laSalle
UNIVERSITAT RAMON LLULL

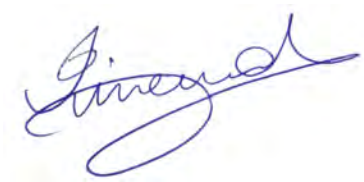
Escola Universitària d'Enginyeria Tècnica de Telecomunicació La Salle

Trabajo Final de Máster

Máster Universitario en Ciencia de Datos

Reconocimiento de Imágenes con Landmarks

Alumno



Jimena Martínez Decia

Profesor Ponente

Xavier Sevillano Domínguez

ACTA DEL EXÁMEN
DEL TRABAJO FINAL DE MÁSTER

Reunido el Tribunal calificador en el día de la fecha, el alumno

D. **Jimena Martínez Decia**

expuso su Trabajo Final de Máster, el cual trató sobre el tema siguiente:

Reconocimiento de Imágenes con Landmarks

Finalizada la exposición y atendidas por parte del alumno las objeciones formuladas por los Srs. miembros del tribunal, éste valoró el mencionado Trabajo con la calificación de

Barcelona,

VOCAL DEL TRIBUNAL VOCAL DEL TRIBUNAL

PRESIDENTE DEL TRIBUNAL

Abstract

El presente proyecto introduce un nuevo dataset para reconocimiento de landmarks (puntos turísticos) con 184.732 imágenes en formato JPG divididas en 17 clases. En primer lugar se realiza una introducción al problema de la visión por computador, sus características y contexto.

A continuación se desarrollan y justifican las decisiones tomadas con respecto al diseño del proyecto, como ser: hardware y software utilizado, criterios de construcción del dataset y técnicas de validación.

En la sección de experimentación y resultados se explican las técnicas utilizadas para crear el modelo, las cuales incluyen la extracción de características a través de transfer learning y la creación de una red neuronal convolucional desde cero.

Finalmente se muestran las conclusiones y enseñanzas obtenidas y se plantean los lineamientos futuros a seguir.

Indice

1 Introducción y objetivos

1.1 El problema

La inmensa cantidad de imágenes y videos disponibles hoy en día, gracias al avance en el hardware y al boom de los móviles con cámara digital, ha instado a la comunidad científica y a la industria a volcarse hacia el campo de la visión por computador.

La visión por computador es un campo interdisciplinario que abarca la extracción automática, el análisis y la comprensión de la información que se puede obtener a partir de una imagen o una secuencia de estas. Implica el desarrollo de una base teórica y algorítmica para lograr una comprensión visual automática, de la misma forma que el sistema visual humano lo puede hacer. Comprender en este contexto, significa la transformación de imágenes visuales en descripciones del mundo que permitan interactuar con otros procesos y provocar una acción adecuada.

En este proyecto en particular se trabajara sobre un aspecto específico de la visión por computador: el reconocimiento de puntos de referencia turísticos, los cuales llamaremos de ahora en adelante landmarks.

1.2 La aplicación

La visión por computador tiene un amplio espectro de aplicaciones: robótica, autos sin conductor, medicina, drones, seguridad y vigilancia, reconocimiento facial, detección de emociones, etc. La clasificación y reconocimiento de imágenes son problemas fundamentales sobre los cuales aún queda mucho por resolver, principalmente cuando se trata de datasets complejos.

Los análisis y modelos que resulten del presente proyecto podrían ser utilizados en buscadores, redes sociales o diversas aplicaciones en que usuarios puedan por ejemplo subir fotos o buscar lugares y destinos turísticos. Pero además de las aplicaciones específicas en el ámbito de landmarks, los resultados obtenidos podrían ser extendidos a otros problemas de visión de computador.

1.3 El objetivo

El objetivo principal del proyecto es su aprobación académica con el fin de obtener el título de Master Universitario en Data Science. Asimismo, el trabajo se orienta a la aplicación de los conocimientos adquiridos durante el transcurso del Master, así como la profundización de los mismos, principalmente en el área de Computer Vision y Deep Learning.

1.4 Estado del Arte

En los últimos años, deep learning ha obtenido un gran éxito en una amplia variedad de dominios. Los resultados experimentales muestran que su performance ha alcanzado el estado del arte en comparación con los métodos tradicionales de machine learning en los campos de: procesamiento de imágenes, visión por computador, reconocimiento de voz, traducción automática, arte, imágenes médicas, procesamiento de información médica, robótica y control, bioinformática, procesamiento del lenguaje natural, ciberseguridad y muchos otros (Alom et al., 2018).

La diferencia clave entre los métodos tradicionales de Machine Learning y el Deep Learning es la forma en que las características son extraídas. Los enfoques tradicionales de Machine Learning realizan la extracción de características a mano con la aplicación de diferentes algoritmos, como ser: SIFT (Scale Invariant Feature Transform), SURF (Speeded Up Robust Features), HOG (Histogram Oriented Gradient), LBP (Local Binary Pattern), etc. A continuación, se aplican algoritmos de clasificación sobre las características extraídas, como por ejemplo: SVM (Support Vector Machine), RF (Random Forest), PCA (Principle Component Analysis), KPCA (Kernel PCA), LDA (Linear Decrement Analysis), FDA (Fisher Decrement Analysis) y muchos más. Además, generalmente se utilizan también otros enfoques para potenciar, en los cuales se aplican varios algoritmos de aprendizaje sobre las características y se toma una decisión de acuerdo a los múltiples resultados obtenidos (Alom et al., 2018).

