadata_parser*⁴⁰ es una librería de Python cuya función principal es extraer metadatos de los documentos de la web.

Los metadatos son un conjunto de información que se utiliza para caracterizar unos datos. El concepto, se utiliza de forma general, para referirse a aquellos datos que hablan de unos datos, es decir, describen el contenido de los archivos e información de estos.

Actualmente los metadatos han adquirido un importante valor en el contexto del BigData y el IOT. La información utilizada actualmente crece de forma exponencial. Debido a esto, la gestión de la información es de gran ayuda para conseguir una mayor eficiencia en nuestros sistemas [46].



Figura 21 - Librería metadata_parser (tomada de [47])

³⁹ <https://www.mongodb.com/es> [Disponible a 12/03/2022]

⁴⁰ <https://www.python.org/downloads/> [Disponible a 12/03/2022]

3.1.8 La librería GeoPy

GeoPy⁴¹ es una librería de Python que proporciona servicios de geocodificación, es decir, de obtención de las coordenadas de países, ciudades, localidades, monumentos famosos, calles, etc. Para ello utiliza servicios de geocodificación de terceros [48].

GeoPy permite tanto geocodificación, proceso de transformar una dirección postal a coordenadas, como geocodificación inversa, proceso de transformar unas coordenadas a dirección postal.

Además, otra de las funcionalidades más útiles de Geopy consiste en la representación de mapas a partir de coordenadas. La librería permite la representación de un mapa centrado en una latitud y longitud que son fijadas por el usuario.



Figura 22 - Librería GeoPy (tomada de [49])

3.2 Arquitectura general de GeoApi

GeoApi es un sistema sobre el que se han desarrollado algoritmos que proporcionan una localización a partir de una imagen o un conjunto de imágenes. Las bases de GeoApi se encuentran formadas por:

1. Acceso a servicios proporcionados por la API Google Vision:
A través de las herramientas de detección de páginas webs y detección de paisajes obtenemos información para conseguir la localización de una imagen. Dicha información es almacenada en un formato JSON que posteriormente será analizado por el algoritmo.
2. El sistema:
Se han desarrollado cuatro algoritmos que proporcionan una ubicación. En este sentido, tres algoritmos que fueron diseñados para analizar imágenes de forma independiente y un cuarto para analizar un conjunto de imágenes.
3. Representación en mapa:
Una vez obtenida la ubicación a través del algoritmo, dicha localización será representada en un mapa.

Partiendo de estos tres elementos, realizaremos tres esquemas que representarán la arquitectura general del sistema para tres opciones distintas. En la primera, se selecciona una imagen y esta se analiza por uno de los tres algoritmos, dependiendo de las preferencias del usuario. En la segunda, se seleccionará un usuario de la base de datos MongoDB, cuyas imágenes habrán sido almacenadas en un directorio. Cada usuario tendrá un conjunto de imágenes anexas, analizaremos con el cuarto algoritmo las diferentes imágenes para obtener las posibles localizaciones donde el usuario haya podido realizar sus publicaciones. Por último, una tercera parte en la que incluiremos una serie de mejoras en lo

⁴¹ <https://www.python.org/downloads/> [Disponible a 12/03/2022]

relativo a la manipulación de imágenes con la detección, localización y eliminación de objetos e individuos.

3.2.1 Arquitectura general n°1

El proceso seguido a este nivel es el siguiente. En primer lugar, se selecciona una imagen y se analiza por la API Google Vision. La información obtenida se almacena en un formato manejable en JSON. Finalmente, partiendo de esos datos, se aplican tres posibles algoritmos para proporcionar la ubicación, y finalmente se representa en un mapa, tal y como se observa en la Figura 23 - Arquitectura general 1 (Edición propia).



Figura 23 - Arquitectura general 1 (Edición propia)

A continuación, explicaremos cada una de las partes que componen el esquema:

3.2.1.1 API Google Vision

El proceso comienza seleccionando una imagen. Para ello, permitiremos al usuario interactuar a través de una interfaz que le permitirá seleccionar la imagen deseada. Para realizarlo lo anterior, utilizaremos los servicios de la librería tkinter. De dicha imagen se determina la ruta hacia el directorio donde se encuentra el archivo, así como el nombre. Ambos datos serán usados posteriormente para que la API Google Vision pueda acceder a la imagen. Además, el formato JSON será guardado con el mismo nombre de la foto y en el mismo directorio donde se encuentra.

```
#Interfaz con el usuario
Tk().withdraw()
filename = askopenfilename() #Abrir dirección que ha escogido
print(filename)
nombre = os.path.basename(filename) #extraer el nombre del archivo
print(nombre)
```

Figura 24 - Interfaz con el usuario (Edición propia)

Como hemos descrito anteriormente, la API de Google Vision proporciona las funciones de detección de paisajes (Figura 25 - Detección de paisajes (Edición propia)) y de detección de páginas webs (Figura 26 - Detección web (Edición propia)), las cuales utilizaremos para obtener las URLs de páginas webs que contienen imágenes similares juntos con sus descripciones.

En primer lugar, la función de detectar paisajes reconocerá aquellos elementos paisajísticos que sean de remarcable importancia, como puede ser una estatua, un edificio o una escultura, tal y como se

observa en la Figura 25 - Detección de paisajes (Edición propia). Una vez reconocido, almacenaremos la descripción de dicho elemento y, anexo a él, dos valores. El primero es el 'score', que es la importancia o esa confianza que tiene la API de que dicha localización sea correcta. Y, por último, el 'bounding_poly' que indica el área que rodea los cuerpos que está utilizando la API para detectar paisajes (estatua, un edificio o una escultura).

```
#control my image object
image=vision.Image(content=content)
response=client.landmark_detection(image=image)
#print(response) ventana de comandos nops da description con el nombre
#confidence score y bounding poly es ese area para detectar
#por ultimo nos daría esa locations coordenadas de donde estaría

landmarks = response.landmark_annotations
#len(landmarks) nos dara el numero de landscapes que detecta
print(response.landmark_annotations)
for landmark in landmarks:
    description = landmark.description #descripción
    score =landmark.score #score
    loc= landmark.locations #elemento paisajístico
    data['fotos'].append({ #almacenamiento JSON
        'url': '',
        'description': '',
        'title': landmark.description,
        'score': landmark.score,
        'Latitud': '',
        'Longitud': '',
    })

#guardar la info como una tabla
df = pd.DataFrame(columns=['description','score','bounding_poly'])
for landmark in landmarks:
    df =df.append(
        dict(
            description = landmark.description,
            locations = landmark.locations,
            score = landmark.score,
        ),
        ignore_index=True
    )
print(df)
```

Figura 25 - Detección de paisajes (Edición propia)

En segundo lugar, utilizaremos la función de la API Google Vision para la detección de páginas webs con el fin de conseguir las URLs de páginas que contienen imágenes similares, tal y como se observa en la Figura 26 - Detección web (Edición propia). Para ello usaremos diferentes funciones de la API Google Vision como pueden ser:

- 'best_guess_labels': busca la mejor relación existente en la web con nuestra imagen. Se consigue a partir de imágenes similares en la web.
- 'full_matching_images': imágenes de la web que coincidan exactamente con la aportada. Puede incluir copias redimensionadas de la imagen.
- 'pages_with_matching_images': URLs de las páginas web que contienen imágenes que sean coincidentes en Internet.
- 'partial_matching_images': muestra imágenes lo suficientemente similares como para compartir algunas características con nuestra imagen. Por ejemplo, una fotografía original probablemente tendrá algunas con algunos aspectos de nuestra imagen.

- ‘visually_similar_images’: imágenes visualmente similares a la nuestra.

```
#create web detection response
web_detection = response.web_detection

if web_detection.pages_with_matching_images:
    print('\n{} Pages with matching images found:'.format(
        len(web_detection.pages_with_matching_images)))

if page.full_matching_images:
    print('\t{} Full Matches found: '.format(
        len(page.full_matching_images)))

for image in page.full_matching_images:
    print('\t\tImage url : {}'.format(image.url))

if page.partial_matching_images:
    print('\t{} Partial Matches found: '.format(
        len(page.partial_matching_images)))

for image in page.partial_matching_images:
    print('\t\tImage url : {}'.format(image.url))

print('.'*50)

#score and description de las webs encontradas
if web_detection.web_entities:
    print('\n{} Web entities found: '.format(
        len(web_detection.web_entities)))

    for entity in web_detection.web_entities:
        print('\n\tScore : {}'.format(entity.score))
        print(u'\tDescription: {}'.format(entity.description))
```

Figura 26 - Detección web (Edición propia)

La implementación mostrada en la Figura 26 - Detección web (Edición propia) sirve como base para la extracción de información de las imágenes que queremos analizar.

Luego, con las diferentes URLs obtenidas, utilizaremos la librería `metadata_parser` para extraer los metadatos asociados a dichas webs. De esta manera, trataremos de obtener la máxima información posible acerca de la imagen. Observamos en la Figura 27 - Metadatos (Edición propia) el proceso descrito de obtención del título y la descripción procedente de los metadatos de una URL (‘url_prueba’).

```
page2=metadata_parser.MetadataParser(url_prueba)
title1=page2.get_metadatas('title')
description1=page2.get_metadatas('description')
```

Figura 27 - Metadatos (Edición propia)

A medida que vamos extrayendo la información asociada a la imagen es necesario almacenarla, en caso contrario, estaríamos perdiendo dicha información. Para ello utilizaremos los archivos JSON. A la hora de guardar nuestros archivos JSON, será importante hacerlo con el mismo nombre de la imagen que hayamos seleccionado y en la misma carpeta donde se encuentre la imagen. Por ese motivo, utilizaremos las variables que extraíamos a partir de la librería `tkinter`, que nos mostraban el nombre y el directorio asociado a la imagen.

A medida que extraemos la información las iremos almacenando en nuestro formato JSON, tal y como se observa en la Figura 29 - Guardar archivo JSON (Edición propia). Previamente, debemos haber declarado nuestra librería.

```
data['fotos'].append({ #almacenamiento JSON
    'url': '',
    'description': '',
    'title': landmark.description,
    'score': landmark.score,
    'Latitud': '',
    'Longitud': '',
})
```

Figura 28 - Almacenar información JSON (Edición propia)

Realizado lo anterior, nos disponemos a almacenar la información que hemos ido guardando dentro de 'data[fotos]', para ello utilizaremos la función 'json.dump'. Habiendo almacenado el archivo JSON en el directorio determinado, abriremos el archivo en modo lectura, y en la ventana de comandos ejecutaremos 'js' para comprobar que la información se ha almacenado correctamente.

```
with open(f'{filename}{archivo}', 'w') as file:
    json.dump(data, file, indent=4)

#lectura
f=open(archivo, "r")
c=f.read()
js=json.loads(c)
```

Figura 29 - Guardar archivo JSON (Edición propia)

Dentro de API Vision, utilizaremos también la librería GeoPy para la realización de operaciones de geocodificación, tratando de obtener de los metadatos o de la información asociada a una determinada URL, alguna localización. A partir de esta localización, y tras aplicar la librería mencionada, obtendríamos la latitud y longitud. En la Figura 30 - Nominatim (Edición propia) se ve como se importa el geocodificador Nominatim de GeoPy y en la Figura 31 - Geocodificación (Edición propia) el modo en el que se realiza el proceso.

```
#Obtencion coordenadas...Geoname
from geopy.geocoders import Nominatim

# calling the Nominatim tool
loc = Nominatim(user_agent="GetLoc")
```

Figura 30 - Nominatim (Edición propia)

```
# entering the location name
getLoc = loc.geocode(description1)
# printing address
print(getLoc.address)
# printing latitude and longitude
Lat1=getLoc.latitude
Lon1=getLoc.longitude
```

Figura 31 - Geocodificación (Edición propia)

3.2.1.2 El sistema de algoritmos

A continuación, iremos describiendo el sistema. Los datos de entrada del mismo son los ficheros JSON generados de manera automática gracias a la API Vision. Para la confección de nuestro sistema, probaremos tres algoritmos diferentes, cada uno irá aumentando en dificultad y mostrarán un resultado diferente en función de los criterios que cada uno tenga asignados.

Algoritmo 1: Score

Es el algoritmo más sencillo, puesto que únicamente tiene en cuenta el score proporcionado (como hemos explicado anteriormente, es la confianza que tiene la API de que el resultado que muestra sea correcto). Dicha ubicación será obtenida a través de la función de detección de páginas webs o, bien, a través de la detección de paisajes que nos ofrece la API Google Vision.

Partiendo del JSON, se procede a su lectura y los resultados se irán almacenando de manera continua en una lista vacía denominada 'info', tal y como se observa en la Figura 32 - Extraer información de JSON (Edición propia). Para ello, crearemos un ciclo for que irá recorriendo el archivo JSON, aquellas descripciones que no nos proporcionen Latitud o Longitud serán desechadas, mientras que aquellas que si contengan coordenadas serán transformadas en una determinada dirección a partir de la operación de geocodificación que proporciona la biblioteca GeoPy.

```
for foto in data['fotos']: #anlizamos los resultados del formato JSON
    Lat = foto['Latitud']
    Long = foto['Longitud']

    if not Lat: #No contienen Latitud
        continue
    else:
        print(Lat, Long)
        var = Lat, Long
        locname = loc.reverse(var) #Transformación de las coordenadas a dirección
        print(locname)
        #introducimos valor en una lista/append hace que se vaya acumulando
        info.append(locname)
```

Figura 32 - Extraer información de JSON (Edición propia)

La API Google Vision nos irá proporcionando tantas URLs como detecte, iremos almacenando las ubicaciones anexas a dichas URLs en nuestra lista. Una vez tengamos toda la información almacenada, seleccionaremos aquella ubicación que tenga un mayor score y volveremos a transformar de nuevo en coordenadas (Figura 32 - Extraer información de JSON (Edición propia))

```
print(info[0]) #seleccionamos aquella que tenga una mayor score
#Obtencion Coordenadas de localizacion
getLoc = loc.geocode(info[0])
print(getLoc.address)
print("Latitude = ", getLoc.latitude, "\n")
print("Longitude = ", getLoc.longitude)
Lat=getLoc.latitude
Long=getLoc.longitude
```

Figura 33 - Selección coordenadas con mayor score (Edición propia)

Posteriormente, representaremos nuestra ubicación. Para la creación de un mapa interactivo utilizaremos las librerías folium, webbrowser y geocoder Figura 34 - Representación de coordenadas (Edición propia) .

```
#abrir un mapa interactivo
#webbrowser.open("https://www.python.org")
var = Lat, Long
mapa=folium.Map(location = var, zoom_start=12) #seleccionamos la porción de mapa deseada

#crear puntos espaciales
tooltip = getLoc
#añadimos la ubicación
marcador1 = folium.Marker(location= var, popup='ubi', tooltip=tooltip)
marcador1.add_to(mapa) #lo añadimos a nuestro mapa

mapa.save('mimapa.html') #guardamos el archivo
webbrowser.open('mimapa.html') #abrir el archivo
```

Figura 34 - Representación de coordenadas (Edición propia)

Algoritmo 2: Title

Este segundo algoritmo está basado en la variable ‘Title’ proporcionada por la API Google Vision y que se corresponde con los Títulos asociado a las diferentes URLs que son proporcionados por la API en base a las imágenes analizadas. Como hemos visto anteriormente, en el primer algoritmo teníamos en cuenta el factor score, que es esa confianza que tiene la API Google Vision de que el resultado que proporciona sea correcto. Sin embargo, utilizando ‘Title’ esperamos que los resultados sean menos precisos, puesto que el Título más repetido no tiene por qué tener asociado una ubicación.

Al igual que en el primer algoritmo desarrollado, utilizaremos una interfaz con el usuario que permitirá a este abrir la imagen que desee. Seleccionada la imagen, se cargará el JSON creado a partir de la API de Google Vision. De nuevo, para recorrer la información de nuestro JSON utilizaremos un ciclo for, el cual irá recorriendo las diferentes variables ‘Title’ que han sido almacenadas. En el caso de que la variable ‘Title’ no exista, pasará a la siguiente. A medida que el algoritmo vaya recorriendo el fichero, este almacenará la información asociada en la cadena de texto ‘cadena’. Pero, previamente pasará por dos líneas de código que a partir de la librería ‘re’, permitirán eliminar tanto signos de puntuación como números, puesto que esto nos producirá errores a la hora de realizar el recuento de los ‘Titles’(Figura 35 - Ciclo for (Edición propia)).

```
for foto in data['fotos']: #anlizamos los resultados del formato JSON
    title = foto['title']

    if title is None: #No contienen title
        continue

    else:
        string = title
        #eliminamos signos puntuacion
        salida = re.sub(r'[\w\s]', ' ', str(string))
        #eliminamos numeros
        newstring = ''.join([i for i in string if not i.isdigit()])
        new_string = re.sub(r"[^a-zA-Z0-9]", " ", newstring)
        #introducimos valor en una cadena de texto
        cadena += str(new_string) #eliminar signos de puntuación
```

Figura 35 - Ciclo for (Edición propia)

Una vez almacenada la información en nuestra cadena, dividiremos el texto en palabras con la función ‘.split()’ y creamos una lista para almacenar la frecuencia con la que se repite cada una de las palabras que esta presente en los ‘titles’ almacenados. Para realizar lo anterior, aplicaremos un ciclo for que utilizará la cadena de texto previamente dividida y a través del paquete ‘counter’ de la librería collections podremos contar la frecuencia con la que se repite cada palabra. Realizado el recuento, representaremos la cadena de texto, la lista de palabras ya dividida y la frecuencia con que se repite cada una de ellas (Figura 36 - Recuento palabras (Edición propia)).

```
listaPalabras = cadena.split() #separamos las palabras de la cadena de texto

#creamos una lista para representar la frecuencia con la que se repiten las palabras
frecuenciaPalab = []

#recuento de las palabras que hay en la lista
for w in listaPalabras:
    frecuenciaPalab.append(listaPalabras.count(w))

print("Cadena\n" + cadena + "\n")
print("Lista\n" + str(listaPalabras) + "\n")
print("Frecuencias\n" + str(frecuenciaPalab) + "\n")
print("Pares\n" + str(list(zip(listaPalabras, frecuenciaPalab))))
```

Figura 36 - Recuento palabras (Edición propia)

Habiendo completado la lista donde se almacenan las palabras que más se han repetido, utilizaremos otro ciclo for, que recorrerá esa lista y almacenará las palabras que se hayan repetido más de dos veces y tengan una localización asociada (comprobándolo con la librería GeoPy) en la cadena de texto ‘localización’ (Figura 37 - Palabra más repetida (Edición propia)).

```
for valor_a, valor_b in zip(frecuenciaPalab, listaPalabras): #obtenemos los valores en cada iteración
    if valor_a > 2: #si palabra se repite mas de dos veces

        if loc.geocode(valor_b) is None: #si la palabra repetida no tiene asignado una localizacion
            continue

        else:
            localizacion += str(valor_b)
            localizacion +=(' ')

    else:
        continue
```

Figura 37 - Palabra más repetida (Edición propia)

Con la información ya almacenada en la cadena de texto ‘localización’, utilizaremos de nuevo la función ‘counter’ para obtener, ahora sí la palabra que ha sido utilizada con una mayor frecuencia. Con esa palabra; como llevamos a cabo en el apartado anterior, obtendremos la localización a partir de la librería GeoPy (Figura 38 - Palabra más repetida (Edición propia)).

```
print(localizacion)
palabra = localizacion.split()
contador = Counter(palabra)
palabra_frecuente = contador.most_common()[0][0] #palabra más repetida
print(palabra_frecuente) #nos da la palabra que más se repite, dentro de las más frecuentes y tienen localizacion
```

Figura 38 - Palabra más repetida (Edición propia)

Finalmente, y siguiendo lo indicado en la sección Algoritmo 1: Score utilizaremos las librerías folium, webbrowser y geocoder para la creación de un mapa interactivo.

Algoritmo 3: Repetición de localización y máxima score

El tercer algoritmo combina las coordenadas GPS que son obtenidas a partir de la diferentes URLs que son proporcionadas por la API Google Vision, junto con el score que es proporcionado por la misma herramienta. Este algoritmo junto con el primer algoritmo desarrollado son considerados más precisos respecto al segundo debido a la utilización del score, que como hemos mencionado anteriormente es la confianza que tiene la API de que esa ubicación sea correcta.

En primer lugar, como realizamos anteriormente, mostraremos una interfaz la cual permite seleccionar la imagen que se desee. A partir de esta imagen se obtendrá el JSON asociado y se extraerá la información que este contenga. A continuación, crearemos una serie de listas las cuales serán utilizadas posteriormente. Dichas listas son:

- coordLat
- coordLong
- score
- coinLAT
- coinLONG
- coinSCORE

Las tres primeras listas serán usadas para el almacenamiento de la información que proviene del formato JSON, mientras que las tres últimas serán utilizadas para almacenar la información de aquellos archivos en los que la localización se ha repetido en las primeras listas, guardando consigo su score asociado.

Ya declaradas las listas, realizaremos un ciclo for con el cual analizaremos los archivos de nuestro formato JSON. En el caso de que dicha información no contenga latitud, longitud o score será desechada. Si la información contiene los anteriores datos, esta será almacenada en las listas 'coordLat', 'coordLong' y 'score'; que vimos previamente. Antes de su almacenamiento, se procederá a reducir los decimales de la latitud y la longitud a dos decimales (Figura 39 - Lectura Formato JSON (Edición propia))

```
for foto in data['fotos']: #anlizamos los resultados del formato JSON
    Lat = foto['Latitud']
    Long = foto['Longitud']
    Score = foto['score']

    if not Lat: #no contienen Latitud
        continue

    if not Score: #no contienen score
        continue

    else:
        Lat1=float('{0:.2f}'.format(Lat)) #eliminamos dos decimales
        Long1=float('{0:.2f}'.format(Long)) #eliminamos dos decimales
        #introducimos valor en una Lista
        coordLat.append(Lat1)
        coordLong.append(Long1)
        score.append(Score)
```

Figura 39 - Lectura Formato JSON (Edición propia)

Una vez almacenada la información, buscaremos aquellas localizaciones que se encuentren repetidas en nuestra lista con un ciclo for, en el que compararemos los diferentes valores de las Latitudes de la lista 'coordLAT' (Figura 38 - Palabra más repetida (Edición propia)).

```
#busqueda de localizaciones repetidas en la lista
for i, h, j in zip(range(len(coordLat)), coordLong, score):
    for j in range(len(coordLat)):
        if i != j:
            if coordLat[i] == coordLat[j]:
                #almacenamiento valores repetidos en las listas
                coincLAT.append(coordLat[i])
                coincLONG.append(coordLong[i])
                coincSCORE.append(score[i])
            else:
                continue
        else:
            continue
```

Figura 40 - Búsqueda latitudes repetidas (Edición propia)

Como podemos ver en Figura 40 - Búsqueda latitudes repetidas (Edición propia) comparamos una determinada latitud con aquellas que sean distintas y nos quedaremos con las que se repitan. A medida que detectemos latitudes iguales, iremos almacenando sus respectivas informaciones en las listas 'coincLAT', 'coincLONG' y 'coincSCORE'.

Una vez tenemos nuestras localizaciones repetidas ya almacenadas, trataremos de obtener aquella que presente un máximo score. Para lo anterior utilizaremos la función 'max' la cual nos da el valor máximo de una lista.

Obtenido el valor máximo, localizamos la posición de dicho valor dentro de la lista con la función 'index'. Con los valores de latitud y longitud asociados a dicha score, conseguiremos la localización; como hemos visto anteriormente, con la librería GeoPy (Figura 41 - Representación localización repetida máxima score (Edición propia)).

```
#Caso de que haya más de una localizacion coincidente
if len(coincSCORE) > 1:
    maximo=max(coincSCORE) #obtenemos el valor máximo de la lista
    lugar = coincSCORE.index(maximo) #Obtener posicion de dicho valor en la lista
    LAT1 = coincLAT[lugar]
    LONG1 = coincLONG[lugar]
    var = LAT1, LONG1
    print(var)

    #abrir un mapa interactivo
    mapa=folium.Map(location = var, zoom_start=12)

    #crear puntos espaciales
    tooltip = var
    marcador1 = folium.Marker(location= var, popup='ubi', tooltip=tooltip)
    marcador1.add_to(mapa)

    mapa.save('mimapa.html')
    webbrowser.open('mimapa.html')
```

Figura 41 - Representación localización repetida máxima score (Edición propia)

En el otro supuesto de no tener ninguna localización coincidente, obtendremos el valor directamente de las listas 'coordLat', 'coordLong', seleccionando él que tenga un mayor score asociado. Posteriormente; como en el caso anterior, representamos dicha ubicación (Figura 42 - No ubicaciones coincidentes (Edición propia)).

```
#Si no hay coincidentes o solo hay una
else:
    LAT2 = coordLat[0]
    LONG2 = coordLong[0]
    var2 = LAT2, LONG2
    print(var2)
```

Figura 42 - No ubicaciones coincidentes (Edición propia)

3.2.1.3 Representación de localizaciones

La representación de localizaciones la haremos a partir de los resultados que han sido proporcionados por los algoritmos. Dicha representación se lleva a cabo una vez aplicado el correspondiente algoritmo.

Dicho esto, para la creación de un mapa interactivo utilizaremos las librerías folium, webbrowser y geocoder. La librería folium nos permite acceder a los mapas y seleccionar una zona en función a unas coordenadas. Con la función ‘folium_map’ seleccionaremos la escala. Podemos ver las líneas de código que plasman lo explicado en los párrafos anteriores en Figura 34 - Representación de coordenadas (Edición propia).

Para la representación de la ubicación, usaremos la función ‘folium.marker’. Una vez tenemos nuestro mapa interactivo confeccionado, guardaremos el mapa como archivo ‘html’ y lo abriremos a través de la librería webbrowser. Ejemplo de ello es lo que se observa en la Figura 34 - Representación de coordenadas (Edición propia).

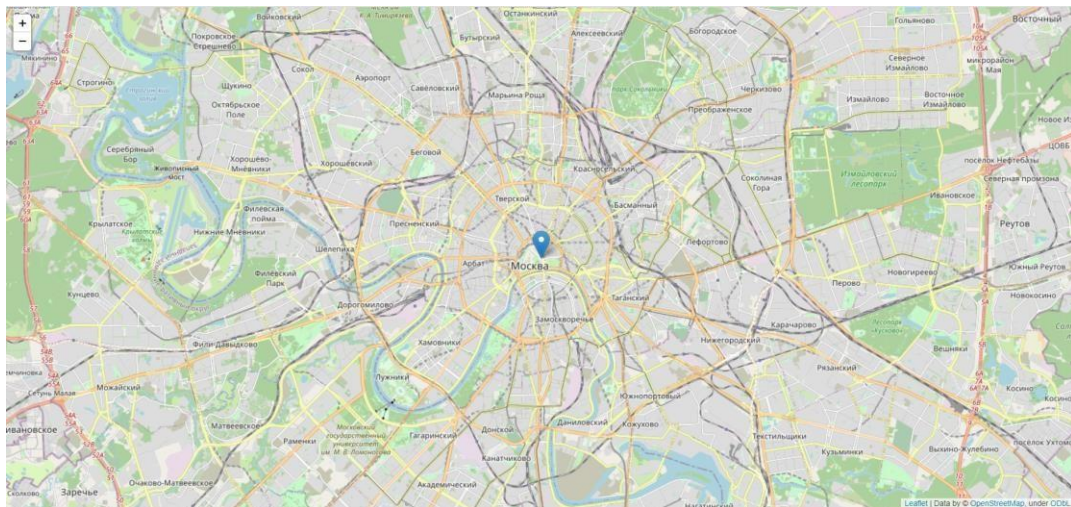


Figura 43 - HTML / representación de coordenadas (Edición propia)

3.2.2 Arquitectura general nº2

En este caso, dado un nombre de usuario de Twitter, se descargan las publicaciones realizadas por él en la red social durante un espacio temporal a partir de la librería Snsrape y toda esa información se almacena en MongoDB. Como en el caso anterior, todas aquellas publicaciones que dispongan de imágenes, cada una será enviada para su análisis a la API Google Vision, la cual extraerá la información necesaria. Dicha información será utilizada por nuestro cuarto algoritmo, que nos dará las probabilidades asociadas a que el usuario se encuentre en una ciudad determinada en ese espacio temporal. La Figura 44 - Arquitectura general 2 (Edición propia) ilustra este proceso.

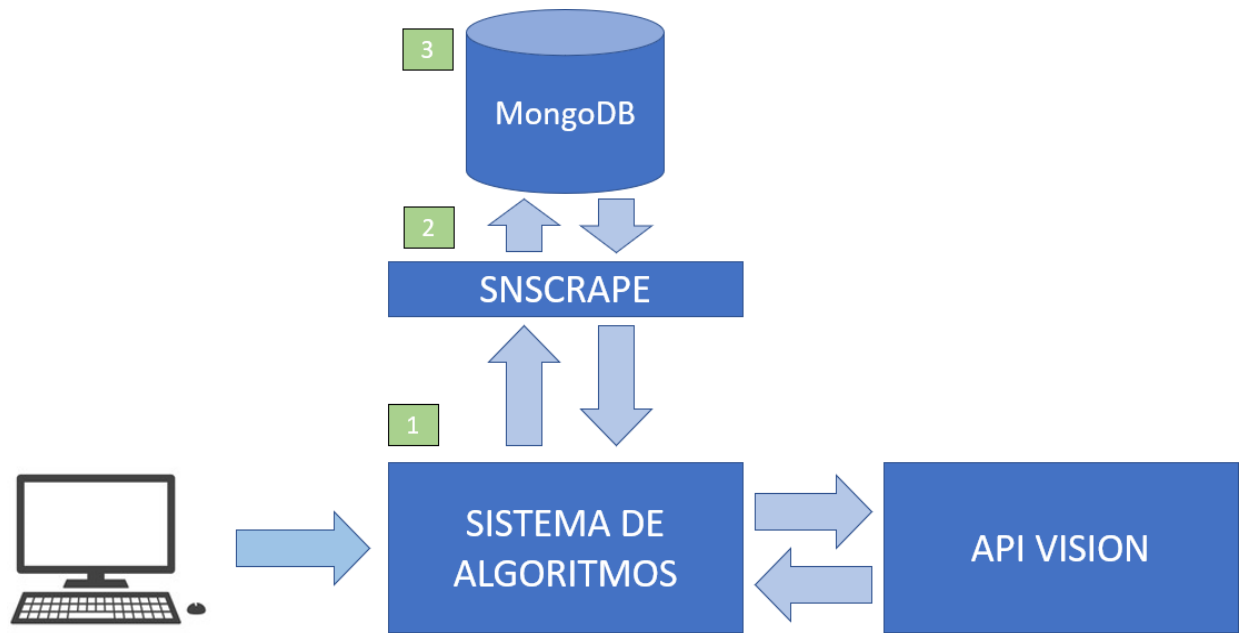


Figura 44 - Arquitectura general 2 (Edición propia)

En el caso de la segunda arquitectura, no será necesario explicar la parte referida a API Google Vision, puesto que ya fue introducida en la sección 3.2.1.1. En este apartado explicaremos: el modo en el que se empleó la librería SnsCrape, el modo en el que se almacenó en la base de datos de MongoDB y el algoritmo de detección de usuarios.

3.2.2.1 Obtención de las publicaciones del usuario con SnsCrape

La librería SnsCrape de Python se utiliza para extraer la información de Twitter que posteriormente será analizada por la API Google Vision. En este sentido, esta librería se fijando recuperar únicamente la información asociada al usuario seleccionado junto con la franja temporal que se considere.

Lo más representativo es la comprobación de la existencia de publicaciones de Twitter que incluyan imágenes o con un vídeo. Puntualizar, que al scrapear un vídeo nos quedaríamos con el primer fotograma del vídeo. Esto se ilustra en la Figura 45 - SnsCrape (Edición propia).

```
if tweet.media is not None:
    for medium in tweet.media:
        if isinstance(medium, snsCrape.modules.twitter.Photo):
            media_json2 = {
                "type": "photo",
                "fullUrl": medium.fullUrl
            }
            media_json.append(media_json2)
        if isinstance(medium, snsCrape.modules.twitter.Video):
            media_json2 = {
                "type": "video",
                "fullUrl": medium.thumbnailUrl
            }
            media_json.append(media_json2)
```

Figura 45 - SnsCrape (Edición propia)

3.2.2.2 Almacenamiento de los datos en la base de datos MongoDB

La base de datos de MongoDB será utilizada para almacenar la información obtenida a partir de Twitter. Dentro del archivo Python ‘MongoDB_DETECCIÓN_USER’ utilizaremos una serie de funciones que listaremos y explicaremos brevemente a continuación:

- `getUserTimeline`: esta función establece el espacio temporal en el que se van a extraer las imágenes de Twitter. Además, la función distingue entre aquellas publicaciones geolocalizadas y aquellas que no lo estén.
- `listarCarpetasUsers`: añade un directorio para cada usuario.
- `getConGeo`: define si la función esta geolocalizada o no lo está.
- `listarCarpetaImg`: da nombre a las imágenes.
- `CreateJSON`: crea un archivo JSON para almacenar la información relacionada con las publicaciones.
- `CreateDIR`: crea un nuevo directorio para almacenar usuarios e imágenes.
- `saveToDisk`: almacena las imágenes en el correspondiente directorio, y si no existe lo crea.
- `SaveMeditoDisk`: almacena la información anexa a las publicaciones en el directorio que debe hacerlo.
- `descargarImgsUser`: es la función encargada de descargar las imágenes, crea un JSON por cada imagen descargada. La información de dicho JSON será almacenada en la base de datos de MongoDB.
- `funcionDescarga`: llama a las diferentes funciones y las ejecuta en una misma función para conseguir el efecto deseado de descargar las imágenes con la información anexa, y almacenarla en los directorios con los nombres preestablecidos.

En la función `descargarImgsUser`, vemos cómo almacenamos la información a medida que scrapeamos los ‘*tweets*’, para ello utilizaremos el comando ‘`insert_one`’ (Figura 46 - Almacenamiento información MongoDB (Edición propia)).

```
tweet_json = createJSON(tweet)
for medium in tweet.media:
    #si tiene una instancia media de tipo photo entonces la guardo
    if isinstance(medium, snsrape.modules.twitter.Photo) or isinstance(medium, snsrape.modules.twitter.Video):
        print("There is a picture in this tweet")
        if bool == True:
            collection.insert_one(tweet_json) #almacenamiento info base de datos
        bool = False
```

Figura 46 - Almacenamiento información MongoDB (Edición propia)

Una vez definidas la funciones, establecemos conexión con la base de Datos.

```
if __name__ == '__main__':
    #try:
        # Abrimos conexión con servidor MongoDB local.
        client = MongoClient(host = ["localhost:27017"], serverSelectionTimeoutMS = 2000)
        client.server_info() # si no se realiza la conexión lanza una exception
        client.server_info()
```

Figura 47 - Base de Datos MongoDB

3.2.2.3 Algoritmo 4: Detección de usuarios

Este algoritmo usa la colección ‘User’ de base de datos de MongoDB. Dentro de esta base de datos se han almacenado las imágenes que han subido en un período de tiempo de usuarios seleccionados. Para facilitar el manejo, estas imágenes se descargarán en una carpeta con el nombre del usuario. A partir de estas fotografías, tratamos de desarrollar un algoritmo el cual nos dará las posibles

localizaciones donde se encontraba el usuario en ese período de tiempo. Para ello nos ayudaremos de los algoritmos que hemos detallado anteriormente, con algunas modificaciones en su código para permitir trabajar de manera conjunta.

En primer lugar, dada la ruta donde se ha descargado previamente todas las imágenes de un usuario en un espacio temporal determinado, es leer cada uno de los archivos que existen dentro de la misma a través de la librería `isfile`. Previamente, extraeremos el contenido con la función `os.listdir`.

```
contenido=os.listdir(ruta)
#Obtener archivos
archivos=[imagen for imagen in contenido if isfile(join(ruta, imagen))]
```

Figura 48 - Funciones `os.listdir` / `isfile` (Edición Propia)

Una vez hecho, declararemos la primera función `algoritmo1`, al que se le suministrarán tres listas:

- `TitleList`: la lista de los títulos asociados a las URLs.
- `LatList`: lista de las diferentes latitudes de las imágenes.
- `LongList`: lista de las diferentes longitudes de las imágenes.

Declararemos una lista la cual va a ser rellenada bien, con los títulos asociados a las URLs(`'Title'`),o bien, con la localización(`'locname'`) en el caso de tener latitud y longitud, dicha localización es extraída de los metadatos asociados a las URLs que proporciona la API Google Vision. Posteriormente, devolveremos esa `'lista1'`. Para recorrer las listas, utilizaremos un ciclo `for` (Figura 49 - Función `algoritmo1` (Edición propia)).

```
def algoritmo1(TitleList, LatList, LongList):

    Lista1=[] # declaramos la Lista 1
    for i, j, k in zip(TitleList, LatList, LongList):

        if not j: # no contienen Latitud
            Lista1.append(i)

        else: #contienen latitud y longitud
            Lat=j
            Long=k
            var = Lat, Long
            #Transformación de las coordenadas a dirección
            locname = loc.reverse(var)
            Lista1.append(locname)
    return Lista1 #devolvemos la Lista1
```

Figura 49 - Función `algoritmo1` (Edición propia)

En segundo lugar, la segunda función será denominada `algoritmo2`. A esta función se le definirán las mismas listas que al primer algoritmo, es decir, `'TitleList'`, `'LatList'` y `'LongList'` (explicadas anteriormente en el `algoritmo1`). Este algoritmo proporcionará las palabras que más se repitan dentro de los diferentes `'Title'` que son obtenidos a partir de las URLs que proporciona la API Google Vision. Estos `'Title'` son almacenados en una lista y, posteriormente, se comprueba cual es la palabra más repetida dentro de esos Títulos que tenga asociada una localización, esa será la palabra que devuelve la función.

Dentro de la función, declararemos la `'Lista2'` que será la que devolverá la función con las localizaciones asignadas por el algoritmo, y las cadenas de texto `'cadena'` y `'localización'` que serán explicadas posteriormente.

Para extraer la información de la lista `'TitleList'`, comenzaremos con un ciclo `for`. Almacenaremos el Título dentro de `'cadena'`. Previamente, eliminaremos los signos numéricos y de puntuación para evitar errores a la hora de obtener las localizaciones (Figura 50 - Función `algoritmo2` (Edición propia)).

```

for i in TitleList:

    if Title is None: #No contienen title
        continue

    else:
        string = i
        #eliminamos signos puntuacion
        salida = re.sub(r'^\w\s',' ',str(string))
        #eliminamos numeros
        newstring = ''.join([i for i in string if not i.isdigit()])
        new_string = re.sub(r"^[a-zA-Z0-9]", " ",newstring)
        #introducimos valor en una cadena de texto
        cadena += str(new_string) #eliminar signos de puntuación

```

Figura 50 - Función algoritmo2 (Edición propia)

Habiendo almacenado ya los Títulos dentro de la lista ‘cadena’, separaremos las palabras de ‘cadena’ usando la función ‘split’ y las introduciremos en la lista ‘listaPalabras’. Con las palabras separadas e introducidas en la lista, recorreremos con un ciclo for la lista ‘listaPalabras’, y a través de la función ‘count’ iremos contando las diferentes palabras y las almacenaremos en la lista ‘frecuenciaPalab’ (Figura 51 - Función algoritmo2 / Recuento de palabras (Edición propia)).

```

listaPalabras = cadena.split() #separamos las palabras de la cadena de texto

#creamos una lista para representar la frecuencia con la que se repiten las palabras
frecuenciaPalab = []

#recuento de las palabras que hay en la lista
for w in listaPalabras:
    frecuenciaPalab.append(listaPalabras.count(w))

```

Figura 51 - Función algoritmo2 / Recuento de palabras (Edición propia)

Posteriormente, con un ciclo for recorreremos las listas ‘frecuenciaPalab’ (número de veces que se ha repetido cada palabra) y ‘listaPalabras’ (Títulos separados por palabras). En el caso de que alguna palabra se repita más de una vez y dicha palabra no tiene una localización asociada, se descarta. Por el contrario, si tiene una localización asociada se almacenará en la cadena ‘localización’ (Figura 52 - Función algoritmo 2 / Selección palabras (Edición propia)).

```

for valor_a, valor_b in zip(frecuenciaPalab, listaPalabras): #obtenemos los valores en cada iteración
    if valor_a > 1: # si palabra se repite mas de dos veces

        if loc.geocode(valor_b) is None: # si la palabra repetida no tiene asignado una localizacion
            continue

        else: # si tiene una localización asociada
            localizacion += str(valor_b) # almacena la palabra
            localizacion += (' ')

    else:
        continue

```

Figura 52 - Función algoritmo 2 / Selección palabras (Edición propia)

Ya almacenada la información en la cadena ‘localización’, buscaremos la palabra más frecuente a través de la función ‘most_common’. En el caso de que el archivo ‘localización’ este vacío, debido a la falta de palabras coincidentes, proporcionaremos el primer ‘Title’ de la lista ‘TitleList’ (Figura 53 - Función algoritmo2 / Devolución Lista2 (Edición propia)).

```
if localizacion == True: #localización tiene contenido
    palabra = localizacion.split() #separamos las palabras
    contador = Counter(palabra) #contador
    palabra_frecuente = contador.most_common()[0][0] #palabra más repetida
    Lista2.append(palabra_frecuente)
    return Lista2 #devolvemos la lista2

else: #si localización esta vacío
    palabra_frecuente = TitleList[0]
    Lista2.append(palabra_frecuente)
    return Lista2 #devolvemos la lista2
```

Figura 53 - Función algoritmo2 / Devolución Lista2 (Edición propia)

Continuaremos con el tercer algoritmo denominado ‘algoritmo3’, en el que al comienzo describiremos una serie de listas las cuales son: ‘titles’, ‘coordLat’, ‘coordLong’, ‘score’, ‘coincLAT’, ‘coincLONG’, ‘coincSCORE’ y ‘Lista3’. A medida que describamos la función, indicaremos el uso de cada una de ellas.