En cambio, en el caso de Deep Learning, las características son aprendidas automáticamente y representadas jerárquicamente en distintos niveles, siendo esto una de sus características más ventajosas en comparación con el Machine Learning tradicional (Alom et al., 2018). En resumen, podemos destacar las principales ventajas de Deep Learning en 4 puntos:

1) Enfoque de aprendizaje universal: Se puede aplicar a casi cualquier dominio.
2) Enfoques robustos de aprendizaje: Al aprender las características de forma automática, las variaciones naturales de los datos se aprenden de forma automática.
3) Generalización: El mismo enfoque de aprendizaje se puede aplicar en diferentes dominios o con distintos datos.
4) Escalabilidad: Es altamente escalable, permitiendo agregar capas a la red sin mayor dificultad (Alom et al., 2018).

Existen diferentes técnicas en Deep Learning para el aprendizaje supervisado: DNN (Deep Neural Networks), CNN (Convolutional Neural Networks), RNN (Recurrent Neural Networks) incluyendo LSTM (Long Short Term Memory) y GRU (Gated Recurrent Units). Las redes convolucionales tienen varias ventajas con respecto a las profundas, fundamentalmente en lo que respecta a la visión por computador. Son más similares al sistema de procesamiento humano y su estructura está altamente optimizada para procesar imágenes 2D y 3D. La capa de max pooling es efectiva absorbiendo variaciones de forma y además este tipo de redes tiene significativamente menos parámetros que las profunda. Sobre todo, sufren menos del problema de gradiente descendiente, ya que son entrenadas con un algoritmo de

aprendizaje basado en gradiente, debido a esto pueden producir pesos altamente optimizados (Alom et al., 2018).

ImageNet es una ontología de imágenes a gran escala, construida sobre la columna vertebral de la estructura de WordNet. ImageNet pretende poblar la mayoría de los 80.000 synsets (conjuntos de sinónimos cognitivos) con un promedio de entre 500/1000 imágenes (Jia Deng et al., 2009). En la actualidad contiene más de 14 millones de URLs de imágenes anotadas de forma manual, pertenecientes a 20 mil categorías distintas. La base de datos con las URLs de las imágenes de terceros, está disponible de forma gratuita. Desde el 2010, el proyecto de ImageNet realiza un concurso anual de software llamado ILSVRC (ImageNet Large Scale Visual Recognition Challenge), donde varios programas de software compiten sobre un subconjunto de ImageNet con mil clases que no se superponen, para clasificar y detectar objetos y escenas correctamente ("ImageNet Wikipedia," n.d.).

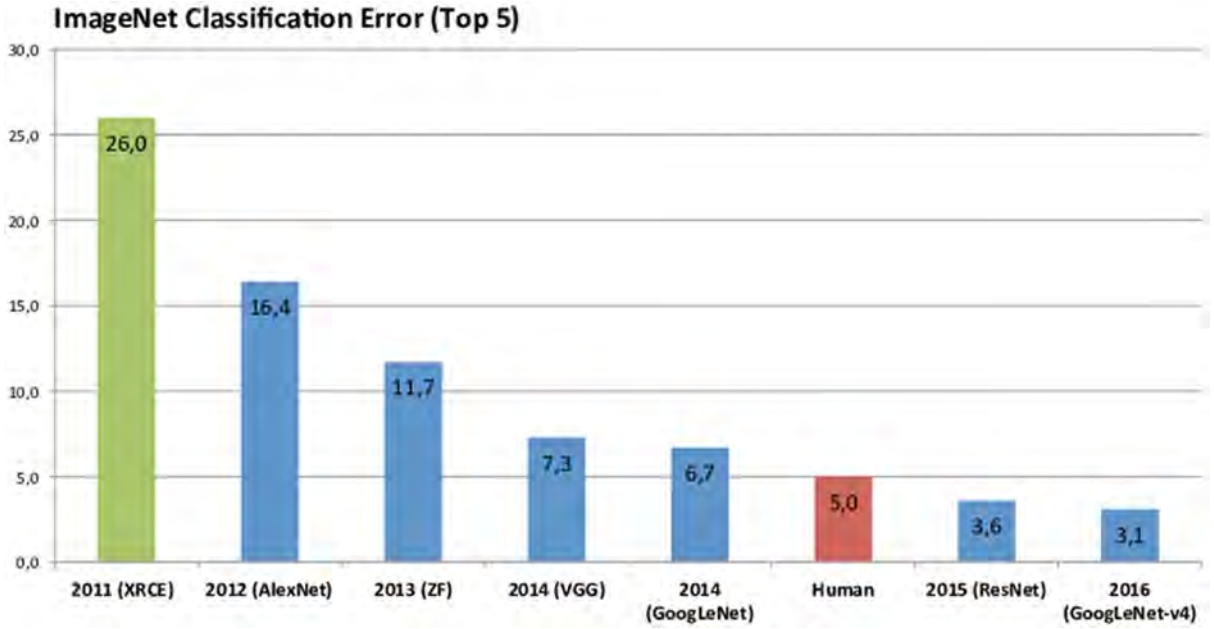


Figura 1: Porcentaje de error de clasificación para el top 5 ("ImageNet," n.d.)