Para el uso del tercer algoritmo, usaremos llamamos a una serie de listas parcialmente similares a la de los dos algoritmos anteriores. Salvo ‘ScoreList’, que indica la puntuación asociada a cada una de las localizaciones que realiza la API Google Vision.

Comenzaremos con un ciclo for, en el que se recorrerán las listas que han sido nombradas en el párrafo anterior. En el caso de que tenga título asociado, este se almacenará en la ‘Lista3’. En el caso de que no tenga coordenadas o score continuará con el ciclo, pero si los tuviera, se almacenarán en las listas ‘coordLat’, ‘coordLong’ y ‘score’, la latitud, la longitud y el score respectivamente (Figura 54 - Función algoritmo3 / Almacenamiento Información (Edición propia)).

```
if title is not None: #tienen título
    titles.append(title)

if not Lat: #No contienen Latitud
    continue

if not Score: #No tiene score
    continue

else: #latitud, longitud y/o score asociado
    #introducimos valor en una Lista
    Lat1=float('{0:.2f}'.format(Lat)) #eliminación de decimales
    Long1=float('{0:.2f}'.format(Long))
    coordLat.append(Lat1) #almacenamiento en listas
    coordLong.append(Long1)
    score.append(Score)
```

Figura 54 - Función algoritmo3 / Almacenamiento Información (Edición propia))

Una vez almacenada la información, buscaremos aquellas localizaciones cuyas coordenadas sean coincidentes. En el caso de que se repita alguna localización, esta se almacenará dentro de las listas ‘coincLAT’, ‘coincLONG’ y ‘coincSCORE’. Recordemos que el ‘algoritmo3’, presenta la localización que más veces se encuentra almacenada dentro de la lista (Figura 55 - Función algoritmo3 / Búsqueda Localizaciones Coincidentes (Edición propia)).

```

#Busqueda de localizaciones en la lista
if coordLat != False: #la lista coordLat tiene info almacenada
    for i, h, j in zip(range(len(coordLat)), coordLong, score):
        for j in range(len(coordLat)):
            if i != j:
                if coordLat[i] == coordLat[j]: #coincidencia
                    coincLAT.append(coordLat[i]) #almacenamiento info
                    coincLONG.append(coordLong[i])
                    coincSCORE.append(score[i])
                else:
                    continue
            else:
                continue

```

Figura 55 - Función algoritmo3 / Búsqueda Localizaciones Coincidentes (Edición propia)

En el caso de que haya más de una localización coincidente, se buscará la que tenga el mayor score asociado. Obtendremos la posición de dicha ubicación en la lista con la función ‘index’ y extraeremos de las listas ‘coincLAT’ y ‘coincLONG’ los valores de la latitud y longitud. Una vez tenemos esos datos, realizaremos la transformación de dichas variables a una localización textual. Dicha localización será almacenada en la ‘Lista3’, que devolverá el algoritmo.

Por el contrario, si no hay localizaciones coincidentes. Devolveremos la única localización asociada a la ubicación con más score, en caso de existir las coordenadas. En el caso de no existir, devolveremos el ‘Title’ con un mayor score (Figura 56 - Función algoritmo3 / Devolución lista (Edición propia)).

```

#Caso de que haya una localizacion coincidente
if coincSCORE == True:
    if len(coincSCORE) >= 1:
        maximo=max(coincSCORE) #obtenemos el valor máximo de la lista
        lugar = coincSCORE.index(maximo) #obtener posicion de dicho valor en la lista
        LAT1 = coincLAT[lugar]
        LONG1 = coincLONG[lugar]
        var = LAT1, LONG1
        locname = loc.reverse(var) #transformación de coordenadas a localización
        Lista3.append(locname)
        return Lista3 #devolvemos la lista3
    else:
        print("")

#Si no hay localizaciones coincidentes
else:
    for i,j,k,l in zip(TitleList, LatList, LongList, ScoreList): #recorremos las listas
        Lat = j
        Long = k
        Score = l
        title = i

        if Lat!='': #existe una latitud y una longitud asociada
            LAT2 = j
            LONG2 = k
            var2 = LAT2, LONG2
            locname2 = loc.reverse(var2) #transformación de coordenadas a localización
            Lista3.append(locname2)
            return Lista3 #devolvemos la lista3
            break

        else: #no existe una latitud y longitud asociada
            Lista3.append(title)
            return Lista3 #devolvemos la lista3
            break

```

Figura 56 - Función algoritmo3 / Devolución lista (Edición propia)

Una vez hemos definido las funciones, comenzaremos declarando las listas ‘TitleList’, ‘LatList’, ‘LongList’ y ‘ScoreList’, las cuales hemos utilizado anteriormente en las tres funciones.

Explicados estos pasos, el procedimiento consiste en, para cada imagen obtenida para un usuario en un espacio temporal mediante Snsrape y almacenada su información en la colección ‘User’, se procederá a analizar cada una a través de la API de Google Vision. El resultado será un fichero JSON para cada imagen.

Con este ciclo se cargarán todos los ficheros JSON obtenidos e iremos introduciendo dicha información en los algoritmos para obtener las localizaciones que serán devueltas en forma de listas (Figura 57 - Lectura JSON / Llamada funciones (Edición propia)).

```
for archivo in archivos:
    i=0
    if archivo[-1]=="n": #acabar en n

        with open(f'{ruta}/{archivo}') as file: #abrimos el archivo
            data = json.load(file)

            for foto in data['fotos']: #anlizamos los resultados del formato JSON
                Title = foto['title']
                Lat = foto['Latitud']
                Long = foto['Longitud']
                Score=foto['score']
                #Listas
                TitleList.append(Title) #introducimos en las listas
                LatList.append(Lat)
                LongList.append(Long)
                ScoreList.append(str(Score))

            #Funciones
            Lista1=algoritmo1(TitleList, LatList, LongList)
            Lista2=algoritmo2(TitleList, LatList, LongList)
            Lista3=algoritmo3(TitleList, LatList, LongList, ScoreList)

    else:
        continue
```

Figura 57 - Lectura JSON / Llamada funciones (Edición propia)

Una vez almacenada la información y devueltas las listas por los algoritmos con las diferentes localizaciones, desarrollaremos la parte del algoritmo que se encarga de obtener la localización final del algoritmo.

En primer lugar, declararemos una serie de listas:

- LAT: almacena la latitud.
- LONG: almacena la longitud.
- elementos_rep: almacenamiento de latitudes repetidas.
- elementos_rep1: almacenamiento de longitudes repetidas.

Con las listas ya declaradas, recorreremos las listas ‘Lista1’, ‘Lista2’ y ‘Lista3’. En el caso de tener una localización asociada que se puede transformar a ubicación, iremos almacenando la información en las listas ‘LAT’ y ‘LONG’. Anteriormente, eliminaremos decimales hasta quedarnos con dos (Figura 58 - Almacenamiento de información repetida (Edición propia)).

```
#Obtener localizaciones finales
for i in Listal:
    if loc.geocode(i) is None:
        continue
    else:
        getLoc=loc.geocode(i) # transformación a coordenadas
        Lat=getLoc.latitude
        Long=getLoc.longitude
        Lat1=float('{0:.2f}'.format(Lat)) # elimicación de decimales
        Long1=float('{0:.2f}'.format(Long))
        LAT.append(str(Lat1)) # almacenamiento en las listas
        LONG.append(str(Long1))
```

Figura 58 - Almacenamiento de información repetida (Edición propia)

Habiendo almacenado las coordenadas coincidentes en las listas, estudiaremos los diferentes casos. En el caso, de que no haya ninguna latitud y longitud asociada, el algoritmo no devolverá ningún valor. En el caso de que en la lista 'LAT' tan solo tenemos una Latitud, se presentará dicha latitud y longitud tras realizar una transformación de nuevo a una localización textual (Figura 59 - Existe solo una Ubicación (Edición propia))

```
elif len(LAT)<=1: #existe solo una ubicación
    latitud=LAT[0]
    longitud=LONG[0]
    variable= latitud, longitud
    LOC=loc.reverse(variable) #transformación de coordenadas a localización
    print("El usuario se encuentra en: \n", LOC)
```

Figura 59 - Existe solo una Ubicación (Edición propia)

En el caso de que exista más de una ubicación posibles, trataremos de buscar si existe más de una localización coincidente. A partir de la función 'most_common' buscamos las latitudes y longitudes que se repitan. Con la función anterior almacenaremos hasta tres variables coincidentes, es decir, las tres latitudes y longitudes que más se hayan repetido. A partir de aquí, dividiremos el sistema en función de que existan tres, dos, una o ninguna localización coincidente (Figura 60 - Asignación variables elementos repetidos (Edición propia)).

```
#Buscamos variables repetidas
counter = Counter(LAT)
if counter.most_common() is not None:
    elementos_rep = counter.most_common() #Elementos repetidos LAT
    counter = Counter(LONG)
    elementos_rep1 = counter.most_common() #Elementos repetidos LONG
    print(elementos_rep, elementos_rep1)
    #Lat
    first=elementos_rep[0]
    lat1=first[0]
    #Long
    first1=elementos_rep1[0]
    long1=first1[0]
```

Figura 60 - Asignación variables elementos repetidos (Edición propia)

Para los diferentes casos se calcularán las probabilidades de una ubicación en función de ser coincidente o no ser coincidente. Estos porcentajes se usarán para dar la probabilidad que tiene una determinada localización de que sea la correcta.

- len(LAT): número de ubicaciones que hay en la lista

- first[1], second[1], third[1]: el número de veces que se repitan las ubicaciones coincidentes dentro de la lista 'LAT'.

```
#probabilidades de que se repita un elemento
prob_nor=1/(len(LAT)) #elemento no repetido
prob_1=(first[1])/(len(LAT)) #elemento más repetido
prob_2=(second[1])/(len(LAT)) #segundo elemento más repetido
prob_3=(third[1])/(len(LAT)) #tercer elemento más repetido
```

Figura 61 - Probabilidades para tres o más elementos repetidos (Edición propia)

Hay que destacar que el cálculo de probabilidades tendrá que ser realizado en función de si no hay elementos repetidos y, también, si hay elementos repetidos en el caso de haber uno, dos, y hasta tres elementos repetidos, esto es realizado para limitar los resultados que arroja el algoritmo. Si lo anterior no es realizado, se producirán errores debido a que habrá variables que no existan. Realizado lo anterior, representaremos las variables con su probabilidad asociada.

Para sintetizar como funciona la representación de las localizaciones en las que existe alguna ubicación coincidente. Comenzaremos representando dicho valor coincidente en una variable. Posteriormente, utilizaremos un condicional if, si existe solo un valor coincidente trabajaremos con dicho valor. En el caso de que existan 2 o más volveremos de nuevo con otro if.

En dicho if(existen dos ubicaciones coincidentes), si solo existen dos ubicaciones coincidentes, almacenaremos los valores de Latitud y Longitud en dichas variables y procederemos a su representación. En el caso de que existan más de dos (máximo tres ubicaciones coincidentes), asignaremos ese tercer valor sus variables correspondientes y procederemos a la representación.

Para la representación, hay que destacar, que existe la posibilidad de que en la lista 'LAT' y 'LONG' puede haber más de una localización y no existir ninguna localización coincidente. En este caso se representará como en Figura 62 - Representación no localizaciones coincidentes (Edición propia).

```
elif len(LAT)<=1:
    latitud=LAT[0]
    longitud=LONG[0]
    variable= latitud, longitud
    LOC=loc.reverse(variable)
    print("El usuario se encuentra en: \n", LOC)
```

Figura 62 - Representación no localizaciones coincidentes (Edición propia)

Por el contrario, si existe una o dos localizaciones coincidentes se representarán tanto dichas ubicaciones como las que existen en las listas de 'LAT' y 'LONG' (Figura 63 - Representación 2 variables coincidentes (Edición propia))

```
#REPRESENTACION MAS DE UN ELEMENTO
VARIABLE = lat1, long1 #variable repetida 1
sitio=loc.reverse(VARIABLE)
print("Existe un " + str(prob_1) + "de que la localización sea: " + str(sitio))
VARIABLE2 = lat2, long2 #variable repetida 2
sitio2 = loc.reverse(VARIABLE2)
print("Existe un " + str(prob_2) + "de que la localización sea: " + str(sitio2))
if len(LAT) != False: #variables no repetidas
    for i, j in zip(LAT, LONG):
        VARIABLE3= i, j
        sitio3=loc.reverse(VARIABLE3)
        print("Existe un " + str(prob_nor) + "de que la localización sea: " + str(sitio3))
```

Figura 63 - Representación 2 variables coincidentes (Edición propia)

En la tesitura de que existan tres localizaciones coincidentes, solo se representarán estas tres. Aunque, en las listas ‘LAT’ y ‘LONG’ existan ciertos valores no coincidentes. Esto lo podemos ver en la Figura 64 - Representación tres elementos coincidentes (Edición propia)). En este caso, las ubicaciones repetidas no se representan.

```
#REPRESENTACION MAS DE UN ELEMENTO
VARIABLE = lat1, long1 #elemento coincidente más repetido
sitio=loc.reverse(VARIABLE)
print("Existe un " + str(prob_1) + "de que la localización sea: " + str(sitio))
VARIABLE2 = lat2, long2 #segundo elemento coincidente más repetido
sitio2 = loc.reverse(VARIABLE2)
print("Existe un " + str(prob_2) + "de que la localización sea: " + str(sitio2))
VARIABLE3 = lat3, long3 #tercer elemento coincidente más repetido
sitio3 = loc.reverse(VARIABLE3)
print("Existe un " + str(prob_3) + "de que la localización sea: " + str(sitio3))
```

Figura 64 - Representación tres elementos coincidentes (Edición propia)

Puntualizar, que es importante declarar las variables first, second o third, cada uno en su correspondiente if o else, puesto que si esto no es así se producirán errores, ya que esa variable no existiría.

En resumen, este algoritmo de Detección de Usuario se nutre de los algoritmos anteriores y nos presentará a través de porcentajes, las probabilidades de que el usuario este en un sitio u en otro en función de los resultados devueltos por las tres funciones. El código se representará en el anexo A2.

3.2.3 Arquitectura general nº3

Con el fin de mejorar la precisión de los resultados, se han implementado una serie de tratamientos previos sobre las imágenes. Estas mejoras permitirán a GeoApi mejorar los resultados obtenidos, puesto que se eliminarán de la imagen aquella información que no sea relevante para la API Google Vision a la hora de devolver sus resultados.

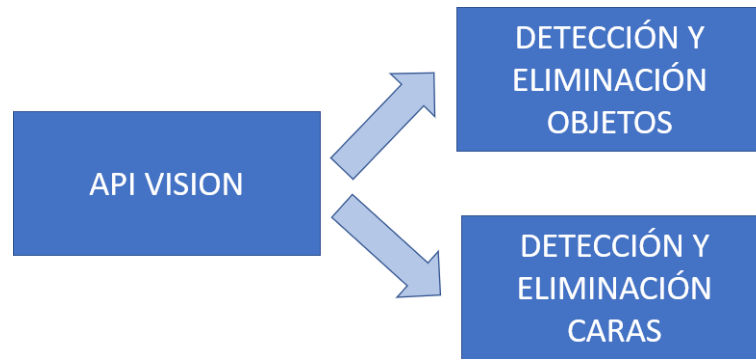


Figura 65 - Mejoras API Google Vision (Edición propia)

Como podemos observar en la Figura 65 - Mejoras API Google Vision (Edición propia) se ha establecido una mejora para la detección y eliminación de objetos en las fotografías, y otra para la detección y eliminación de caras. A continuación, indagaremos en cada una de ellas.

3.2.3.1 Detección y eliminación de objetos

Tratando de mejorar el uso que GeoApi hace de las imágenes, se ha desarrollado un paso previo en que se hace uso de la API de Google Vision. Concretamente, se hará uso de la función de detección y eliminación de objetos, una función adicional que permite a GeoApi detectar aquellos objetos que aparezcan en la imagen y que no sean relevantes para determinar su ubicación, y taparlos utilizando un fondo negro. De esta manera, GeoApi debiera tener más facilidades para realizar su labor.

Dada una imagen a procesar en modo lectura, extraemos la información y aplicamos las siguientes funciones para extraer la información de la imagen a partir de la API de Google Vision (Figura 66 - Extraer Información API Google (Edición propia))

```
image=vision.Image(content=content)
response=client.object_localization(image=image)
#response nos dará el nombre, score, bounding_poly

localized_object_annotations = response.localized_object_annotations
#localized_object_annotations nos dará el resultado
```

Figura 66 - Extraer Información API Google (Edición propia)

Una vez tenemos la información deseada, la almacenamos en una tabla a partir de la librería pandas. Para lo anterior utilizaremos un ciclo for que irá recorriendo cada uno de los objetos localizados (Figura 67 - Almacenamiento Información librería 'pandas' (Edición propia)).

```
df = pd.DataFrame(columns=['name', 'score']) #2 columnas: 1 nombre / 2 score asociado
for obj in localized_object_annotations: #ciclo for para cada objeto localizado
    df = df.append( #añade la información en modo diccionario
        dict(name= obj.name,
              score= obj.score),
        ignore_index=True
    )
```

Figura 67 - Almacenamiento Información librería 'pandas' (Edición propia)

Posteriormente, declararemos dos imágenes:

- Pillow_image: presenta la imagen donde están los objetos localizados. Estos objetos se encuentran dentro de un recuadro.

- `img_del`: imagen donde dichos objetos han sido eliminados. Es la imagen con la que finalmente nos quedaremos para su posterior análisis.