En 2012 AlexNet (Krizhevsky, Sutskever, & Geoffrey E., 2012) gano la competición ILSVRC, siendo la primera vez que se utilizó Deep Learning. AlexNet logro alcanzar el estado del arte en cuanto a precisión en el reconocimiento de imágenes, superando a los métodos tradicionales de machine learning y visión por computador. Esto significó un gran avance para el machine learning y la visión por computador y marco el punto en la historia donde el interés por el Deep learning comenzó a crecer rápidamente (Alom et al., 2018). Otro hito importante en la historia del Deep Learning obtenido gracias a ImageNet, fue en 2015 cuando ResNet (Wu, Zhong, & Liu, 2017) gano la competición con un porcentaje de error de clasificación para el top 5 menor al 5% obtenido por los seres humanos.

2 Diseño del proyecto

2.1 Hardware

La primera decisión a tomar al inicio del proyecto fue con respecto al hardware a utilizar para realizarlo, ya que para trabajar con imágenes y Deep Learning es imprescindible contar con GPU. La computación acelerada por GPU, se refiere al uso de una unidad de procesamiento de gráficos (GPU) junto con una CPU, lo cual ofrece un rendimiento sin precedentes ya que descarga las partes de la aplicación que consumen gran cantidad de computo a la GPU, mientras que el resto del código se ejecuta en la CPU (“NVidia,” n.d.).

La diferencia entre la GPU y la CPU está en cómo procesan las tareas, mientras que las CPUs constan de unos pocos núcleos optimizados para el procesamiento secuencial en serie, las GPUs constan de miles de núcleos más pequeños y eficientes diseñados para manejar múltiples tareas simultáneamente (“NVidia,” n.d.).

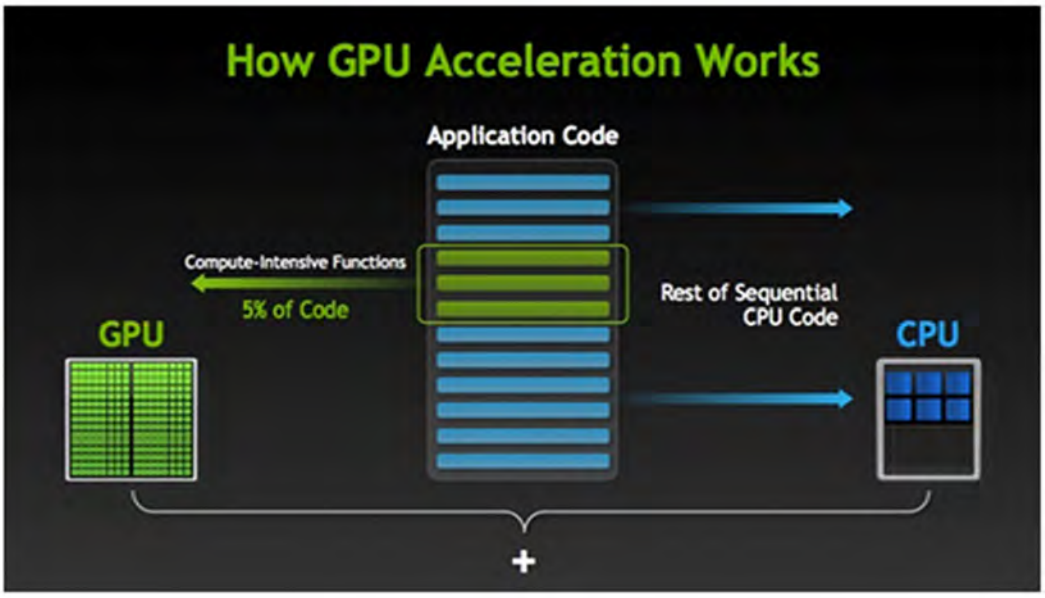


Figura 2: Funcionamiento GPU y CPU (“NVidia,” n.d.).

En la actualidad la única empresa dedicada a comercializar GPUs especializadas para Machine Learning es NVIDIA. Fundada en 1993 por Jensen Huang, Chris Malachowsky y Curtis Priem, en 1999 crea la GPU, la cual inicialmente fue utilizada en el campo de los videojuegos hasta que en el 2015 NVIDIA incursiona en el campo del Deep Learning.

Se realizó un análisis comparativo de las dos opciones disponibles en la actualidad para cumplir con el requerimiento de hardware, comprar un nuevo ordenador o utilizar una máquina virtual:

	Máquina Virtual	**Comprar Ordenador**
Costo Inicial	Nulo	Alto (de 1200 euros en adelante dependiendo de las prestaciones)
Costo Mensual Procesamiento	Se paga por la cantidad de horas que se utiliza (de 1 euro la hora en adelante dependiendo de las prestaciones)	No tiene
Costo Mensual Almacenamiento	Se paga por el espacio que se utiliza de forma mensual, sin importar la cantidad de horas que se utilizó la máquina	No tiene
Escalabilidad	Fácil	Difícil
Necesidad de conexión	Se precisa buena conexión a internet para trabajar	No se precisa conexión a internet para trabajar
Software	Instalado e incluido en los costos	Se debe instalar y aunque no está incluido el software que se va a utilizar no requiere licencia

Tabla 1: Máquina Virtual vs Ordenador Físico

En un inicio se optó por la opción de la máquina virtual fundamentalmente por dos motivos, la ausencia de costo inicial y la escalabilidad, ya que al inicio del proyecto no se contaba con el conocimiento exacto de cuales iban a ser los requerimientos de hardware, por lo que este último punto fue importante a la hora de decidir. La máquina virtual seleccionada fue la Data Science Virtual Machine de Azure, debido a que cuenta con un entorno pre configurado especialmente para proyectos de Data Science e Inteligencia Artificial y además ofrecía un mes gratis de prueba.

Figura 3: Data Science Virtual Machine Azure (“Microsoft Virtual Machines,” n.d.)

Luego de trabajar dos meses y medio con la máquina virtual se decidió adquirir un ordenador por los siguientes motivos:

- Costos: Los costos resultaron elevados por dos motivos, las imágenes precisaban una importante capacidad de almacenamiento, lo que generaba un costo base de 80 euros mensuales y las horas necesarias para entrenar las redes neuronales eran muchas, lo que sumaba costos variables elevados también.
- Conexión: No siempre se disponía de buena conexión a internet lo cual muchas veces dificultaba el trabajo.

El ordenador seleccionado fue un Portatil Gaming-Asus GL502VS-GZ289T con las siguientes características:

- Procesador: Intel Core i7-7700HQ
- RAM: 32GB
- Disco HDD: 1TB
- Disco SSD: 512GB
- GPU: NVIDIA GeForce GTX1070

2.2 Software

El lenguaje de programación seleccionado para desarrollar el proyecto fue Python y la plataforma para Deep Learning TensorFlow con el framework de Keras, a continuación se detallan los motivos para cada una de las elecciones.

Python: Es un lenguaje de programación interpretado, multiparadigma (soporta programación orientada a objetos, programación imperativa y programación funcional), usa tipado dinámico y es multiplataforma (“Python Wikipedia,” n.d.). Sus principales ventajas son:

- Licencia de código abierto
- Incluye un modo interactivo que permite realizar pruebas de forma sencilla
- Amplia comunidad, lo que significa gran cantidad de documentación, conferencias y workshops y miles de módulos desarrollados por terceros
- Licencia de código abierto, por lo que es libremente utilizable y distribuible, incluso para uso comercial
- Amigable y fácil de aprender, una de sus características es el uso de palabras donde otros lenguajes utilizan símbolos, por ejemplo, los operadores lógicos
- Gran biblioteca para diversidad de tareas

TensorFlow: Biblioteca de software para cálculos numéricos de alto rendimiento. Sus principales ventajas son:

- Código abierto

- Arquitectura flexible: Permite una fácil implementación en variedad de plataformas (CPU, GPU, TPU) y desde escritorios hasta clústeres de servidores y dispositivos móviles.
- Desarrollado por el departamento de Inteligencia Artificial de Google, por lo que cuenta con un sólido respaldo (“Tensorflow,” n.d.)

Keras: Framework para redes neuronales de código abierto escrito en Python. Sus principales ventajas son:

- Prioriza la experiencia del desarrollador, por lo que es fácil de aprender y usar
- Fuerte adopción de la industria y la comunidad de investigadores
- Permite convertir de manera sencilla los modelos en productos
- Admite múltiples backends: TensorFlow, CNTK, Theano
- Compatible con multiples GPUs
- Soporta entrenamiento distribuido
- Apoyado por las principales compañias en el ecosistema de Deep Learning, como ser Google, Microsoft, NVIDIA y Amazon.

2.3 Construcción Dataset

El puntapié inicial del proyecto fue la publicación en Kaggle de la competencia llamada Google Landmark Recognition Challenge. La competencia, lanzada por Google, invitaba a construir modelos que hiciesen posible reconocer de forma correcta los landmarks en el dataset de imágenes publicado. En comparación con los datasets existentes en la actualidad este nuevo dataset es mucho más grande, contiene una mayor diversidad de landmarks con imágenes más realistas y desafiantes (“Kaggle,” n.d.).

El dataset de Google fue construido en base a un algoritmo diseñado para buscar landmarks en la web, que combina distintas fuentes de información y técnicas como ser etiquetado GPS de las imágenes, información de sitios dedicados al turismo y comparación de imágenes (Zheng et al., 2009).

Debido a las restricciones en la distribución de los archivos reales, el dataset contiene las URLs que apuntan a cada imagen desde donde pueden ser descargadas. Dado que las imágenes no están directamente bajo el control de los organizadores de la competición, un pequeño porcentaje de ellas podrá no estar disponible.

El primer aspecto a tomar en cuenta fue con que compresión iban a ser descargadas, para lo cual se estudiaron distintas investigaciones, llegando a la conclusión que las redes son sorprendentemente resistentes a las distorsiones de compresión JPEG. Solo a niveles de calidad muy bajos, el rendimiento comienza a disminuir. Esto significa que podemos estar razonablemente seguros de que las redes profundas funcionarán bien en datos comprimidos (Dodge & Karam, 2016).

Se descargaron únicamente los datos de test, de las 1.225.029 de rutas que contenía el archivo pudieron ser descargadas efectivamente 1.218.620 imágenes, todas se guardaron en formato JPG con calidad de compresión del 90%.