Proseguiremos con un nuevo ciclo `for` que irá recorriendo cada uno de los objetos localizados y llamará a las funciones ‘`draw_borders`’ y ‘`remove_borders`’ (Figura 68 - Ciclo `for` / Funciones (Edición propia))

```
for obj in localized_object_annotations:
    r, g, b= 0, 0, 0 #color negro
    pillow_image = draw_borders(pillow_image, obj.bounding_poly, (r, g, b), pillow_image.size, obj.name, obj.score)
    img_del = remove_borders(pillow_image, obj.bounding_poly, (r, g, b), pillow_image.size, obj.name, obj.score)
```

Figura 68 - Ciclo `for` / Funciones (Edición propia)

La primera función es denominada ‘`draw_borders`’ (Figura 69- Función `draw_borders` (Edición propia) con ella recuadraremos los objetos que aparecen en la imagen. Para su uso necesitamos introducir la siguiente información:

- `pillow_image`: imagen de la que se extrae la localización de los objetos.
- `obj.bounding_poly`: coordenadas de los recuadros que contienen un objeto en su interior.
- `pillow_image.size`: tamaño de la imagen.
- `obj.name`: nombre del objeto detectado.
- `obj.score`: confianza que tiene la API de que ese objeto sea el que está indicando.

```
def draw_borders(pillow_image, bounding, color, image_size, caption='', confidence_score=0):

    width, height = image_size #establecemos anchura y altura
    draw = ImageDraw.Draw(pillow_image) #permite trabajar sobre la imagen
    draw.polygon([ #dibujamos el poligono
        bounding.normalized_vertices[0].x *
        width, bounding.normalized_vertices[0].y*height,
        bounding.normalized_vertices[1].x *
        width, bounding.normalized_vertices[1].y*height,
        bounding.normalized_vertices[2].x *
        width, bounding.normalized_vertices[2].y*height,
        bounding.normalized_vertices[3].x *
        width, bounding.normalized_vertices[3].y*height], fill=None, outline=color)

    #formato de nuestra letra
    font_size = width * height // 22000 if width * height > 400000 else 12
    font = ImageFont.load_default()

    #formato de txtto
    draw.text((bounding.normalized_vertices[0].x * width,
        bounding.normalized_vertices[0].y * height), font=font, text=caption, fill=color)

    #introducimos confidence score
    draw.text((bounding.normalized_vertices[0].x * width,
        bounding.normalized_vertices[0].y* height + 20), font=font, text='Confidence Score {0:.2f}%'.format(confidence_score))

    #devolvemos imagen modificada
    return pillow_image
```

Figura 69- Función `draw_borders` (Edición propia)

Dentro de la función, comenzaremos definiendo la anchura y la altura de la imagen. Posteriormente, dibujaremos un polígono alrededor del objeto localizado, y presentaremos el nombre y el score. A través de ‘`font_size`’ estableceremos el tamaño de letra requerido. Por último, devolveremos la imagen a través de ‘`pillow_image`’.

La segunda función es la denominada ‘`remove_borders`’. Con esta función cubriremos de negro los recuadros que han sido representados anteriormente en la función ‘`draw_borders`’. Para su correcto funcionamiento estaremos introduciendo los mismos campos que utilizábamos en nuestra función ‘`draw_borders`’. Salvo la imagen que ahora será denominada ‘`img_del`’, pero funcionalmente será la misma imagen que en el caso anterior.

Los cambios con respecto a la función ‘draw_borders’, están en que tan solo será necesario declarar las dimensiones de la imagen, representar el polígono y rellenar dicha figura de color negro en lugar de dejarlo vacío.

También, para evitar que nuestra eliminación de objetos nos rellene de negro todos los recuadros, pondremos una condición para aquellas en las que el nombre(‘content’) de los objetos identificados interfiera con el nombre de ciertos monumentos o edificios de interés (Figura 70 - Función remove_borders (Edición propia)).

```
def remove_borders(pillow_image, bounding, color, image_size, caption='', confidence_score=0):
    width, height = image_size #definimos las dimensiones
    draw = ImageDraw.Draw(img_del)
    if caption == 'Animal' or caption == 'Building' or caption == 'Monument' or caption == 'Statue' or caption == 'Window':
        print('')
    else:
        draw.polygon([ #representamos el polígono
            bounding.normalized_vertices[0].x *
            width, bounding.normalized_vertices[0].y*height,
            bounding.normalized_vertices[1].x *
            width, bounding.normalized_vertices[1].y*height,
            bounding.normalized_vertices[2].x *
            width, bounding.normalized_vertices[2].y*height,
            bounding.normalized_vertices[3].x *
            width, bounding.normalized_vertices[3].y*height], fill=color, outline=color)
    return img_del #devolvemos la imagen
```

Figura 70 - Función remove_borders (Edición propia)

Una vez la función nos haya devuelto la imagen que utilizaremos ‘img_del’, guardamos dicha imagen a través de la función ‘save’ y procedemos a abrirla para su posterior análisis. La imagen será guardada en el mismo archivo donde se encontraba la fotografía original.

La principal diferencia entre ambas funciones es que la función ‘draw_borders’ recuadra los objetos y la función ‘remove_borders’ cubre de negro esos mismos objetos, salvo que hayamos especificado que ese tipo de objeto no sea modificado.

3.2.3.2 Detección y eliminación de caras

Utilizando de nuevo las funciones de la API Google Vision, implementaremos la función de eliminar rostros. Esta función será muy útil, puesto que una gran parte de las imágenes que los usuarios publican en las redes se corresponden con autorretratos. En este caso, hemos hecho uso de la función detectar caras de la API Google Vision.

Dada una imagen a procesar, la abrimos en modo lectura, extraemos la información y aplicamos las siguientes funciones para extraer la información de la imagen a partir de la API de Google Vision). En este sentido, la imagen podría ser una que ya ha sido modificada previamente después de haber eliminado objetos considerados como no relevantes. Una vez extraída la información, utilizaremos el método de detección de caras de la API Google Vision para obtener los rostros presentes en la imagen (Figura 71 - Detección Caras (Edición propia)).

```
#Create our image object/ content is the binary object
image = vision.Image (content=content)
#Face detection method
response = client.face_detection (image=image)
```

Figura 71 - Detección Caras (Edición propia)

La variable ‘response’ nos muestra las diferentes facciones de la cara, así como las coordenadas que encuadran el rostro. Posteriormente, con la función ‘response.face_annotations’ obtendremos los estados de ánimo que estima la API Google Vision en función del análisis del rostro.

Definidas las funcionalidades de la API, abrimos la imagen que deseamos modificar y con un ciclo for recorremos la variable ‘faceAnnotation’ (Figura 72 - faceAnnotation / Abrir Imagen / Ciclo for (Edición propia))

```
faceAnnotation = response.face_annotations
#abrimos la imagen
face_image = Image.open(image_path)

for face in faceAnnotation:
    r, g, b= 0, 0, 0 #color negro
    #llamada a la función
    face_image = remove_faces(face_image, face.bounding_poly.vertices, (0, 0, 0), face_image.size)
```

Figura 72 - faceAnnotation / Abrir Imagen / Ciclo for (Edición propia)

La función ‘remove_faces’, será la encargada de eliminar las caras. A continuación, introduciremos los parámetros de la misma. Estos son:

- face_image: la imagen que hemos seleccionado.
- Face.bounding_poly.vertices: los vértices que encuadran los rostros que queremos eliminar.
- face_image.size: dimensiones de la imagen.

Dentro de la función, definimos las dimensiones de la imagen. Posteriormente, con la función ‘draw’ pintaremos el polígono que recuadrará los rostros. Para pintar el polígono utilizaremos los 4 vértices; con sus coordenadas x e y, que serán extraídas de ‘Face.bounding_poly.vertices’. Una vez hemos recuadrado la imagen, con ‘fill’ rellenaremos dicho polígono de color negro. Con los rostros ya suprimidos devolveremos la imagen (Figura 73 - Función remove_faces (Edición propia)).

```
#Función borrar caras
def remove_faces(face_image, vertex, color, image_size):
    width, height = image_size #ancho, altura de la imagen
    draw = ImageDraw.Draw(face_image) #dibujamos el polígono

    draw.polygon([
        vertex[0].x, vertex[0].y,
        vertex[1].x, vertex[1].y,
        vertex[2].x, vertex[2].y,
        vertex[3].x, vertex[3].y], fill=color, outline=color)

    return face_image #devolvemos la imagen
```

Figura 73 - Función remove_faces (Edición propia)

Fuera de la función, devuelta ya la imagen, procederemos a mostrar y a guardar dicha imagen ya modificada para su posterior uso en la detección y eliminación de objetos (Figura 74 - Mostar / Guardar la imagen (Edición propia)).

```
pillow_image= Image.open(image_path) #mostramos la imagen
img_del= Image.open(image_path) #guardamos la imagen
```

Figura 74 - Mostar / Guardar la imagen (Edición propia)

4 RESULTADOS

Para la realización de la validación de los resultados obtenidos de la aplicación del sistema GeoApi, será necesario crear previamente un conjunto de datos que servirán para las pruebas. Por tanto, el primer paso será definir el modo de adquisición de estas fotos de prueba.

4.1 Datos de prueba

A este respecto, se ha decidido centrar la búsqueda en imágenes tomadas en cuatro ciudades diferentes. Estas son: Pontevedra, Santiago, Madrid y Santander. Para ello, enfocaremos el problema de recolectar las imágenes de dos formas diferentes.

En primer lugar, buscaremos en Twitter mensajes que cumplan tres condiciones: que posean en su contenido unas palabras claves concretas, que dispongan de fotos y que estén geolocalizados en las ciudades indicadas. En segundo lugar, tendrán que cumplir las dos primeras condiciones anteriores, pero no la tercera. A este respecto, para cada ciudad definiremos unas coordenadas y un radio. Por último, fijamos una fecha de referencia a partir de la cual comenzaremos la búsqueda.

```
for search_santiago in searches_santiago:
    #crearColeccion("Santiago", locSantiago, search_santiago, since, maxTweets,True)
    crearColeccion("Santiago_SinGeo", locSantiago, search_santiago, since, maxTweets, False)
for search_madrid in searches_madrid:
    #crearColeccion("Madrid", locMadrid, search_madrid, since, maxTweets,True)
    crearColeccion("Madrid_SinGeo", locMadrid, search_madrid, since, maxTweets,False)
for search_santander in searches_santander:
    #crearColeccion("Santander", locSantander, search_santander, since, maxTweets,True)
    crearColeccion("Santander_SinGeo", locSantander, search_santander, since, maxTweets, False)
for search_pontevedra in searches_pontevedra:
    #crearColeccion("Pontevedra", locPontevedra, search_pontevedra, since, maxTweets,True)
    crearColeccion("Pontevedra_SinGeo", locPontevedra, search_pontevedra, since, maxTweets, False)
```

Figura 75 - Consulta imágenes con geo/sin geo (Edición propia)

Toda la información relativa a los mensajes obtenidos se almacena en la base de datos de MongoDB, que han sido explicadas en 3.1.6, y a su vez para evitar el acceso repetitivo a ella, se descargarán las imágenes asociadas, almacenando en su nombre el identificador del Tweet junto con la fecha y hora de la imagen. Resaltar, que antes del uso de las imágenes, será necesario someter a estas a un cribado, para desechar aquellas que no sean adecuadas.

Marcaremos el objetivo de analizar 40 imágenes por ciudad, como veremos posteriormente, esto no se podrá llevar a cabo en algunas ciudades.

Una vez hayamos realizado este proceso y estén clasificadas en su carpeta correspondiente según en la ciudad en la que hayan sido tomadas, y; además, estén nombradas con su formato fecha-hora correspondiente, pasaremos a realizar dos estudios por separado.

- Con geolocalización: analizaremos las imágenes; que han sido extraídas de Twitter, con una localización asociada.
- Sin geolocalización: analizaremos las imágenes; que han sido extraídas de Twitter, con una localización asociada.

CIUDAD	Nº IMÁGENES CON GEO	Nº IMÁGENES SIN GEO
MADRID	40	40
PONTEVEDRA	16	40
SANTANDER	20	40
SANTIAGO	11	40

Tabla 1 - Nº imágenes válidas por ciudad con/sin geo (Edición propia)

Podemos ver que dentro de la Tabla 1 - Nº imágenes válidas por ciudad con/sin geo (Edición propia), el número de imágenes sin localizar es mayor que el número de imágenes localizadas. Lo anterior, se debe a una disminución en la tendencia de los usuarios a localizar sus imágenes.

En segundo lugar, para el uso del algoritmo de Detección de Usuarios, se recurrirá a un conjunto de imágenes procedentes de un solo usuario, dichas imágenes se almacenarán en una segunda base de datos de MongoDB. Esta segunda base de datos selecciona los usuarios que sus imágenes ya fueron descargadas para la primera base de datos y descarga imágenes en un período de tiempo determinado, con un máximo número de tweets y con un concepto determinado.

```
# Seleccionamos la base de datos existente para guardar tweets
db = client['TweetsUser']
ciudad_santander="Santander" #nombre de las ciudades
ciudad_pontevedra = "Pontevedra"
ciudad_madrid = "Madrid"
ciudad_santiago = "Santiago"
congeo1 = True #Existe geo
congeo2 = False #Esta vacía
temporalidad=2 #2 días
maxTweets=4 #número máximo de tweets
```

Figura 76 - Criterios de búsqueda imágenes Detección de usuario (Edición propia)

Como en los casos anteriores, antes de comenzar con las pruebas para la detección de las localizaciones, es preciso realizar un cribado en dichos usuarios cuyas imágenes no sean adecuadas. Por ejemplo, podemos encontrar muchas imágenes que son caricaturas, juegos, alimentos, etc. Por ese motivo descartaremos aquellos usuarios que no sean adecuadas por el contenido de sus imágenes o aquellas que tan solo tengan una imagen. También, resaltar que de aquellos usuarios que seleccionemos, dentro de su carpeta descartaremos algunas imágenes cuyo contenido no sea acorde.

Como objetivo, marcamos un máximo de diez usuarios por ciudad buscando analizar un total de entre 6 y 8 imágenes. Pero, como veremos a continuación en algunas ciudades no se cumple el requisito de que haya 10 usuarios que reúnan las características deseadas.

Finalmente, partiendo de los mensajes y sus imágenes descargadas de la forma comentada, se extraerá la información relativa a los usuarios creadores de esos mensajes en un espacio temporal delimitado, permitiendo realizar un seguimiento temporal. A continuación, presentaremos una tabla con el número de usuarios por ciudad, de los cuales se puede llegar a extraer la información.

CIUDAD	Nº USUARIO	Nº IMÁGENES
MADRID	20	66
PONTEVEDRA	3	7
SANTANDER	7	18
SANTIAGO	5	12

Tabla 2 - Nº Usuarios e Imágenes por ciudad (Edición propia)

4.2 Pruebas realizadas

En un primer momento utilizaremos como entradas del sistema descrito cada una de las imágenes que conforma el conjunto de imágenes con geolocalización y sin geolocalización, y se analizarán los resultados obtenidos por los tres algoritmos descritos.

En un segundo momento, utilizaremos como entradas del sistema descrito el conjunto de imágenes capturadas para el análisis de los usuarios y se analizarán los resultados obtenidos por el algoritmo de detección de usuarios.

Finalmente, utilizaremos 3.2.3 y efectuaremos las pruebas acerca del funcionamiento de las operaciones de tratamiento de imágenes.

4.2.1 Resultados obtenidos sobre imágenes con geolocalización

A continuación, se plasman los resultados en un Excel que almacena información tal como: el nombre del usuario, una descripción de la imagen, resultado de los algoritmos y media de los algoritmos de forma conjunta. De un modo más detallado, las columnas que la componen son:

- Numero: es el número de la imagen en la tabla.
- ALG1, ALG2, ALG3: son los nombres de los tres algoritmos. Recordando lo visto en la descripción de algoritmos, el primero de ellos es el que busca el mayor score, el segundo trata de conseguir la palabra que más se repite dentro del 'title' asociado a las URLs que proporciona la API Google Vision y el tercero trata de obtener aquellas coordenadas que se repiten más, dentro de las ubicaciones que presenta la API Google Vision en base a la imagen a analizar. A cada columna se le asigna un valor determinado:
 - 1: Localización correcta.
 - 0,5: Localización en zona cercana (misma provincia)
 - 0: Localización no encontrada o errónea.
- Porcentaje: media de los valores asignados en las columnas ALG1, ALG2 Y ALG3 para una determinada imagen.
- Usuario: el nombre del usuario al que pertenecen las o la imagen. Esta columna no se representará por razones de confidencialidad.
- Descripción: un breve resumen de lo que es la imagen, realizado de manera manual.
- Ciudad: donde ha sido tomada la imagen.
- Localizado: si la API Google Vision ha conseguido obtener información de la imagen o no. En el caso de que no, se remarcará en la casilla con un NO y se cubrirá la casilla de rojo.

Como remarcamos anteriormente, en esta parte se analizarán aquellas imágenes obtenidas de Twitter que tienen asociada una localización. Resaltar que se han seleccionado un total de 88 imágenes. A continuación, mostraremos una tabla con los primeros sesenta resultados obtenidos:

NUMERO	ALG1	ALG2	ALG3	PORCENTAJE	USUARIO	DESCRIPCIÓN	CIUDAD	LOCALIZADO
1	1	0	1	0,6666667		Imagen el retiro desenfocada	MADRID	
2	0	0	0	0		Estatuas puerta del sol	MADRID	
3	1	0	1	0,6666667		Imagen el retiro desenfocada	MADRID	
4	1	0	1	0,6666667		Palacio de Cristal	MADRID	
5	0	0	0	0		Imagen concierto	MADRID	NO
6	0	0	0	0		Calle Madrid	MADRID	NO
7	0	0	0	0		Calle Madrid	MADRID	
8	1	0	1	0,6666667		Estadio Santiago Bernabeu	MADRID	
9	0	0	0	0		Ayuntamiento Madrid Noche	MADRID	
10	0	1	0	0,3333333		Edificio Metropolis	MADRID	
11	0	0	0	0		Palacio real	MADRID	
12	0	0	0	0		Palacio real	MADRID	
13	0	0	0	0		Estatua	MADRID	NO
14	1	0	1	0,6666667		Palacio de Cristal	MADRID	
15	1	0	1	0,6666667		Fachada catedral	MADRID	
16	1	1	1	1		Plaza mayor	MADRID	
17	1	0	1	0,6666667		Santiago Bernabeu	MADRID	
18	0	0	0	0		Puerta del sol	MADRID	
19	1	0	1	0,6666667		Lago Retiro	MADRID	
20	1	0	1	0,6666667		Estatua Plaza	MADRID	
21	0	0	0	0		Hotel Paza España	MADRID	NO
22	1	1	1	1		Gran Vía	MADRID	
23	1	0	1	0,6666667		Templo De Bod	MADRID	
24	1	0	0	0,3333333		Edificio	MADRID	
25	1	0	1	0,6666667		Escultura	MADRID	
26	1	0	1	0,6666667		Puerta del sol	MADRID	
27	1	0	0	0,3333333		Estación Tren	MADRID	
28	0	0	0	0		Calle Madrid	MADRID	NO
29	1	0	1	0,6666667		Calle Gran Vía	MADRID	
30	1	0	1	0,6666667		Calle Gran Vía	MADRID	
31	0	0	0	0		Calle Madrid	MADRID	
32	1	1	1	1		Plaza Mayor	MADRID	
33	1	0	0	0,3333333		Palacio Cristal	MADRID	
34	1	0	1	0,6666667		Plaza Mayor	MADRID	
35	1	1	1	1		Plaza Mayor	MADRID	
36	1	0	0	0,3333333		Parada Metro	MADRID	
37	1	0	1	0,6666667		Placio Cristal	MADRID	
38	0	0	0	0		Iglesia	MADRID	NO
39	1	0	1	0,6666667		Ayuntamiento Madrid	MADRID	
40	0	0	0	0		Torres kyo	MADRID	NO
41	0	0	0	0		Concierto	PONTEVEDRA	
42	0,5	0	0,5	0,3333333		Iglesia de la peregrina	PONTEVEDRA	
43	0	0	0	0		Carrera, calle Pontevedra	PONTEVEDRA	
44	1	0	1	0,6666667		Carrera, calle Pontevedra	PONTEVEDRA	
45	0	0	0	0		Calle Pontevedra	PONTEVEDRA	NO
46	1	0	0	0,3333333		Plaza Toros Pontevedra	PONTEVEDRA	
47	0,5	0	0,5	0,3333333		Ilgesia Peregrina	PONTEVEDRA	
48	0	0	0	0		Estatua Valle Inclán	PONTEVEDRA	NO
49	1	0	1	0,6666667		Estatua Loro	PONTEVEDRA	
50	0	0	0	0		Concierto Pontevedra	PONTEVEDRA	NO
51	0	0	0	0		Calle Pontevedra	PONTEVEDRA	
52	0,5	0	1	0,5		Calle Pontevedra	PONTEVEDRA	
53	0	0	0	0		Concierto Pontevedra	PONTEVEDRA	
54	0	0	0	0		Plaza Pontevedra	PONTEVEDRA	NO
55	0	0	0	0		Gente paseando Pontevedra	PONTEVEDRA	
56	0,5	1	1	0,8333333		Estatua Loro Pontevedra	PONTEVEDRA	
57	0	0	0	0		Playa Sardinero	SANTANDER	NO
58	0	0	0	0		Playa Sardinero	SANTANDER	NO
59	0	0	0	0		Ayuuntamiento Santander	SANTANDER	
60	1	0	1	0,6666667		Hotel bahía	SANTANDER	

Tabla 3 - Resultados de Imágenes con Geolocalización (Edición propia)

A continuación, mostraremos una tabla con los resultados obtenidos en base a Tabla 3 - Resultados de Imágenes con Geolocalización (Edición propia):

ALGORITMO	PRECISIÓN
IMÁGENES NO LOCALIZADAS	23%
ALGORITMO 1	53%
ALGORITMO 2	11%
ALGORITMO 3	45%
TOTAL	0,37%

Tabla 4 - Resultado Final Imágenes con Geolocalización (Edición propia)

De Tabla 3 - Resultados de Imágenes con Geolocalización (Edición propia) podemos ver que aproximadamente en un 23% de las imágenes no se puede extraer información por parte de la API. Muchas de estas imágenes se corresponden con entornos que son bastante complicados encontrar diferencias entre ellos, como pueden ser playas, calles o imágenes donde hay una gran cantidad de personas u objetos.