En primer lugar se realizó un análisis exploratorio del dataset a alto nivel del cual se extrajeron los siguientes datos y conclusiones:

El dataset está compuesto por 14951 ids de landmarks distintos, con la siguiente distribución:

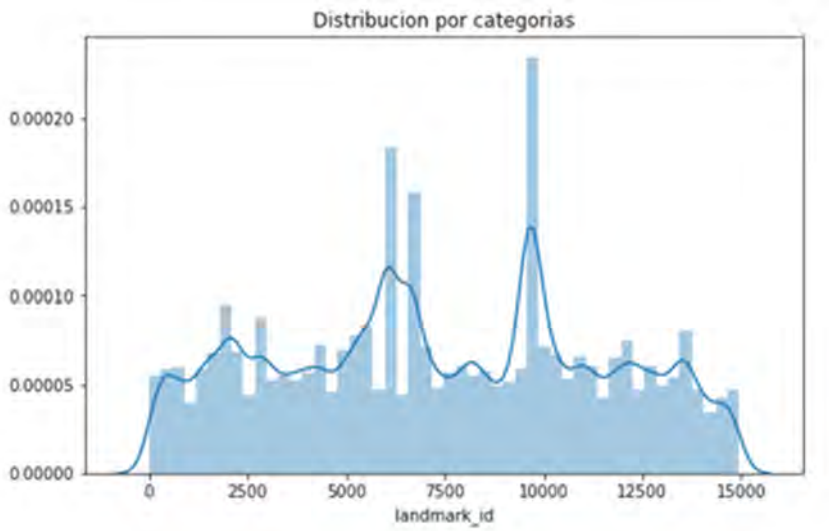


Figura 4: Distribución de los ids de los landmarks

Y los siguientes datos estadísticos:

count	14951,00
mean	81,93
std	707,23
min	1,00
25%	6,00
50%	14,00
75%	46,00
max	50337,00

Tabla 2: Datos estadísticos dataset

Como se puede apreciar de acuerdo a la distribución en la Figura 4 y los datos estadísticos de la Tabla 2, las diferencias entre las cantidades de imágenes en las diferentes clases es muy grande, la clase que más imágenes tiene alcanza 50337, mientras que la que menos tiene una. Por otro lado podemos notar esto también si vemos los datos de media y desviación estándar, ya que la primera es 81,93, mientras que la segunda la supera ampliamente con 707,23.

Otros datos a destacar con respecto a la distribución de las imágenes entre los diferentes landmarks son: 159 landmarks contienen solo una imagen, la suma de imágenes de los 1000 landmarks con mayor cantidad es de 815347, mientras la suma de la cantidad de imágenes de los 1000 landmarks con menos cantidad es de solamente 2391, además un 43% de los landamarks tienen menos de 10 ocurrencias.

Por supuesto estos datos nos muestran parte de la complejidad del dataset, ya que vamos a contar con un gran porcentaje de clases que tendrán muy pocas imágenes para entrenar la red en su reconocimiento. Además que la cantidad de clases diferentes es muy alta y con grandes diferencias de información entre ellas. En la siguiente Figura 5, observaremos la cantidad de imágenes en las 20 clases con mayor cantidad.

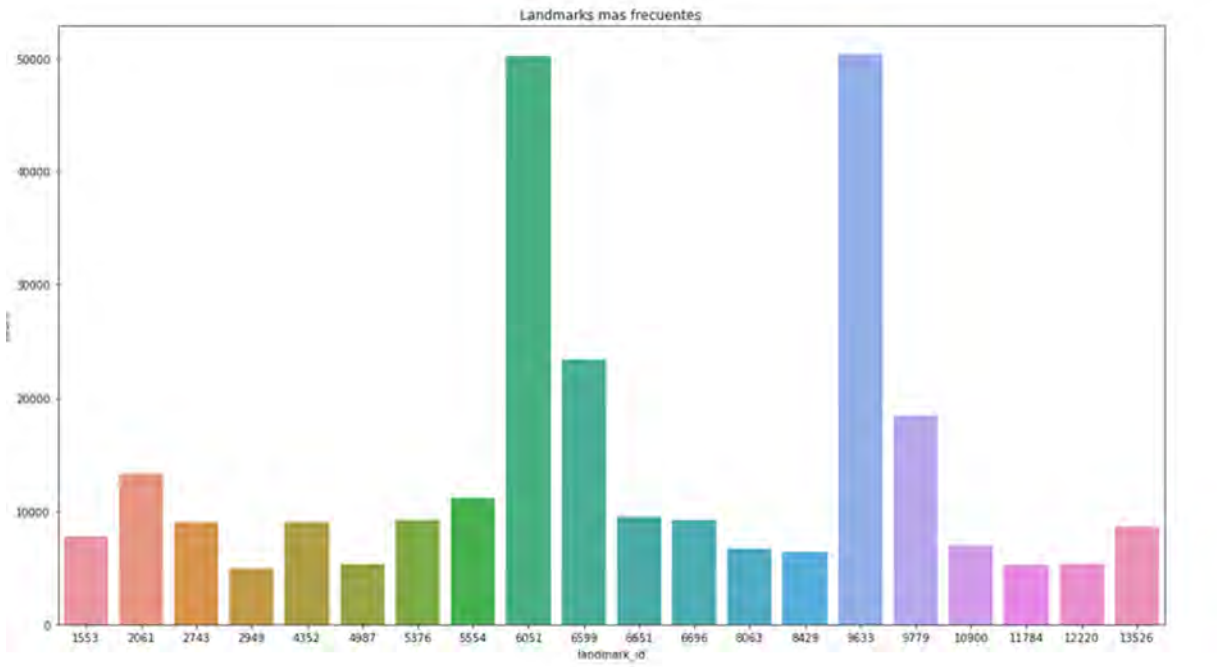


Figura 5: Ids de landmarks más frecuentes

Como se aprecia en la gráfica, hasta en las primeras 20 clases la diferencia es grande, a continuación miraremos algunas de las imágenes de las 3 primeras:

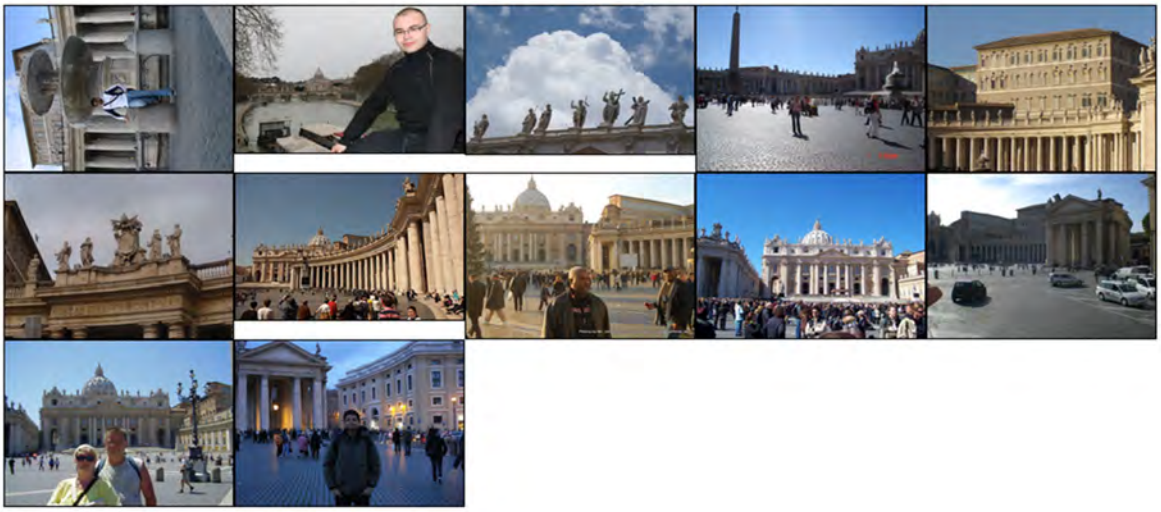


Figura 6: Basílica de San Pedro

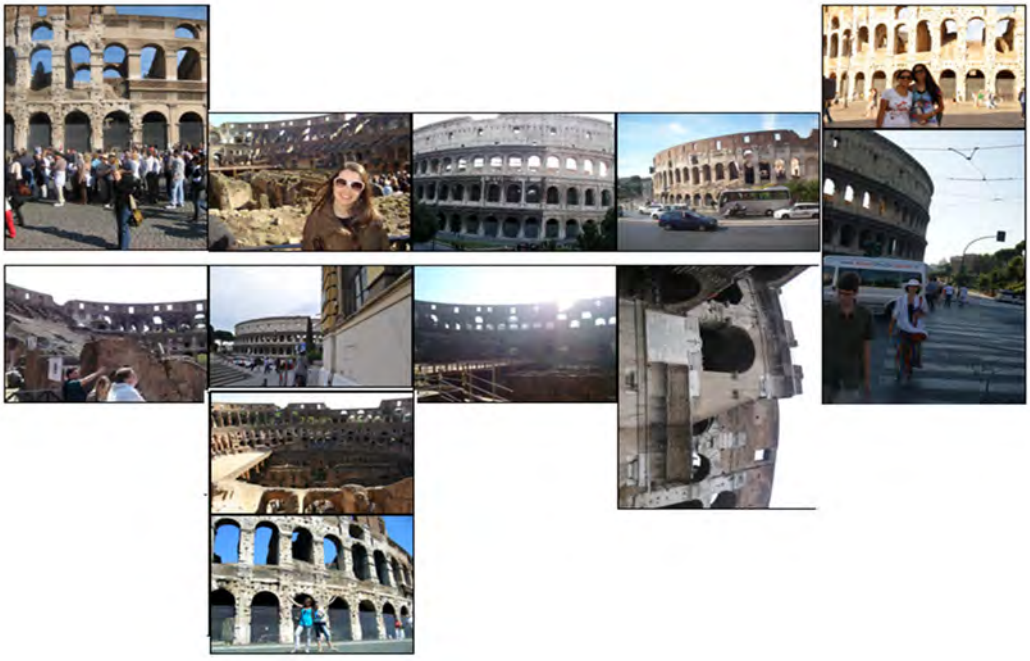


Figura 7: Coliseo Romano

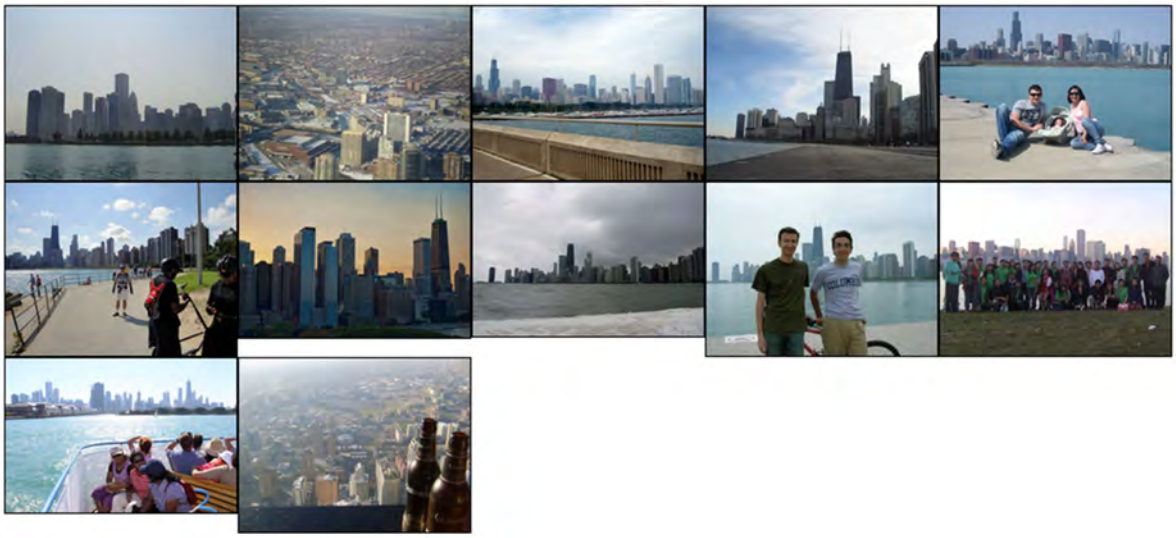


Figura 8: Willis Tower

Mirando estas primeras imágenes se puede apreciar la diversidad de las mismas, algunas son desde adentro de los landmarks, otras de afuera a distintas distancias, con gente posando o sin gente posando, de diferente calidad, con diferente luminosidad e inclusive en diferentes momentos del día o año. Por supuesto todos estos factores incrementan también la complejidad para su reconocimiento.

Otro aspecto observado fue la ubicación de los distintos landmarks, pero como se puede observar en la Figura 9, están distribuidos por todo el mundo por lo que un principio no nos aporta información extra.