También vemos que el algoritmo el cual ha obtenido un mejor resultado es el Algoritmo 1, puesto que es el que selecciona la información con un mayor score (escoge la que más posibilidades tiene de ser correcta).

Tras este, se encuentra el tercer algoritmo. Su precisión en parte se basa en ese uso de aquellas coordenadas que sean comunes y en el caso de no haber ninguna localización coincidente trabajaría como el primer algoritmo mostrando la localización con el score más alto.

En el caso del segundo algoritmo, este ha obtenido una menor precisión puesto que en muchas ocasiones estas palabras que se repiten en gran parte son palabras denominadas como stopwords, es decir, preposiciones, determinantes o artículos. En definitiva, elementos que no aportan información relevante al texto.

4.2.2 Resultados obtenidos sobre imágenes sin geolocalización

A continuación, se presentan los resultados en un Excel que almacena información tal como: el nombre del usuario, una descripción de la imagen, resultado de los algoritmos y media de los algoritmos de forma conjunta. Las columnas utilizadas en nuestra tabla son las mismas que en el apartado anterior.

Para la realización de las pruebas, hemos analizado un total de 40 imágenes por ciudad, constituyendo un total de 160 imágenes. Hay que recalcar, que al igual que en las imágenes con geolocalización, las imágenes también han sido obtenidas de Twitter. A continuación, presentaremos una tabla con las 60 primeras imágenes de la tabla.

NUMERO	ALG1	ALG 2	ALG3	PORCENTAJE	USUARIO	DESCRIPCIÓN	CIUDAD	LOCALIZADO
1	1	0	1	0,6666667		Imagen el retiro desenfocada	MADRID	
2	0	0	0	0		Estatuas puerta del sol	MADRID	
3	1	0	1	0,6666667		Imagen el retiro desenfocada	MADRID	
4	1	0	1	0,6666667		Palacio de Cristal	MADRID	
5	0	0	0	0		Imagen concierto	MADRID	NO
6	0	0	0	0		Calle Madrid	MADRID	NO
7	0	0	0	0		Calle Madrid	MADRID	
8	1	0	1	0,6666667		Estadio Santiago Bernabeu	MADRID	
9	0	0	0	0		Ayuntamiento Madrid Noche	MADRID	
10	0	1	0	0,3333333		Edificio Metropolis	MADRID	
11	0	0	0	0		Palacio real	MADRID	
12	0	0	0	0		Palacio real	MADRID	
13	0	0	0	0		Estatua	MADRID	NO
14	1	0	1	0,6666667		Palacio de Cristal	MADRID	
15	1	0	1	0,6666667		Fachada catedral	MADRID	
16	1	1	1	1		Plaza mayor	MADRID	
17	1	0	1	0,6666667		Santiago Bernabeu	MADRID	
18	0	0	0	0		Puerta del sol	MADRID	
19	1	0	1	0,6666667		Lago Retiro	MADRID	
20	1	0	1	0,6666667		Estatua Plaza	MADRID	
21	0	0	0	0		Hotel Paza España	MADRID	NO
22	1	1	1	1		Gran Vía	MADRID	
23	1	0	1	0,6666667		Templo De Bod	MADRID	
24	1	0	0	0,3333333		Edificio	MADRID	
25	1	0	1	0,6666667		Escultura	MADRID	
26	1	0	1	0,6666667		Puerta del sol	MADRID	
27	1	0	0	0,3333333		Estación Tren	MADRID	
28	0	0	0	0		Calle Madrid	MADRID	NO
29	1	0	1	0,6666667		Calle Gran Vía	MADRID	
30	1	0	1	0,6666667		Calle Gran Vía	MADRID	
31	0	0	0	0		Calle Madrid	MADRID	
32	1	1	1	1		Plaza Mayor	MADRID	
33	1	0	0	0,3333333		Palacio Cristal	MADRID	
34	1	0	1	0,6666667		Plaza Mayor	MADRID	
35	1	1	1	1		Plaza Mayor	MADRID	
36	1	0	0	0,3333333		Parada Metro	MADRID	
37	1	0	1	0,6666667		Placio Cristal	MADRID	
38	0	0	0	0		Iglesia	MADRID	NO
39	1	0	1	0,6666667		Ayuntamiento Madrid	MADRID	
40	0	0	0	0		Torres kyo	MADRID	NO
41	0	0	0	0		Concierto	PONTEVEDRA	
42	0,5	0	0,5	0,3333333		Iglesia de la peregrina	PONTEVEDRA	
43	0	0	0	0		Carrera, calle Pontevedra	PONTEVEDRA	
44	1	0	1	0,6666667		Carrera, calle Pontevedra	PONTEVEDRA	
45	0	0	0	0		Calle Pontevedra	PONTEVEDRA	NO
46	1	0	0	0,3333333		Plaza Toros Pontevedra	PONTEVEDRA	
47	0,5	0	0,5	0,3333333		Illesia Peregrina	PONTEVEDRA	
48	0	0	0	0		Estatua Valle Inclán	PONTEVEDRA	NO
49	1	0	1	0,6666667		Estatua Loro	PONTEVEDRA	
50	0	0	0	0		Concierto Pontevedra	PONTEVEDRA	NO
51	0	0	0	0		Calle Pontevedra	PONTEVEDRA	
52	0,5	0	1	0,5		Calle Pontevedra	PONTEVEDRA	
53	0	0	0	0		Concierto Pontevedra	PONTEVEDRA	
54	0	0	0	0		Plaza Pontevedra	PONTEVEDRA	NO
55	0	0	0	0		Gente paseando Pontevedra	PONTEVEDRA	
56	0,5	1	1	0,8333333		Estatua Loro Pontevedra	PONTEVEDRA	
57	0	0	0	0		Playa Sardinero	SANTANDER	NO
58	0	0	0	0		Playa Sardinero	SANTANDER	NO
59	0	0	0	0		Ayuuntamiento Santander	SANTANDER	
60	1	0	1	0,6666667		Hotel bahía	SANTANDER	

Tabla 5 - Resultado de imágenes sin Geolocalización (Edición propia)

Analizando la Tabla 5 - Resultado de imágenes sin Geolocalización (Edición propia), se presenta la tabla con las conclusiones de la precisión de los algoritmos y aquellas imágenes de las que se ha podido extraer información a un formato JSON.

ALGORITMO	PRECISIÓN
IMÁGENES NO LOCALIZADAS	28%
ALGORITMO 1	48%
ALGORITMO 2	4%
ALGORITMO 3	42%
TOTAL	31%

Tabla 6 - Resultado Final de Imágenes sin Geolocalización (Edición propia)

En primer lugar, cabe destacar que el número de imágenes de las que no se ha podido obtener su información y almacenarla en un formato JSON ha aumentado a 41 imágenes, pero también ha aumentado el número de imágenes analizadas a 160. Es decir, que aproximadamente de 1 de cada 4 imágenes no se ha podido extraer información. Posiblemente, debido a la falta de geolocalización que hace que las imágenes se correspondan con sitios menos comunes de las ciudades. Estas imágenes que la API Google Vision no es capaz de localizar, normalmente, se corresponden con: puentes, viviendas, zonas verdes como pueden ser bosques e imágenes del tráfico. Estas son las zonas donde se ha detectado que la API Google Vision tiene más debilidades a la hora de obtener información

En lo referido a los algoritmos, el primer algoritmo mantiene una buena precisión de búsqueda del 48% haciendo que la mitad de las imágenes analizadas sean posibles de localizar. En parte, el buen funcionamiento de este algoritmo se basa; como recalcamos anteriormente, en la selección del score más grande. Hay que recalcar que la API Google Vision devuelve los resultados de forma ordenada, es decir, primero devuelve los resultados con un score más alto. De lo anterior, se puede obtener la relación que existe entre imágenes con un mayor score asociado y una mejor obtención de la localización.

En cuanto al segundo algoritmo, obtiene precisiones muy bajas, en concreto solo ha sido capaz de localizar seis imágenes. Probablemente, la existencia de ubicaciones más remotas, hacen que los resultados de búsquedas de páginas web sean menores, consecuentemente los metadatos asociados a dichas imágenes también lo son. Por lo que este algoritmo; que funciona con la palabra más frecuente de los títulos, es poco eficaz. También, existen URLs cuyos títulos no son lo suficientemente representativos, esto impide obtener la suficiente información como para obtener una localización correcta.

El tercer algoritmo mantiene una precisión similar a la del primer algoritmo. Hay que destacar, que al utilizar imágenes no geolocalizadas se obtiene; por lo general, menos información de las imágenes. Al obtener menos información, se obtienen menos localizaciones, perjudicando así al algoritmo 3 cuyo objetivo principal es buscar aquella localización que más se repita para su posterior representación. Consecuentemente, en muchos casos no se consigue lo anterior. El algoritmo 3, en el caso de no obtener una localización repetida, escoge la localización que tenga asociado un score más alto. Esto, conlleva a que la diferencia entre el algoritmo 1 y el algoritmo 3 sea bastante reducida, diferenciándose tan solo en un margen del 6%, es decir, el algoritmo 1 ha conseguido identificar un 6% más de imágenes que el algoritmo 3.

4.2.3 Comparativa entre resultados de conjuntos de datos con y sin geolocalización

Tras realizar las diferentes pruebas, y observando las imágenes extraídas por los dos métodos (descritos en la sección 4.1) se extrae una posible conclusión. Las imágenes que tienen una geolocalización asociada se corresponden; según las imágenes obtenidas, normalmente, con imágenes de zonas o lugares conocidos de esas ciudades. Mientras que las imágenes que no tienen una

localización asociada se corresponden con lugares o zonas que son menos conocidas, y, por tanto, resulta más difícil extraer información de la propia imagen. Esta conclusión se refleja del aumento de 16 a 21 imágenes que la API de Google Vision no es capaz de obtener información.

Comparando el primer algoritmo, podemos ver que se mantiene estable con ambos tipos de imágenes, haciéndolo un algoritmo bastante fiable cuya precisión oscila entre el 45 y el 55%.

En el caso del segundo algoritmo, pese a obtener bajos resultados en ambos análisis podemos ver un gran descenso en las imágenes sin localizar. Probablemente; como hemos resaltado anteriormente, debido a esas imágenes que se corresponden con calles y zonas menos conocidas de la ciudad. Podemos ver un descenso del 14 al 3%.

En lo relativo al tercer algoritmo, mantiene resultados fiables en ambos análisis. Pero, he de destacar que en el primer caso al encontrarse trabajando con imágenes más reconocibles las localizaciones coincidentes que encontró fueron mayores, con lo que pudo desarrollar su función con una mayor frecuencia. Mantiene una precisión entre el 45 y el 40%.

4.2.4 Resultados obtenidos sobre imágenes de forma conjunta

Tenemos un conjunto de 35 usuarios y la tabla se encuentra formada por una serie de columnas las cuales son:

- Número: el número de la imagen.
- Carpeta: el nombre del usuario. No se muestra por razones de privacidad.
- N° imgs: número de imágenes existentes de cada carpeta.
- Ciudad: ciudad donde ha sido tomada la primera imagen del usuario, el resto de las imágenes pueden pertenecer a ciudades distintas.
- Loc: resultado de las localizaciones, hasta un máximo de 3 localizaciones.
- Porc: porcentaje de acierto que el algoritmo aplica a cada Loc(localización).
- Localizado: si ha conseguido localizar con éxito o no.

Para la columna ‘localizado’ hemos establecido un código de colores en función de si ha conseguido la localización correcta o no. Dicho código es descrito a continuación:

- Rojo
 - Columna de localizado: el algoritmo no ha proporcionado la localización correcta.
 - Loc, Porc y Localizado (rojo en las tres columnas): el algoritmo no tiene suficiente información para obtener ninguna localización.
- Naranja: el algoritmo ha conseguido la localización correcta, pero no con el máximo porcentaje posible.
- Verde: el algoritmo ha proporcionado la localización correcta con el máximo porcentaje asociada a ella.

A continuación, se muestra la tabla con los resultados obtenidos:

LOCALIZACIÓN DE IMÁGENES SUBIDAS POR USUARIOS DE REDES SOCIALES
MEDIANTE UTILIZACIÓN DE HERRAMIENTAS DE BÚSQUEDA VISUAL DE IMÁGENES

NUMERO	CARPETA	Nº IMGNS	CIUDAD	LOC1	PORC1	LOC2	PORC2	LOC3	PORC3	LOCALIZADO
1		3	MADRID	Madrid, Centro	0,7	Valencia	0,3			
2		7	MADRID	Tenerife	0,33	Valencia	0,33	Madrid	0,33	
3		3	MADRID	Madrid, Centro	0,5	Madrid, Centro	0,5			
4		2	MADRID	Madrid,Area Metropolitana	0,8	Madrid, Centro	0,2			
5		2	MADRID	Ciudad de Mexico	0,375	Roquetas Mar	0,125	USA	0,125	
6		2	MADRID	Ecuador	0,75	Madrid, Centro	0,25			
7		2	MADRID	Roquetas Mar	0,6	Mexico	0,2	Brasil	0,2	
8		2	MADRID	USA	0,43	Valencia	0,3	Madrid	0,29	
9		5	MADRID	Madrid, Centro	0,67	Madrid, Centro	0,33			
10		3	MADRID	Madrid, Centro	0,66	Madrid, Area	0,33			
11		4	MADRID	Barcelona	0,4	Ecuador	0,34	Madrid	0,26	
12		7	MADRID	Madrid, Centro	0,67	Madrid, Area	0,33			
13		2	MADRID	Madrid, Area	0,5	Valencia	0,5			
14		5	MADRID	Madrid, Centro	0,67	Madrid, Area	0,33			
15		3	MADRID	Mexico	0,67	Madrid	0,33			
16		2	MADRID	Ecuador	1					
17		3	MADRID	Brasil	0,82	Madrid	0,18			
18		3	MADRID	Barcelona	0,57	Guatemala	0,14	Madrid	0,14	
19		2	MADRID							
20		4	MADRID	Brasil	0,57	Algeria	0,14	USA	0,14	
21		2	PONTEVEDRA	Pontevedra	1					
22		2	PONTEVEDRA	Ourense	0,5	Pontevedra	0,5			
23		3	PONTEVEDRA							
24		2	SANTANDER	Argentina	1					
25		2	SANTANDER							
26		3	SANTANDER	Santander	0,5	Malaga	0,25	Flaciusova ulic	0,25	
27		3	SANTANDER							
28		2	SANTANDER							
29		3	SANTANDER	Santander	1					
30		3	SANTANDER	Santander	1					
31		2	SANTIAGO	Santiago	1					
32		2	SANTIAGO	Santiago	1					
33		3	SANTIAGO-FIN	Torre Hércules	0,75	Huesca	0,25			
34		2	SANTIAGO	Santiago	1					
35		3	FINISTERRE	Torre Hércules	0,6	Bota do Peregrino	0,4			

Tabla 7 -Resultados Detección de Usuario (Edición propia)

En base a la Tabla 7 -Resultados Detección de Usuario (Edición propia) realizamos los siguientes cálculos:

ALGORITMO	PRECISIÓN
Imágenes No suficiente información	18%
Localizado Verde	64%
Localizado Naranja	18%
Localizado Rojo	18%
Proporciona 3 Localizaciones	27%
Proporciona 2 Localizaciones	36%
Proporciona 1 Localizaciones	32%

Tabla 8 - Resultado Final Detección Usuarios (Edición propia)

Extrayendo conclusiones, vemos que en un 18% de imágenes no se ha concluido con ninguna localización. Pese a tener más de una imagen por cada usuario, el algoritmo no ha llegado a ninguna conclusión con la información que le ha sido aportada.

En cuanto a los usuarios de los que se ha aportado la localización correcta, vemos que en un 82% de los casos el algoritmo ha conseguido llegar a la conclusión correcta(verde) o parcialmente correcta(naranja). Tan solo en un 18% de los usuarios no ha conseguido llegar a una conclusión correcta, juntando ambos tipos de error. Tanto del que no se tiene suficiente información, como en el que el algoritmo ha proporcionado localizaciones incorrectas.

En un 63% de los casos de asignan al menos dos localizaciones a un usuario (dos y tres localizaciones), lo que muestra signo de la correcta ejecución del algoritmo (estas ponderaciones se deben tener en cuenta, dentro del 82% de imágenes en las que el algoritmo consigue al menos proporcionar una localización, es decir, sería el 63% del 82%).

Dentro de ese 63%, el 57% de las ocasiones está proporcionando la ubicación correcta, el 21,5% estará proporcionando una ubicación semicorrecta (color naranja) y el restante 21,5% estará proporcionando la ubicación incorrecta.

Dentro del 32% de las ocasiones donde se proporciona tan solo una ubicación, el 86% de las localizaciones proporcionadas son correctas. Mientras, que tan solo el 14% de las localizaciones son incorrectas.

4.2.5 Comparativa de ciudades

A continuación, presentaremos las tablas las cuales nos harán entender como funciona GeoApi con respecto a los resultados obtenidos en las diferentes ciudades. Esto nos permitirá analizar los resultados obtenidos, así como las causas que los provocan.

CIUDAD	ÉXITO	Nº IMAG.	SIN INFORMACIÓN	CON INFORMACIÓN
MADRID	42,5%	40	18%	82%
PONTEVEDRA	17%	16	25%	75%
SANTANDER	30%	20	40%	60%
SANTIAGO	51%	11	18%	82%

Tabla 9 - Análisis Ciudades con Geolocalización (Edición propia)

La tabla se encuentra compuesta por cuatro filas; que equivalen al nombre de las cuatro ciudades que son: Madrid, Pontevedra, Santander y Santiago, y cuatro marcadores distintos que comprenden los siguientes significados:

- ÉXITO: media del éxito que han conseguido en la localización los tres algoritmos. Recordemos que los posibles valores son: 1, 0,5 y 0.
- Nº IMAG: número de imágenes a analizar en cada ciudad.
- SIN INFORMACIÓN: número de imágenes de las cuales no se han conseguido obtener información y almacenarla en su correspondiente JSON.
- CON INFORMACIÓN: tanto por ciento de las imágenes de las cuales se ha obtenido información y ha sido almacenada correctamente en el archivo JSON.

En este primer caso, analizaremos las imágenes que tienen una geolocalización asociada. El primer factor que describimos es la tasa media de éxito que han conseguido los tres algoritmos en las diferentes ciudades, podemos observar cómo Santiago y Madrid están por delante de Santander y Pontevedra. Esto es posiblemente debido a un mayor número de imágenes en las que se presentan

monumentos o lugares más conocidos. Como ejemplo de ello, vemos que en Santiago la mayoría de las imágenes se corresponden con la Catedral o la plaza que se encuentra anexa a ella, consecuentemente su mayor porcentaje de éxito. La capital Madrid obtiene también un buen porcentaje, probablemente porque contiene imágenes de calles o zonas emblemáticas, como pueden ser el retiro, es Palacio de Cristal o Gran Vía. Los peores porcentajes los vemos en Pontevedra y Santander, posiblemente este menor porcentaje estará relacionado con tener monumentos y calles menos populares.

Hablando de la capacidad de obtener información de las imágenes por la API Google Vision, observamos como los mayores porcentajes de éxito se obtienen de forma similar en Madrid, Pontevedra y Santiago. Sin embargo, en Santander se consigue un porcentaje considerablemente menor. Gran parte de la culpa de que de esas imágenes no se consiga obtener información, es que dichos archivos se corresponden en la mayoría de sus casos con elementos paisajísticos como bien pueden ser: playas, mar, puentes... De ahí relacionamos esto con una menor tasa tanto en Pontevedra como en Santander.

A continuación, presentamos la tabla para realizar ese análisis de las ciudades en las imágenes sin Geolocalización.

CIUDAD	ÉXITO	Nº IMAG.	SIN INFORMACIÓN	CON INFORMACIÓN
MADRID	36%	40	25%	75%
PONTEVEDRA	17%	40	33%	67%
SANTANDER	48%	40	28%	72%
SANTIAGO	36%	40	17%	83%

Tabla 10 - Análisis Ciudades sin Geolocalización (Edición propia)

En el caso de las ciudades sin geolocalización, vemos como la tasa media de los diferentes algoritmos para las cuatro ciudades ha disminuido, salvo en el caso de Santander. De lo anterior, concluimos que este tipo de imágenes; por lo general, suelen corresponderse con partes o zonas menos conocidas o incluso no corresponderse con la ciudad.

Pese a lo anterior, los porcentajes se mantienen superiores en las ciudades de Madrid y Santiago con respecto a Pontevedra. Por lo que podemos ratificar lo que se dijo con respecto a esta tasa en las imágenes geolocalizadas, es decir, que las ciudades de Madrid y Santiago obtienen mejores porcentajes debido a que presentan elementos más conocidos en sus imágenes. En la Tabla 10 - Análisis Ciudades sin Geolocalización (Edición propia) consideramos su alta tasa de éxito como una excepción. Esta se encuentra relacionada con la aparición en la mayoría de sus imágenes de su infraestructura más emblemática ‘El Palacio de la Magdalena’.

Acerca del porcentaje de imágenes de las que se ha extraído información con éxito a partir de la API Google Vision. Concluimos que, al igual que en la tasa de éxito en la localización, se ha producido una disminución de los porcentajes, salvo en Santander lo que podría ser una excepción. Esta disminución, se encuentra posiblemente relacionada con el uso de imágenes que se corresponden con zonas menos conocidas de la ciudad ante lo que la API tiene una peor respuesta.

A continuación, presentaremos una tabla donde se analiza los resultados del análisis de las ciudades para la detección de imágenes de forma conjunta:

CIUDAD	ÉXITO	Nº USUARIOS.	SIN INFORMACIÓN	CON INFORMACIÓN
MADRID	75%	10	5%	95%
PONTEVEDRA	66%	3	33%	67%
SANTANDER	42%	7	43%	57%
SANTIAGO	100%	5	0%	100%

Tabla 11 - Análisis ciudades conjunto de imágenes (Edición propia)

La tabla se encuentra compuesta por cuatro filas; que equivalen al nombre de las cuatro ciudades que son: Madrid, Pontevedra, Santander y Santiago, y cuatro marcadores cuyo significado es el mismo que el de la Tabla 9 - Análisis Ciudades con Geolocalización (Edición propia). La única diferencia se produce en la columna ‘Nº USUARIO’ donde se plasma el número de usuarios a analizar en cada ciudad, dentro de cada usuario existe un número determinado de imágenes (no el mismo para todos los usuarios).

En cuanto a la primera columna, vemos se han obtenido porcentajes altos comenzando con Santiago con un 100% de las localizaciones, le sigue Madrid con un 75%, tras Madrid esta Pontevedra con un 66% y, por último, Santander con un 42%.

Podemos ver como con este algoritmo se han obtenido mejores resultados por lo general. También podemos establecer una relación a este caso, puesto que el número de imágenes que son usados para obtener la localización es mayor. De este modo, es más inusual que no se obtenga la información necesaria para realizar un buen análisis de cada usuario. Además, utilizando los tres algoritmos que desarrollamos al principio nos aseguramos de que con sus distintas funcionalidades alguno proporcione la ubicación correcta.

En lo referido al número de usuarios, vemos como lógicamente el mayor número se observa en Madrid, posteriormente Santander, Santiago y, en último lugar Pontevedra, Observamos como existe relación entre el número de usuarios y la población existente en cada ciudad.

En cuanto a las carpetas de usuarios de las que se ha conseguido extraer información, vemos como tan solo en las ciudades de Santander y Pontevedra ha habido usuarios de los que no se ha conseguido obtener ninguna información. De lo anterior, podemos sacar las mismas conclusiones que en los apartados 4.2.1, 4.2.2 y . Las ciudades menos turísticas, son las que obtienen peores resultados a la hora de que la API Google Vision pueda obtener información de ellas.

4.2.6 Detección y eliminación de objetos

Una vez implementada nuestra función para la localización y eliminación de imágenes, realizaremos una selección de imágenes de la web y comprobaremos el correcto funcionamiento de la mejora implementada.

A continuación, presentaremos una tabla con los resultados adquiridos, y un ejemplo de la imagen antes de ser sometido a la eliminación de objetos y la misma imagen después de haber sido sometida a dicho proceso:

NUMERO	DESCRIPCIÓN	ÉXITO
1	TORRE EIFFEL	SÍ
2	TORREL EIFFEL	SÍ
3	TORRE EIFFEL	SÍ
4	TORRE EIFFEL	SÍ
5	MONUMENTO CABALLERIA	SÍ
6	MONUMENTO CABALLOS	SÍ
7	CEMENTERIO FLORES	SÍ
8	MONUMENTO COCHES	SÍ
9	MUSEO LOUVRE	SÍ
10	ARCO TRIUNFO	SÍ

Tabla 12 - Análisis Detección Eliminación Objetos (Edición propia)

La tabla está confeccionada por tres columnas:

- Columna 1: número de la imagen.
- Columna 2: la descripción, donde ha sido tomada la imagen.
- Columna3: el éxito, tiene dos opciones las cuales son sí o no.
 - Sí: se han borrado los elementos deseados.
 - No: no se han borrado los elementos deseados o se ha borrado el monumento o edificio del que se quiere obtener la información.

Vemos que todas las imágenes que han sido sometidas a esta función, se ha conseguido eliminar todos los elementos no deseados o, bien, la mayoría de ellos. La API de Google Vision es capaz de localizar todos los objetos en imágenes cercanas donde no haya una gran cantidad de detalles, en este tipo de imágenes selecciona y recuadra los elementos de mayor tamaño.

Otro aspecto positivo de la API es que no solo es capaz de detectar objetos, sino que es capaz de detectar la silueta de las personas y eliminarla de la imagen. Esto es bastante útil, puesto que disminuimos la cantidad de información que GeoApi tiene que procesar.

Podremos ir mejorando continuamente el funcionamiento de esta parte de código, añadiendo en la función aquellas palabras que no queremos que vuelvan a ser eliminadas por la API.

```
if caption == 'Animal' or caption == 'Building' or caption == 'Monument' or caption == 'Statue' or caption == 'Window'
    print('')
```

Figura 77 - Objetos No son Eliminados (Edición propia)

A continuación, mostraremos la imagen antes y después de ser editada:



Figura 78 - Antes de detección y eliminación de objetos (tomada de [50])



Figura 79 - Después de detección y eliminación de objetos (Edición propia)

4.2.7 Detección y eliminación de caras

Una vez implementada nuestra función para la localización y eliminación de caras, realizaremos una selección de imágenes de la web y comprobaremos el correcto funcionamiento de la mejora implementada.

Presentaremos un ejemplo de una imagen en el paso previo de ser sometida a la detección y eliminación de caras y, posteriormente, con dicha función implementada. Posteriormente, insertaremos una tabla mostrando los resultados de esta aplicación.

Prosiguiendo con los resultados, mostraremos una tabla con las diez imágenes a las que las hemos sometido a análisis:

NÚMERO	DESCRIPCIÓN	Nº PERSONAS	DETECTADAS
1	PANTEÓN ATENAS	2	2
2	MONUMENTO JUDÍOS BERLÍN	1	1
3	COLISEUM	1	1
4	PUENTE SAN FRANCISCO	1	1
5	ROMA PLAZA ESPAÑA	1	1
6	SAGRADA FAMILIA	1	1
7	FONTANA DI TREVI	1	1
8	MONUMENTO CHE GUEVARÁ	2	1
9	DESIERTO	1	1
10	PLAZA ROJA MOSCÚ	2	2

Tabla 13 - Detección Eliminación Caras (Edición propia)

Las columnas que componen la tabla son:

- Columna 1: número de la imagen.
- Columna 2: la descripción, donde ha sido tomada la imagen.
- Columna 3: el número de personas que aparecen en la imagen.
- Columna 4: de esas personas que aparecen en la imagen, cuantas de ellas son detectadas y eliminadas correctamente.

Podemos visualizar como en la tabla tan solo una persona no consigue ser detectada correctamente. La no detección de los rostros en las imágenes no nos va a producir ningún perjuicio en la aplicación, por lo que podemos admitir este tipo de errores. De todos modos, la tasa de detección de rostros es bastante alta, podemos predecir que su implementación en GeoApi será de gran ayuda para la API Google Vision.

A continuación, mostraremos las imágenes antes y después de la localización y eliminación de caras.



Figura 80 - Antes de detección y eliminación de caras (tomada de [51])



Figura 81 - Después de detección y eliminación de caras (Edición propia)

Por cuestión de tiempo, no hemos podido realizar una prueba que muestre con datos empíricos como mejoran estas dos funciones la obtención de localizaciones por parte de la API Google Vision.

5 CONCLUSIONES Y LÍNEAS FUTURAS

5.1 Conclusiones

La conclusión fundamental obtenida con este TFG, es que no sólo se han alcanzado los objetivos marcados, sino que también se han sobrepasado al haber implementado tratamientos de mejora de imágenes como son la detección y eliminación, tanto de caras, como de objetos.

En primer lugar, se realizó un estudio de las diferentes funcionalidades de la API Google Vision, para trabajar con aquellas que nos dieran unos mejores resultados. Consecuentemente, seleccionamos la detección de páginas webs, detección de paisajes, detección de caras y detección de objetos. A continuación, realizamos el código que permite acceder a la API mencionada para utilizar aquellas funcionalidades requeridas. También se implementó todo lo necesario para poder manejar la información proporcionada en JSON.

En segundo lugar, desarrollamos el sistema que incorpora los diferentes algoritmos. El primer algoritmo presentará la ubicación a la cual se le haya asignado una mayor puntuación. El segundo algoritmo arrojará la localización asociada partiendo de las palabras procedentes del metadato del título de la página web en la que se encuentre imágenes similares a la tratada. Por último, nuestro tercer algoritmo presenta la localización que la API presente con mayor frecuencia, es decir, las coordenadas que se repitan un mayor número de veces. Estos algoritmos son diseñados para trabajar con una sola imagen.

El salto cualitativo se produce con el desarrollo del algoritmo denominado ‘Detección de usuario’, el cual ofrece las diferentes localizaciones en las que se ha encontrado el usuario en un espacio temporal determinado. Este algoritmo permite trabajar con un conjunto de imágenes a diferencia de los anteriores.

La siguiente mejora implementada está relacionada con el tratamiento de imágenes, para la cual utilizamos las funciones de detección de objetos y detección de caras de la API Google Vision. Estas permiten; como destacamos anteriormente, la detección y eliminación de objetos, individuos y rostros faciales. Estas mejoras implican mayor facilidad para GeoApi para conseguir arrojar resultados.

Una vez hemos confeccionado nuestros algoritmos, también hemos realizado las pruebas con las imágenes extraídas de Twitter. De las pruebas realizadas pudimos extraer una serie de conclusiones. Comenzando, con que la función de detección de paisajes trabaja correctamente cuando detecta esculturas, monumentos o edificios conocidos, pero no es tan precisa a la hora de detectar calles, edificios o paisajes que no seas muy populares. Por otro lado, la función de detección de páginas webs arroja buenos resultados con aquellas imágenes que se encuentran dentro de webs que se puedan acceder desde el buscador de Google.

También, los resultados ofrecidos por GeoApi no ofrecen resultados tan claros cuando las imágenes se corresponden con fotografías que han sido tomadas en la naturaleza, como pueden ser bosques, montañas o ríos.

Hay que destacar, que nuestro algoritmo funciona mejor con ciudades conocidas como pueden ser Madrid o Santiago, puesto que el número de resultados de las páginas webs y elementos paisajísticos conocidos es mayor.

Con respecto al algoritmo de detección de paisajes, este funciona de manera bastante eficiente con ciudades y elementos paisajísticos conocidos. Lo sorprendente, es que en algunas ocasiones arroja como primer resultado una localización la cual no tiene que ver nada con las imágenes que han sido analizadas. Esto, posiblemente es debido al origen de las páginas webs de donde se obtiene la información. En algunas ocasiones ha presentado localizaciones como Brasil, Ecuador o Estados Unidos, con una imagen que pertenecía a Madrid.

Por último, en lo referido a las mejoras en el tratamiento de imágenes, las mejoras se han implementado de forma satisfactoria en las imágenes a analizar. Pero, no podemos extraer una conclusión clara acerca de una mejora de los resultados ofrecidos por la API Google Vision, puesto que por falta de tiempo no hemos podido realizar pruebas que demuestren empíricamente una mejora en los resultados ofrecidos por la API de Google Vision.

5.2 Líneas futuras

Se proponen una serie de aspectos en los que este trabajo podría seguir evolucionando para constituir una herramienta para la búsqueda inversa de imágenes más completa.

En primer lugar, como hemos recalcado anteriormente, la API Google Vision a través de la función de detección de páginas webs obtiene menos URLs cuando el buscador de Google no tiene indexada la imagen. Consecuentemente, se deberá tratar de investigar e implementar otra serie de funcionalidades las cuales aumenten la información que tenemos acerca de dicha imagen, esto nos permitirá obtener localizaciones más precisas.

Relacionado con lo anterior, otro de los aspectos a mejorar es la obtención de metadatos. De muchas de las URLs obtenidas no se han conseguido obtener los metadatos de forma efectiva o, bien, estos no eran capaces de presentar información de utilidad. Se debería desarrollar un sistema que extraiga los metadatos de forma efectiva y clasifique estos para que en el caso de que no sean de utilidad, desecharlos.

Para la obtención de una mayor precisión se deberían establecer una serie de mejoras en los algoritmos sobre todo en el segundo y en el tercer sistema. En el segundo; que es el que realiza la búsqueda a partir de la palabra más frecuente dentro de los títulos proporcionados por los metadatos, habría que eliminar aquellas palabras como pueden ser pronombres, artículo, preposiciones... que no tienen una localización asociada, es decir, las stopwords. En el caso del tercer algoritmo, la mejora está relacionada con un aumento de la información que proporciona la API Google Vision, cuanto mayor información mayor exactitud en la representación de localizaciones.

Otra línea futura, podría ser el análisis de imágenes procedentes de otra red social como puede ser Instagram. Esta red social está mucho más centrada en imágenes y vídeos. Además, también las publicaciones de sus usuarios pueden ser localizadas. Por lo que sería de interés comparar los resultados.

En último lugar, sería importante la implementación de más herramientas para la edición de imágenes, y la realización de un estudio que compruebe de verdad como ese tratamiento de imágenes afecta a la obtención de información por parte de la API Google Vision.

6 BIBLIOGRAFÍA

- [1] T. Graspas, «rockcontent,» 17 12 2017. [En línea]. Available: <https://rockcontent.com/es/blog/historia-de-las-redes-sociales/>. [Último acceso: 14 03 2022].
- [2] P. d. M. Gonzalez, «LaRazon,» 17 1 2022. [En línea]. Available: <https://larazon.es/espana/20220117/gjrtememond4vegknrgfqa434a.html>. [Último acceso: 26 Enero 2022].
- [3] «ARA,» [En línea]. Available: https://es.ara.cat/sociedad/internautas-ayudan-policia-nacion-identificar-menor-riesgo-abusos_1_4242794.html.
- [4] J. J. Flores Cueto, «Boletín electrónico de la Unidad de Virtualización Académica - UVA,» 2009. [En línea]. Available: http://files.andresalvarez.webnode.es/2000000092-d07c9d2704/redes_sociales.pdf.
- [5] «webempresa,» 11 03 2022. [En línea]. Available: <https://www.webempresa.com/blog/que-es-twitter-como-funciona-2.html>. [Último acceso: 01 03 2018].
- [6] D. Carroll, «From Twitter to Traffic Predictor,» 05 05 2021. [En línea]. Available: <https://www.cmu.edu/news/stories/archives/2021/may/traffic-prediction.html>. [Último acceso: 11 2021].
- [7] R. Fernández, «statista,» 03 2021. [En línea]. Available: <https://es.statista.com/estadisticas/489153/porcentaje-de-internautas-en-las-redes-sociales-en-espana> [Último acceso: 11 03 2022].
- [8] «Internet Live Stats,» [En línea]. Available: <https://www.internetlivestats.com/twitter-statistic> [Último acceso: 11 03 2022].
- [9] C. Shu, «From Twitter to Traffic Predictor,» 19 06 2019. [En línea]. Available: https://techcrunch.com/2019/06/18/twitter-will-remove-location-tagging-in-tweets-citing-lack-of-use/?guccounter=1&guce_referrer=aHR0cHM6Ly93d3cuZ29vZ2xlLmNvbS8&guce_referrer_sig=QAAAKhMCohGtfzjvvqTq1mV4SiLQN45dGrbV0_anE-nYJkhUQPXenLer0jeZ06mP4tgDuMk_QKbh. [Último acceso: 11 03 2022].
- [10] «FourSquare,» [En línea]. Available: <https://foursquare.atlassian.net/servicedesk/customer/portal/2> [Último acceso: 11 03 2022].
- [11] «elandroidefeliz,» 29 09 2021. [En línea]. Available: <https://elandroidefeliz.com/obtene-coordenadas-gps-de-una-foto/>. [Último acceso: 11 03 2022].

- [12] «Google Play,» [En línea]. Available: <https://play.google.com/store/apps/details?id=jp.mydns.usagigoya.imagesearchviewer&hl=es&gl=US>. [Último acceso: 11 03 2022].
- [13] «Orientatech,» 07 2017. [En línea]. Available: <https://www.orientatech.es/aplicacion-camfin> [Último acceso: 11 03 2022].
- [14] «WikiInclusion,» [En línea]. Available: https://wikinclusion.org/index.php/CAM_FIND.
- [15] C. Tech, «Giztab,» [En línea]. Available: <https://www.giztab.com/picture-this-la-app-para-reconocer-plantas/>.
- [16] E. Ralls, «Plant Snap,» RedOrbit, 2012. [En línea]. Available: <https://www.plantsnap.com/who-we-are/>. [Último acceso: 11 03 2022].
- [17] M. t. e. y. r. demográfico, «Gobierno de España,» [En línea]. Available: <https://www.miteco.gob.es/es/ceneam/recursos/pag-web/plantner-plantsnap.aspx>.
- [18] «Mas Movil,» [En línea]. Available: <https://blog.masmovil.es/buscar-imagenes-google-desde-movil>
- [19] «Playstore,» [En línea]. Available: <https://play.google.com/store/apps/details?id=com.palmteam.imagesearch&hl=es&gl=US>. [Último acceso: 11 03 2022].
- [20] andr2id. [En línea]. Available: <https://andro2id.com/buscar-persona-por-foto/>.
- [21] «Playstore,» [En línea]. Available: <https://play.google.com/store/apps/details?id=hippeis.com.photochecker&hl=es&gl=US>. [Último acceso: 11 03 2022].
- [22] «Xataka,» [En línea]. Available: <https://www.xataka.com/basics/google-lens-todo-que-puedes-hacer-con-la-aplicacion-google>.
- [23] «Google Lens,» [En línea]. Available: <https://lens.google/intl/es-419/>. [Último acceso: 11 03 2022].
- [24] E. Hidalgo, «Redes Zone,» 04 01 2013. [En línea]. Available: <https://www.redeszone.net/2013/01/04/creepy-geolocaliza-a-tus-amigos-de-twitter-o-flickr/>. [Último acceso: 11 03 2022].
- [25] J. Rodríguez y V. Fernández, Cómo redactar el estado del arte de un trabajo, Editorial Genios, 2010.
- [26] «WeSee,» [En línea]. Available: <https://okdescargas.com/wesee-buscador-a-traves-de-imagenes/>.
- [27] «WeSee» <https://www.wesee.com/>, [En línea]. Available: <https://www.wesee.com/>. [Último acceso: 12 03 2022].
- [28] A. Kurmakaeva, «Human Level,» [En línea]. Available: <https://www.humanlevel.com/diccionario-marketing-online/yandex#:~:text=Yandex%20es%20un%20buscador%20ruso,Arkady%20Volozh%20e%20Ily%20Segalovich..> [Último acceso: 12 03 2022].
- [29] G. González, «Genbeta,» 31 12 2018. [En línea]. Available: <https://www.genbeta.com/buscadores/subir-foto-a-yandex-excelente-examen-tu-privacidad-red-que-puedes-hacer-ahora>. [Último acceso: 12 03 2022].
- [30] S. Cardozo, «Nichoseo,» 26 11 2019. [En línea]. Available: <https://nichoseo.com/yandex-se> [Último acceso: 12 03 2022].

- [31] «rockcontent,» 07 06 2022. [En línea]. Available: <https://rockcontent.com/es/blog/quien-fund-google/>. [Último acceso: 12 03 2022].
- [32] V. Urbina, «fditf.com,» 27 02 2019. [En línea]. Available: <http://www.fditf.com/bi/google-app-script/geocodificacion-inversa-en-google-sheets-reverse-geocoding/>. [Último acceso: 12 03 2022].
- [33] David, «Coseom,» [En línea]. Available: <https://www.coseom.com/es/que-es-google-que-es-lo-qu-hace-a-google-tan-diferente-del-resto-de-companias/>. [Último acceso: 12 03 2022].
- [34] «Centro de formación técnica para la industria,» [En línea]. Available: <https://www.cursosaula21.com/que-es-python/>. [Último acceso: 12 03 2022].
- [35] A. Vallejo, «Genbeta,» [En línea]. Available: <https://www.genbeta.com/actualidad/python-s-convierte-lista-tiobe-lenguaje-popular-red-superando-incluso-a-c>. [Último acceso: 12 03 2022].
- [36] «IDLE,» 25 02 2022. [En línea]. Available: <https://www.mclibre.org/consultar/python/otros/pytho-idle.html>. [Último acceso: 12 03 2022].
- [37] Google, «Chroome Magnament API,» [En línea]. Available: <https://developers.google.com/chrome/management>.
- [38] I. Peñalba, «Google Vision API: El reconocimiento de imágenes de Google Photos para cualquier aplicación,» [En línea]. Available: https://www.elespanol.com/elandroidelibre/noticias-novedades/desarrollo-y-programacion/20160219/google-vision-api-reconocimiento-photos-cualquier-aplicacion/103490016_0.html.
- [39] «Google cloud,» Google, [En línea]. Available: <https://cloud.google.com/vision>. [Último acceso: 03 2022].
- [40] «Google cloud,» Google, [En línea]. Available: <https://cloud.google.com/vision>. [Último acceso: 03 2022].
- [41] A. Sarasa, Gestión de la Información Web usando Python, Barcelona: UOD, 2016.
- [42] M. Beck, «Betterprogramming,» 03 12 2020. [En línea]. Available: <https://betterprogramming.pub/how-to-scrape-tweets-with-snsrape-90124ed006af>. [Último acceso: 12 03 2022].
- [43] S. Tomlins, «The Starup,» 23 12 2020. [En línea]. Available: <https://medium.com/swlh/how-to-scrape-tweets-by-location-in-python-using-snsrape-8c870fa6ec25>. [Último acceso: 12 03 2022].
- [44] «MongoDB,» [En línea]. Available: <https://www.mongodb.com/es>. [Último acceso: 12 03 2022].
- [45] «OVHCloud,» [En línea]. Available: <https://www.ovhcloud.com/es-es/public-cloud/mongodb>. [Último acceso: 12 03 2022].
- [46] «PowerData,» PowerData, 2016. [En línea]. Available: <https://blog.powerdata.es/el-valor-de-l-gestion-de-datos/que-son-los-metadatos-y-cual-es-su-utilidad>. [Último acceso: 13 Febrero 2022].
- [47] «PowerData,» 18 03 2016. [En línea]. Available: <https://blog.powerdata.es/el-valor-de-la-gestion-d-datos/que-son-los-metadatos-y-cual-es-su-utilidad>. [Último acceso: 12 03 2022].
- [48] «Geopy,» [En línea]. [Último acceso: 2022].
- [49] K. Esmukov, «GitHub,» 03 07 2018. [En línea]. Available: <https://github.com/geopy/geopy/issues/311>. [Último acceso: 12 03 2022].

- [50] «Pinterest,» [En línea]. Available: <https://www.pinterest.ru/pin/170573904613622346/>. [Últim acceso: 14 03 2022].
- [51] «Flickr,» [En línea]. Available: <https://www.flickr.com/photos/7503705@N07/4216344219> [Último acceso: 14 03 2022].

ANEXO I: ALGORITMO 1

```
import json
import io, os
from google.cloud import vision
from geopy.geocoders import GeoNames
import pandas as pd
import metadata_parser
import json
#Obtencion coordenadas...Geoname
from geopy.geocoders import Nominatim
#abrir imagenes y obtener url
from tkinter import *
from PIL import Image, ImageTk
from tkinter import filedialog
from tkinter import Tk
from tkinter.filedialog import askopenfilename
import re
from collections import Counter
import geocoder
import folium
import webbrowser

#Interfaz con el usuario
Tk().withdraw()
filename = askopenfilename()#Abrir dirección que ha escogido
print(filename) #imprimir dirección del directorio
nombre = os.path.basename(filename)#extraer el nombre del archivo
print(nombre)#imprimir nombre del archivo

#crear file JSON/eliminar signos de puntuación
s = nombre
out = re.sub(r'[\^\\w\\s]',",",s) #eliminamos signos de puntuación
print(out) #añadimos .json
archivo= out +'.json'
```

```
print(archivo)

loc = Nominatim(user_agent="GetLoc")

#creamos una lista vacia
info= []

with open(f'{filename}{archivo}') as file:
    data = json.load(file)

    for foto in data['fotos']: #anlizamos los resultados del formato JSON
        Title = foto['title']
        Lat = foto['Latitud']
        Long = foto['Longitud']

        if not Lat: #No contienen Latitud
            info.append(Title)
        else:
            print(Lat, Long)
            var = Lat, Long
            #Transformación de las coordenadas a dirección
            locname = loc.reverse(var)
            print(locname)
            #introducimos valor en una lista/append hace que se vaya acumulando
            info.append(locname)

print(info[0])
#Obtencion Coordenadas de localizacion
getLoc = loc.geocode(info[0])
print(getLoc.address)
print("Latitude = ", getLoc.latitude, "\n")
print("Longitude = ", getLoc.longitude)
Lat=getLoc.latitude
Long=getLoc.longitude

#abrir un mapa interactivo
```

```
#webbrowser.open("https://www.python.org")
var = Lat, Long
mapa=folium.Map(location = var, zoom_start=12)

#crear puntos espaciales
tooltip = getLoc
marcador1 = folium.Marker(location= var, popup='ubi', tooltip=tooltip)
marcador1.add_to(mapa)

mapa.save('mimapa.html')
webbrowser.open('mimapa.html')
```

ANEXO II: ALGORITMO 2

```
import json
import io, os
from google.cloud import vision
from geopy.geocoders import GeoNames
import pandas as pd
import metadata_parser
import json
#Obtencion coordenadas...Geoname
from geopy.geocoders import Nominatim
#abrir imagenes y obtener url
from tkinter import *
from PIL import Image, ImageTk
from tkinter import filedialog
from tkinter import Tk
from tkinter.filedialog import askopenfilename
import re
from collections import Counter
import geocoder
import folium
import webbrowser

#Interfaz con el usuario
Tk().withdraw()
filename = askopenfilename()#Abrir dirección que ha escogido
print(filename) #imprimir dirección del directorio
nombre = os.path.basename(filename)#extraer el nombre del archivo
print(nombre)#imprimir nombre del archivo

#crear file JSON/eliminar signos de puntuación
s = nombre
out = re.sub(r'[\^\\w\s]',",",s) #eliminamos signos de puntuación
print(out)#añadimos .json
archivo= out +'.json'
```

```
print(archivo)

loc = Nominatim(user_agent="GetLoc")

#creamos cadenas de texto
cadena=""
localizacion =""

#Abrimos formato JSON
with open(f'{filename}{archivo}') as file:
    data = json.load(file)

    for foto in data['fotos']: #anlizamos los resultados del formato JSON
        title = foto['title']

        if title is None: #No contienen title
            continue

        else:
            string = title
            #eliminamos signos puntuacion
            salida = re.sub(r'[^\w\s]', ' ',str(string))
            #eliminamos numeros
            newstring = ".join([i for i in string if not i.isdigit()])
            new_string = re.sub(r"[^a-zA-Z0-9]", " ",newstring)
            #introducimos valor en una cadena de texto
            cadena += str(new_string) #eliminar signos de puntuación

print(cadena)

listaPalabras = cadena.split() #separamos las palabras de la cadena de texto

#creamos una lista para representar la frecuencia con la que se repiten las palabras
frecuenciaPalab = []

#recuento de las palabras que hay en la lista
```

```
for w in listaPalabras:
    frecuenciaPalab.append(listaPalabras.count(w))

print("Cadena\n" + cadena + "\n")
print("Lista\n" + str(listaPalabras) + "\n")
print("Frecuencias\n" + str(frecuenciaPalab) + "\n")
print("Pares\n" + str(list(zip(listaPalabras, frecuenciaPalab))))

for valor_a, valor_b in zip(frecuenciaPalab, listaPalabras): #obtenemos los valores en cada iteración
    if valor_a > 1: #si palabra se repite mas de dos veces

        if loc.geocode(valor_b) is None: #si la palabra repetida no tiene asignado una localizacion
            continue

        else:
            localizacion += str(valor_b)
            localizacion += ( ' ')
        else:
            continue

if localizacion != True:
    print(localizacion)
    palabra = localizacion.split()
    contador = Counter(palabra)
    palabra_frecuente = contador.most_common()[0][0] #palabra más repetida
    print(palabra_frecuente) #nos da la palabra que más se repite, dentro de las más frecuentes y tienen
    localizacion

    getLoc=loc.geocode(palabra_frecuente) #obtención de valores

    print(getLoc.address)
    print("Latitude = ", getLoc.latitude, "\n")
    print("Longitude = ", getLoc.longitude)
    Lat=getLoc.latitude
    Long=getLoc.longitude
    var = Lat, Long
```

```
#abrir un mapa interactivo
```

```
mapa=folium.Map(location = var, zoom_start=12)
```

```
#crear puntos espaciales
```

```
tooltip = getLoc
```

```
marcador1 = folium.Marker(location= var, popup='ubi', tooltip=tooltip)
```

```
marcador1.add_to(mapa)
```

```
mapa.save('mimapa.html')
```

```
webbrowser.open('mimapa.html')
```

```
else:
```

```
print("Este algoritmo no es válido para esa búsqueda")
```

ANEXO III: ALGORITMO 3

```
import json
import io, os
from google.cloud import vision
from geopy.geocoders import GeoNames
import pandas as pd
import metadata_parser
import json
#Obtencion coordenadas...Geoname
from geopy.geocoders import Nominatim
#abrir imagenes y obtener url
from tkinter import *
from PIL import Image, ImageTk
from tkinter import filedialog
from tkinter import Tk
from tkinter.filedialog import askopenfilename
import re
from collections import Counter
import geocoder
import folium
import webbrowser

#Interfaz con el usuario
Tk().withdraw()
filename = askopenfilename()#Abrir dirección que ha escogido
print(filename) #imprimir dirección del directorio
nombre = os.path.basename(filename)#extraer el nombre del archivo
print(nombre)#imprimir nombre del archivo

#crear file JSON/eliminar signos de puntuación
s = nombre
out = re.sub(r'[\^\\w\s]',",",s) #eliminamos signos de puntuación
print(out)#añadimos .json
archivo= out +'.json'
```

```
print(archivo)

loc = Nominatim(user_agent="GetLoc")

#creamos una lista vacia
coordLat=[]
coordLong=[]
score=[]
coincLAT=[]
coincLONG=[]
coincSCORE=[]
info=[]

with open(f'{filename}{archivo}') as file:
    data = json.load(file)

    for foto in data['fotos']: #anlizamos los resultados del formato JSON
        Lat = foto['Latitud']
        Long = foto['Longitud']
        Score = foto['score']
        title = foto['title']

        if title is not None:
            info.append(title)

        if not Lat: #No contienen Latitud
            continue

        if not Score:
            continue

    else:
        #introducimos valor en una Lista
        Lat1=float('{0:.2f}'.format(Lat))
        Long1=float('{0:.2f}'.format(Long))
        coordLat.append(Lat1)
```

```
coordLong.append(Long1)
score.append(Score)

#Busqueda de localizaciones en la lista
for i, h, j in zip(range(len(coordLat)), coordLong, score):
    for j in range(len(coordLat)):
        if i != j:
            if coordLat[i] == coordLat[j]:
                coincLAT.append(coordLat[i])
                coincLONG.append(coordLong[i])
                coincSCORE.append(score[i])
            else:
                continue
        else:
            continue

#Caso de que haya más de una localizacion coincidente
if len(coincSCORE) > 1:
    maximo=max(coincSCORE) #obtenemos el valor máximo de la lista
    lugar = coincSCORE.index(maximo) #Obtener posicion de dicho valor en la lista
    LAT1 = coincLAT[lugar]
    LONG1 = coincLONG[lugar]
    var = LAT1, LONG1
    print(var)

#abrir un mapa interactivo
mapa=folium.Map(location = var, zoom_start=12)

#crear puntos espaciales
tooltip = var
marcador1 = folium.Marker(location= var, popup='ubi', tooltip=tooltip)
marcador1.add_to(mapa)

mapa.save('mimapa.html')
webbrowser.open('mimapa.html')
```

```
#Si no hay coincidentes o solo hay una
else:
    for foto in data['fotos']: #anlizamos los resultados del formato JSON
        Lat = foto['Latitud']
        Long = foto['Longitud']
        Score = foto['score']
        title = foto['title']

        if Lat!="":
            LAT2 = foto['Latitud']
            LONG2 = foto['Longitud']
            var2 = LAT2, LONG2
            print(var2)
            break

        else:
            print(title)
            getLoc = loc.geocode(foto['title'])
            print(getLoc.address)
            print("Latitude = ", getLoc.latitude, "\n")
            print("Longitude = ", getLoc.longitude)
            Lat=getLoc.latitude
            Long=getLoc.longitude
            var2= Lat, Long
            print(var2)
            break

#abrir un mapa interactivo
mapa=folium.Map(location = var2, zoom_start=12)

#crear puntos espaciales
tooltip = var2
marcador1 = folium.Marker(location= var2, popup='ubi', tooltip=tooltip)
marcador1.add_to(mapa)
mapa.save('mimapa.html')
webbrowser.open('mimapa.html')
```

ANEXO IV: ALGORITMO DETECCIÓN USUARIO

```
import json
import io, os
from google.cloud import vision
from geopy.geocoders import GeoNames
import pandas as pd
import metadata_parser
import json
#Obtencion coordenadas...Geoname
from geopy.geocoders import Nominatim
#abrir imagenes y obtener url
from tkinter import *
from PIL import Image, ImageTk
from tkinter import filedialog
from tkinter import Tk
from tkinter.filedialog import askopenfilename
import re
from collections import Counter
import geocoder
import folium
import webbrowser
from os.path import isfile, join
from PIL import Image

#Interfaz con el usuario
#Tk().withdraw()
#filename = askopenfilename()#Abrir dirección que ha escogido
ruta= r'C:\Users\PABLO\Desktop\TFG\Pythonapi\visionapi\User-20220304T153800Z-001\User\Madrid\@TukTourMadrid'
contenido=os.listdir(ruta)
#Obtener archivos
archivos=[imagen for imagen in contenido if isfile(join(ruta, imagen))]

def algoritmo1(TitleList, LatList, LongList):
```

```
Lista1=[]
for i, j, k in zip(TitleList, LatList, LongList):

    if not j: #No contienen Latitud
        Lista1.append(i)

    else:
        Lat=j
        Long=k
        var = Lat, Long
        #Transformación de las coordenadas a dirección
        locname = loc.reverse(var)
        Lista1.append(locname)
return Lista1
```

```
def algoritmo2(TitleList, LatList, LongList):
    Lista2=[]
    cadena ="
    localizacion="

    for i in TitleList:

        if Title is None: #No contienen title
            continue

        else:
            string = i
            #eliminamos signos puntuacion
            salida = re.sub(r'^\w\s',' ',str(string))
            #eliminamos numeros
            newstring = ".join([i for i in string if not i.isdigit()])
            new_string = re.sub(r"^[a-zA-Z0-9]", " ",newstring)
            #introducimos valor en una cadena de texto
            cadena += str(new_string) #eliminar signos de puntuación
```

```
listaPalabras = cadena.split() #separamos las palabras de la cadena de texto

#creamos una lista para representar la frecuencia con la que se repiten las palabras
frecuenciaPalab = []

#recuento de las palabras que hay en la lista
for w in listaPalabras:
    frecuenciaPalab.append(listaPalabras.count(w))

#print("Cadena\n" + cadena + "\n")
#print("Lista\n" + str(listaPalabras) + "\n")
#print("Frecuencias\n" + str(frecuenciaPalab) + "\n")
#print("Pares\n" + str(list(zip(listaPalabras, frecuenciaPalab))))

for valor_a, valor_b in zip(frecuenciaPalab, listaPalabras): #obtenemos los valores en cada iteración
    if valor_a > 1: #si palabra se repite mas de dos veces

        if loc.geocode(valor_b) is None: #si la palabra repetida no tiene asignado una localizacion
            continue

        else:
            localizacion += str(valor_b)
            localizacion += (' ')

    else:
        continue

if localizacion == True:
    #print(localizacion)
    palabra = localizacion.split()
    contador = Counter(palabra)
    palabra_frecuente = contador.most_common()[0][0] #palabra más repetida
    Lista2.append(palabra_frecuente)
    return Lista2

else:
    if TitleList is None:
```

```
    palabra_frecuente = TitleList[0]
    Lista2.append(palabra_frecuente)
    return Lista2
else:
    return Lista2

def algoritmo3(TitleList, LatList, LongList, ScoreList):
    titles=[]
    coordLat=[]
    coordLong=[]
    score=[]
    coincLAT=[]
    coincLONG=[]
    coincSCORE=[]
    Lista3=[]

    for i,j,k,l in zip(TitleList, LatList, LongList, ScoreList): #anlizamos los resultados del formato
JSON
        Lat = j
        Long = k
        Score = l
        title = i

        if title is not None:
            titles.append(title)

        if not Lat: #No contienen Latitud
            continue

        if not Score:
            continue

    else:
        #introducimos valor en una Lista
        Lat1=float('{0:.2f}'.format(Lat))
        Long1=float('{0:.2f}'.format(Long))
```

```
coordLat.append(Lat1)
coordLong.append(Long1)
score.append(Score)

#Busqueda de localizaciones en la lista
if coordLat != False:
    for i, h, j in zip(range(len(coordLat)), coordLong, score):
        for j in range(len(coordLat)):
            if i != j:
                if coordLat[i] == coordLat[j]:
                    coincLAT.append(coordLat[i])
                    coincLONG.append(coordLong[i])
                    coincSCORE.append(score[i])
                else:
                    continue
            else:
                continue
        else:
            print("")

#Caso de que haya más de una localizacion coincidente
if coincSCORE == True:
    if len(coincSCORE) > 1:
        maximo=max(coincSCORE) #obtenemos el valor máximo de la lista
        lugar = coincSCORE.index(maximo) #Obtener posicion de dicho valor en la lista
        LAT1 = coincLAT[lugar]
        LONG1 = coincLONG[lugar]
        var = LAT1, LONG1
        locname = loc.reverse(var)
        Lista3.append(locname)
        return Lista3
    else:
        print("")

#Si no hay coincidentes o solo hay una
else:
```

```
for i,j,k,l in zip(TitleList, LatList, LongList, ScoreList):
```

```
    Lat = j
```

```
    Long = k
```

```
    Score = l
```

```
    title = i
```

```
    if Lat!="":
```

```
        LAT2 = j
```

```
        LONG2 = k
```

```
        var2 = LAT2, LONG2
```

```
        locname2 = loc.reverse(var2)
```

```
        Lista3.append(locname2)
```

```
        return Lista3
```

```
        break
```

```
    else:
```

```
        Lista3.append(title)
```

```
        return Lista3
```

```
        break
```

```
#FUNCION
```

```
loc = Nominatim(user_agent="GetLoc")
```

```
TitleList=[]
```

```
LatList=[]
```

```
LongList=[]
```

```
ScoreList=[]
```

```
for archivo in archivos:
```

```
    if archivo[-1]=="n":
```

```
        with open(f'{ruta}/{archivo}') as file:
```

```
            data = json.load(file)
```

```
            for foto in data['fotos']: #anlizamos los resultados del formato JSON
```

```
                Title = foto['title']
```

```
        Lat = foto['Latitud']
        Long = foto['Longitud']
        Score=foto['score']
        #Listas
        TitleList.append(Title)
        LatList.append(Lat)
        LongList.append(Long)
        ScoreList.append(str(Score))

    #Funciones
    Lista1=algoritmo1(TitleList, LatList, LongList)
    Lista2=algoritmo2(TitleList, LatList, LongList)
    Lista3=algoritmo3(TitleList, LatList, LongList, ScoreList)

else:
    continue

#print(Lista1)
#print("."*50)
#print(Lista2)
#print("."*50)
#print(Lista3)
#print("."*50)

LAT=[]
LONG=[]
elementos_rep=[]
elementos_rep1=[]
LATbor=[]
LONGbor=[]

#Obtener localizaciones finales
if Lista1 is not None:
    for i in Lista1:
        if loc.geocode(i) is None:
            continue
```

```
else:
    getLoc=loc.geocode(i)
    Lat=getLoc.latitude
    Long=getLoc.longitude
    Lat1=float('{0:.2f}'.format(Lat))
    Long1=float('{0:.2f}'.format(Long))
    LAT.append(str(Lat1))
    LONG.append(str(Long1))
else:
    print("")

if Lista2 is not None:
    for i in Lista2:
        if loc.geocode(i) is None:
            continue
        else:
            getLoc=loc.geocode(i)
            Lat=getLoc.latitude
            Long=getLoc.longitude
            Lat2=float('{0:.2f}'.format(Lat))
            Long2=float('{0:.2f}'.format(Long))
            LAT.append(str(Lat2))
            LONG.append(str(Long2))
else:
    print("")

if Lista3 is not None:
    for i in Lista3:
        if loc.geocode(i) is None:
            continue
        else:
            getLoc=loc.geocode(i)
            Lat=getLoc.latitude
            Long=getLoc.longitude
            Lat3=float('{0:.2f}'.format(Lat))
            Long3=float('{0:.2f}'.format(Long))
```

```
LAT.append(str(Lat3))
LONG.append(str(Long3))
else:
    print("")

print(LAT, LONG)
if len(LAT)==False:
    print("El algoritmo no es capaz de detectar la localización del usuario")

elif len(LAT)<=1:
    latitud=LAT[0]
    longitud=LONG[0]
    variable= latitud, longitud
    LOC=loc.reverse(variable)
    print("El usuario se encuentra en: \n", LOC)

else:
    #Buscamos variables repetidas
    counter = Counter(LAT)
    if counter.most_common() is not None:
        elementos_rep = counter.most_common()#Elementos repetidos LAT
        counter = Counter(LONG)
        elementos_rep1 = counter.most_common()#Elementos repetidos LONG
        print(elementos_rep, elementos_rep1)
        #Lat
        first=elementos_rep[0]
        lat1=first[0]
        #Long
        first1=elementos_rep1[0]
        long1=first1[0]
        #MAS DE UN ELEMENTO REPETIDO
        if len(elementos_rep) > 1:
            second=elementos_rep[1]
            lat2=second[0]
            second1=elementos_rep1[1]
```

```
long2=second1[0]
#Si tiene más de dos elementos repetidos
if len(elementos_rep) > 2:
    third=elementos_rep[2]
    lat3=third[0]
    third1=elementos_rep1[2]
    long3=third1[0]
    #probabilidades de que se repita un elemento
    prob_nor=1/(len(LAT))
    prob_1=(first[1])/(len(LAT)) #elemento más repetido
    prob_2=(second[1])/(len(LAT)) #segundo elemento más repetido
    prob_3=(third[1])/(len(LAT)) #Tercer elemento más repetido
    #Eliminamos valores repetidos de las listas LAT, LONG
    LAT= set(LAT) #dejar solo 1 de cada valor
    LONG= set(LONG)
    #convertir de nuevo a lista
    LAT=list(LAT)
    LONG=list(LONG)
    #borramos los valores que están repetidos
    LAT.remove(lat1)
    LAT.remove(lat2)
    LAT.remove(lat3)
    LONG.remove(long1)
    LONG.remove(long2)
    LONG.remove(long3)
    #REPRESENTACION MAS DE UN ELEMENTO
    VARIABLE = lat1, long1
    sitio=loc.reverse(VARIABLE)
    print("Existe un " + str(prob_1) + "de que la localización sea: " + str(sitio))
    VARIABLE2 = lat2, long2
    sitio2 = loc.reverse(VARIABLE2)
    print("Existe un " + str(prob_2) + "de que la localización sea: " + str(sitio2))
    VARIABLE3 = lat3, long3
    sitio3 = loc.reverse(VARIABLE3)
    print("Existe un " + str(prob_3) + "de que la localización sea: " + str(sitio3))
else:
```

```
#probabilidades de que se repitan dos elementos, pero no más de dos
prob_nor=1/(len(LAT))
prob_1=(first[1])/(len(LAT)) #elemento más repetido
prob_2=(second[1])/(len(LAT)) #segundo elemento más repetido
#Eliminamos valores repetidos de las listas LAT, LONG
LAT= set(LAT) #dejar solo 1 de cada valor
LONG= set(LONG)
#convertir de nuevo a lista
LAT=list(LAT)
LONG=list(LONG)
#borramos los valores que están repetidos
LAT.remove(lat1)
LAT.remove(lat2)
LONG.remove(long1)
LONG.remove(long2)
#REPRESENTACION MAS DE UN ELEMENTO
VARIABLE = lat1, long1
sitio=loc.reverse(VARIABLE)
print("Existe un " + str(prob_1) + "de que la localización sea: " + str(sitio))
VARIABLE2 = lat2, long2
sitio2 = loc.reverse(VARIABLE2)
print("Existe un " + str(prob_2) + "de que la localización sea: " + str(sitio2))
if len(LAT)!= False:
    for i, j in zip(LAT, LONG):
        VARIABLE3= i, j
        sitio3=loc.reverse(VARIABLE3)
        print("Existe un " + str(prob_nor) + "de que la localización sea: " + str(sitio3))
    else:
        print("")

else:
    #probabilidades de que se repita un elemento
    prob_nor=1/(len(LAT))
    prob_1=(first[1])/(len(LAT)) #elemento más repetido
    #segundo elemento más repetido
    #Eliminamos valores repetidos de las listas LAT, LONG
```

```
LAT= set(LAT) #dejar solo 1 de cada valor
LONG= set(LONG)
#convertir de nuevo a lista
LAT=list(LAT)
LONG=list(LONG)
#borramos los valores que están repetidos
LAT.remove(lat1)
LONG.remove(long1)
#REPRESENTACION MAS DE UN ELEMENTO
VARIABLE = lat1, long1
sitio=loc.reverse(VARIABLE)
print("Existe un " + str(prob_1) + "de que la localización sea: " + str(sitio))
if len(LAT)!= False:
    for i, j in zip(LAT, LONG):
        VARIABLE2= i, j
        sitio2=loc.reverse(VARIABLE2)
        print("Existe un " + str(prob_nor) + "de que la localización sea: " + str(sitio2))
else:
    print("")

else:
    #probabilidades de que no se repita ningún elemento
    prob_nor=1/(len(LAT))
    #si no hay elementos repetidos
    if len(elementos_rep) ==0:
        if len(LAT)>0:
            for i, j in zip(LAT, LONG):
                VARIABLE= i, j
                sitio=loc.reverse(VARIABLE)
                print("Existe un " + str(prob_nor) + "de que la localización sea: " + str(sitio))
```

ANEXO V: API GOOGLE VISION

```
import io, os
from google.cloud import vision
from geopy.geocoders import GeoNames
import pandas as pd
import metadata_parser
import json
#Obtencion coordenadas...Geoname
from geopy.geocoders import Nominatim
#abrir imagenes y obtener url
from tkinter import *
from PIL import Image, ImageTk
from tkinter import filedialog
from tkinter import Tk
from tkinter.filedialog import askopenfilename
import re
from PIL import Image, ImageDraw, ImageFont, ImageOps
import cv2

#FUNCIONES
#Función dibujar el cuadro
def draw_borders(pillow_image, bounding, color, image_size, caption="", confidence_score=0):

    width, height = image_size #establecemos anchura y altura
    draw = ImageDraw.Draw(pillow_image) #permite trabajar sobre la imagen
    draw.polygon([ #dibujamos el polígono
        bounding.normalized_vertices[0].x *
        width, bounding.normalized_vertices[0].y*height,
        bounding.normalized_vertices[1].x *
        width, bounding.normalized_vertices[1].y*height,
        bounding.normalized_vertices[2].x *
        width, bounding.normalized_vertices[2].y*height,
        bounding.normalized_vertices[3].x *
        width, bounding.normalized_vertices[3].y*height], fill=None, outline=color)
```

```
#formato de nuestra letra
font_size = width * height // 22000 if width * height > 400000 else 12
font = ImageFont.load_default()
#formato de texto
draw.text((bounding.normalized_vertices[0].x * width,
           bounding.normalized_vertices[0].y * height), font=font, text=caption, fill=color)
#ntroducimos confidence score
draw.text((bounding.normalized_vertices[0].x * width,
           bounding.normalized_vertices[0].y * height + 20), font=font, text='Confidence Score
{0:.2f}%'.format(confidence_score), fill=color)
#devolvemos imagen modificada
return pillow_image

#funcion eliminar objetos
def remove_borders(pillow_image, bounding, color, image_size, caption="", confidence_score=0):
    width, height = image_size #definimos las dimensiones
    draw = ImageDraw.Draw(img_del)
    if caption == 'Animal' or caption == 'Building' or caption == 'Monument' or caption == 'Statue'
or caption == 'Window' or caption == 'House' or caption == 'Sculpture':
        print("")
    else:
        draw.polygon([ #representamos el polígono
            bounding.normalized_vertices[0].x *
            width, bounding.normalized_vertices[0].y*height,
            bounding.normalized_vertices[1].x *
            width, bounding.normalized_vertices[1].y*height,
            bounding.normalized_vertices[2].x *
            width, bounding.normalized_vertices[2].y*height,
            bounding.normalized_vertices[3].x *
            width, bounding.normalized_vertices[3].y*height], fill=color, outline=color)
    return img_del #devolvemos la imagen

#Función borrar caras
def remove_faces(face_image, vertex, color, image_size):
    width, height = image_size
    draw = ImageDraw.Draw(face_image)
```

```
print(vertex[0])

draw.polygon([
    vertex[0].x, vertex[0].y,
    vertex[1].x, vertex[1].y,
    vertex[2].x, vertex[2].y,
    vertex[3].x, vertex[3].y], fill=color, outline=color)

return face_image

#libreria json
data = {}
data['fotos'] = []

#Interfaz con el usuario
Tk().withdraw()
filename = askopenfilename()#Abrir dirección que ha escogido
print(filename)
nombre = os.path.basename(filename)#extraer el nombre del archivo
print(nombre)

# calling the Nominatim tool
loc = Nominatim(user_agent="GetLoc")

os.environ['GOOGLE_APPLICATION_CREDENTIALS'] = r'tfgq33961014a7a01e9b80.json'
client = vision.ImageAnnotatorClient()

file_name= nombre #nombre archivo
image_path= filename #ruta

#DETECCIÓN DE IMÁGENES
#extraer info binaria img_del
with io.open(image_path, 'rb') as image_file:
    #pasar la info binaria to a content object
    content = image_file.read()
```

```
#Create our image object/ content is the binary object
image = vision.Image (content=content)
#Face detection method
response = client.face_detection (image=image)
#ejecutamos terminal response/nos da diferentes partes de la cara
#y sus coordenadas
#type(response) nos dirá el tipo de archivo con el que estamos trabajando

#Convertimos a una JSON rpsponse
#Ejecutando terminal len(faceAnnotation) da el nº de caras
#trabajar con la cara 0..... faceAnnotation[0]
#bounding_poly the area of the face
#landmarks identify different elements as eyes, nose, mouth...
#nos da información también estado de animo
faceAnnotation = response.face_annotations
#abrimos la imagen
face_image = Image.open(image_path)

for face in faceAnnotation:
    r, g, b= 0, 0, 0 #color negro
    #llamada a la función
    face_image = remove_faces(face_image, face.bounding_poly.vertices, (0, 0, 0), face_image.size)

face_image.show()
face_image.save(image_path)

#DETECCIÓN OBJETOS
#extraer info binaria
with io.open(image_path, 'rb') as image_file:
    #pasar la info binaria to a content object
    content = image_file.read()

#control my image object
image=vision.Image(content=content)
response=client.object_localization(image=image)
#response nos dará el nombre, score, bounding_poly
```

```
localized_object_annotations = response.localized_object_annotations
#localized_object_annotations nos dará el resultado

df = pd.DataFrame(columns=['name', 'score']) #2 columnas: 1 nombre / 2 score asociado
for obj in localized_object_annotations: #ciclo for para cada objeto localizado
    df = df.append( #añade la información en modo diccionario
        dict(name= obj.name,
            score= obj.score),
        ignore_index=True
    )

pillow_image= Image.open(image_path)
img_del= Image.open(image_path)

for obj in localized_object_annotations:
    r, g, b= 0, 0, 0 #color negro
    pillow_image = draw_borders(pillow_image, obj.bounding_poly, (r, g, b), pillow_image.size,
    obj.name, obj.score)
    img_del = remove_borders(img_del, obj.bounding_poly, (r, g, b), pillow_image.size, obj.name,
    obj.score)

#enseñar imagen
pillow_image.show()
img_del.show()

#guardar imagen
img_del.save(image_path)

#CONTINUA CON LA DETECCIÓN DE PAISAJES
#control my image object
image=vision.Image(content=content)
response=client.landmark_detection(image=image)
#print(response) ventana de comandos nops da description con el nombre
#confidence score y bounding poly es ese area para detectar
```

```
#por ultimo nos daría esa locations coordenadas de donde estaría

landmarks = response.landmark_annotations
#len(landmarks) nos dara el numero de landscapes que detecta
print(response.landmark_annotations)
for landmark in landmarks:
    description = landmark.description
    score =landmark.score
    loc= landmark.locations
    data['fotos'].append({
        'url': ",
        'description': ",
        'title': landmark.description,
        'score': landmark.score,
        'Latitud': ",
        'Longitud': ",
    })

#guardar la info como una tabla
df = pd.DataFrame(columns=['description','score','bounding_poly'])
for landmark in landmarks:
    df =df.append(
        dict(
            description = landmark.description,
            locations = landmark.locations,
            score = landmark.score,
        ),
        ignore_index=True
    )
print(df)
print('.'*50)

#FUNCION QUE NOS VA A DAR LAS WEBS
file_name= nombre
image_path= filename
#extraer info binaria
```

```
with io.open(image_path, 'rb') as image_file:
    #pasar la info binaria to a content object
    content = image_file.read()

image = vision.Image(content=content)
response=client.web_detection(image=image)
#print(response) description texto asociado con la imagen para su busqueda
#score...nos da la confianza de que sea esa opcion
#también en la parte inferior nos da url relacionada

#create web dectection response
web_detection = response.web_detection

#print(web_detection.best_guess_labels)
#print(web_detection.full_matching_images)
#print(web_detection.pages_with_matching_images)
#print(web_detection.partial_matching_images)
#print(web_detection.visually_similar_images)

if web_detection.pages_with_matching_images:
    print("\n{ } Pages with matching images found:".format(
        len(web_detection.pages_with_matching_images)))

    for page in web_detection.pages_with_matching_images:
        url_prueba=page.url

        #obtencion megadata
        try:
            if metadata_parser.MetadataParser(url_prueba) is None:
                #page2=metadata_parser.MetadataParser(url_prueba)
                #title1=page2.get_metadatas('title')
                #description1=page2.get_metadatas('description')
                #print(page2.get_metadatas('title'))
                #print(page2.get_metadatas('description'))
                print("\n\tPage url : {}".format(page.url))
                #separacion de palabras
```

```
#creacion json
```

```
if loc.geocode(url_prueba) is None:
```

```
    data['fotos'].append({  
        'url': page.url,  
        'description': "",  
        'title': "",  
        'score': "",  
        'Latitud': "",  
        'Longitud': "",  
    })
```

```
else:
```

```
    # entering the location name  
    getLoc = loc.geocode(page.url)  
    # printing address  
    print(getLoc.address)  
    # printing latitude and longitude  
    Lat1=getLoc.latitude  
    Lon1=getLoc.longitude  
    print("Latitude = ", getLoc.latitude, "\n")  
    print("Longitude = ", getLoc.longitude)  
    data['fotos'].append({  
        'url': page.url,  
        'description': "",  
        'title': "",  
        'score': "",  
        'Latitud': Lat1,  
        'Longitud': Lon1,  
    })
```

```
#obtencion megadata
```

```
else:
```

```
    page2=metadata_parser.MetadataParser(url_prueba)  
    title1=page2.get_metadatas('title')  
    description1=page2.get_metadatas('description')
```

```
print(page2.get_metadatas('title'))
print(page2.get_metadatas('description'))
print("\n\tPage url : {}".format(page.url))
#separacion de palabras
#creacion json

if loc.geocode(url_prueba) is None:
    data['fotos'].append({
        'url': page.url,
        'description': description1,
        'title': title1,
        'score': " ",
        'Latitud': " ",
        'Longitud': " ",
    })

else:
    # entering the location name
    getLoc = loc.geocode(page.url)
    # printing address
    print(getLoc.address)
    # printing latitude and longitude
    Lat1=getLoc.latitude
    Lon1=getLoc.longitude
    print("Latitude = ", getLoc.latitude, "\n")
    print("Longitude = ", getLoc.longitude)
    data['fotos'].append({
        'url': page.url,
        'description': description1,
        'title': title1,
        'score': " ",
        'Latitud': Lat1,
        'Longitud': Lon1,
    })

except:
    print("No se ha conseguido obtener metadata\n")
```

```
try:
    if page.full_matching_images:
        print('\t{} Full Matches found: {}'.format(
            len(page.full_matching_images)))

    for image in page.full_matching_images:
        print('\t\tImage url : {}'.format(image.url))

    if page.partial_matching_images:
        print('\t{} Partial Matches found: {}'.format(
            len(page.partial_matching_images)))

    for image in page.partial_matching_images:
        print('\t\tImage url : {}'.format(image.url))

    print('.*50)

#score and description de las webs encontradas
if web_detection.web_entities:
    print('\n{} Web entities found: {}'.format(
        len(web_detection.web_entities)))

    for entity in web_detection.web_entities:
        print('\n\tScore    : {}'.format(entity.score))
        print(u'\tDescription: {}'.format(entity.description))
        #almacenamiento información en json

    if loc.geocode(entity.description) is None:
        data['fotos'].append({
            'url': "",
            'description': entity.description,
            'title': "",
            'score': entity.score,
            'Latitud': "",
            'Longitud': "",
```

```
    })

    else:
        # entering the location name
        getLoc = loc.geocode(entity.description)
        # printing address
        print(getLoc.address)
        # printing latitude and longitude
        Lat1=getLoc.latitude
        Lon1=getLoc.longitude
        print("Latitude = ", getLoc.latitude, "\n")
        print("Longitude = ", getLoc.longitude)
        data['fotos'].append({
            'url': image.url,
            'description': entity.description,
            'title': "",
            'score': entity.score,
            'Latitud': Lat1,
            'Longitud': Lon1,
        })

except:
    print("No coincidencias con full matching o partial matching \n")

#crear file JSON/eliminar signos de puntuación
s = nombre
out = re.sub(r'[^\w\s]','',s) #eliminamos signos de puntuación
print(out)#añadimos .json
archivo= out +'.json'
print(archivo)

with open(f'{filename}{archivo}', 'w') as file:
    json.dump(data, file, indent=4)

#lectura
f=open(f'{filename}{archivo}', "r")
c=f.read()
```

```
js=json.loads(c)
```