Figura 9: Ubicación Landmarks (Noh, Araujo, Sim, Weyand, & Han, 2017)

Por último, miramos los sitios web desde donde provenían las imágenes y como se puede apreciar en la Figura 10, en su mayoría son provenientes de cuentas de Google.

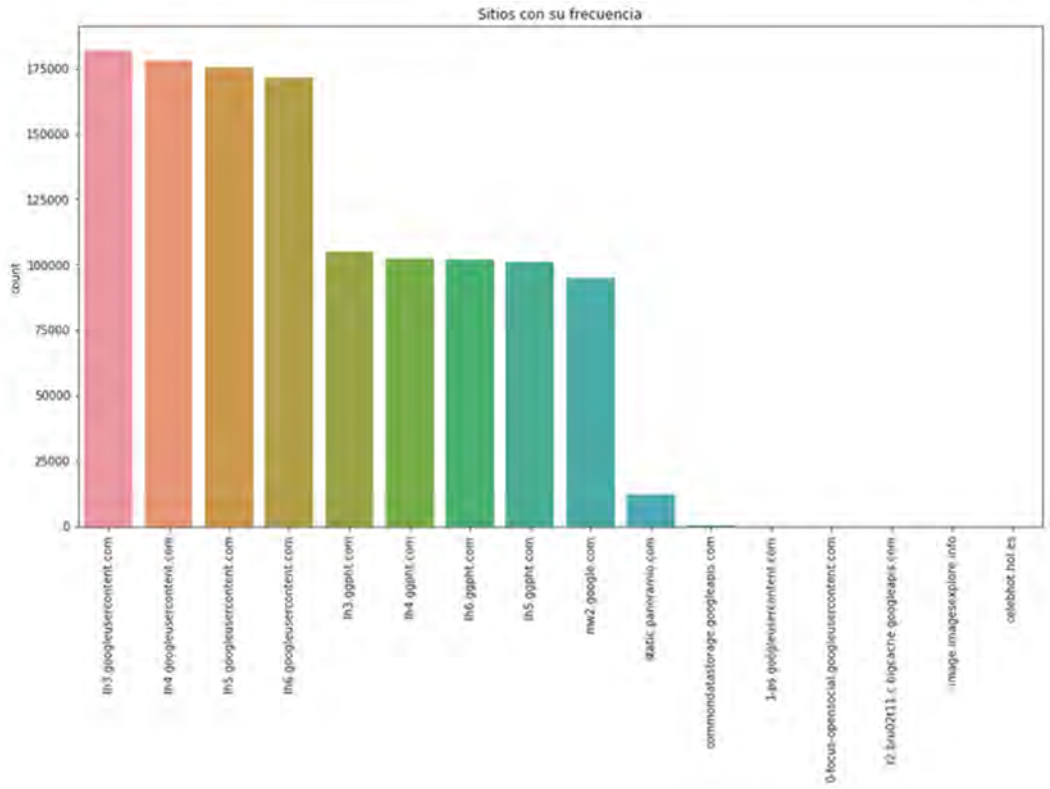


Figura 10: Sitios webs de donde provienen las imágenes

En una segunda instancia antes de comenzar con el diseño del modelo, se realizó un segundo análisis exploratorio del dataset más exhaustivo, intentando mirar manualmente la mayor cantidad posible de imágenes. A través de este análisis se recopilaron las siguientes observaciones:

- Importante cantidad de imágenes mal clasificadas

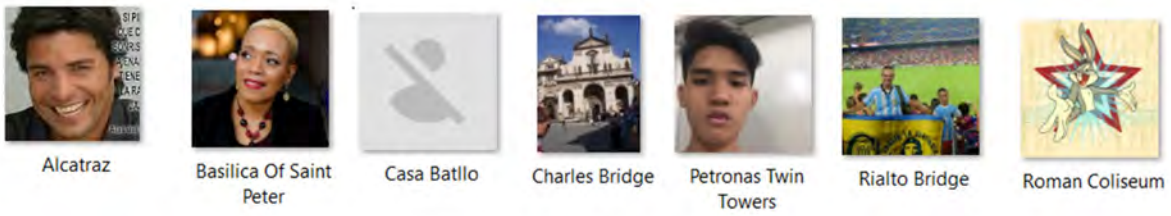


Figura 11: Ejemplo de imágenes mal clasificadas en el dataset de Google

- Gran cantidad de clases que no sirven por muy baja cantidad de imágenes
- Clases agrupadas por distintos criterios: landmarks (el Coliseo, la Torre Eiffel, etc), ciudades (Florencia, Chicago, etc), festivales (New Orleans Jazz & Heritage Festival, etc)
- Imágenes muy heterogéneas, por ejemplo de dentro y fuera de los landmarks, de landmarks naturales y construidos por el ser humano, etc.

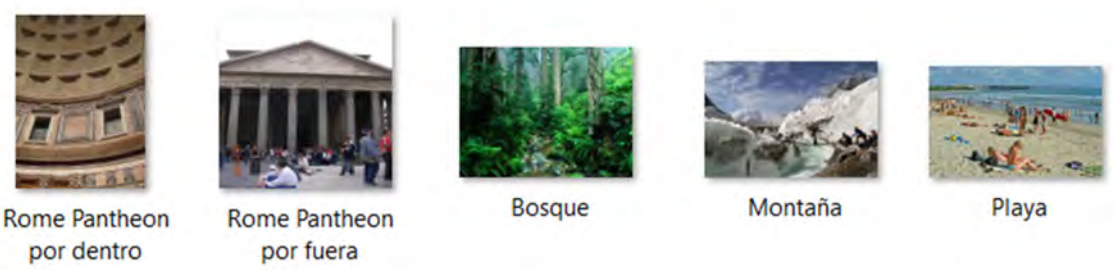


Figura 12: Heterogeneidad del dataset

Varios de estos puntos pueden ser explicados por el hecho de que el dataset no fue etiquetado manualmente sino a través de un algoritmo, por lo que por ejemplo una imagen tomada en un puente o desde el puente va a tener aproximadamente las mismas coordenadas GPS, aunque en una se muestre el puente y en otra no. Debido a estos aspectos el dataset tal cual esta no resulta ser el mejor para trabajar en un modelo que obtenga una alta precisión reconociendo landmarks, probablemente los objetivos perseguidos por la competición sean otros, por lo que se decidió crear un dataset específico para este proyecto.

Para construir el dataset se aplicaron los siguientes criterios:

- Las imágenes deben contener landmarks (puntos turísticos) reconocidos
- Estos landmarks deben de ser de construcción humana, se excluyen los naturales, como por ejemplo: playas, cascadas, montañas, volcanes, etc
- La imagen debe mostrar el exterior o parte de este, se excluyen las imágenes del interior. Esta decisión se tomó en base a 3 motivos: es mayor la cantidad de imágenes exteriores, las imágenes interiores tienen mayor complejidad para reconocer (en landmarks como el Louvre o el Museo del Hermitage pueden obtenerse miles de imágenes completamente distintas), desde el punto de vista de una posible aplicación comercial futura tiene más aplicabilidad.
- Ante dudas en el etiquetado de una imagen se tomaron en cuenta los siguientes criterios: si el ojo humano era capaz de reconocerlo intentando ser objetivo (ya que el etiquetador debía manejar cierta subjetividad porque muchas de las imágenes ya estaban previamente etiquetadas), si el buscador de Google lo podía reconocer y en el caso de que la imagen tuviese más de un landmark se etiquetaba con el que tuviese mayor preponderancia.
- Para todos los landmarks se buscaron imágenes que lo muestren en distintas condiciones: noche y día, distintos ángulos, diferentes partes del landmark, oclusiones, variadas condiciones climáticas, diferentes momentos en el tiempo, distinta iluminación, calidad y fuente de las imágenes variadas y distintas distancias al punto.
- Por último se tuvo en cuenta el tamaño de la imagen, las imágenes muy pequeñas no fueron incluidas en el dataset.

Finalmente para su construcción se analizaron más de 500.000 imágenes manualmente, lo cual llevo un total de 197 horas hombres, de 5 fuentes distintas:

- El dataset de Google publicado en la competición de Kaggle
- Paris: Dataset con fotos de distintos landmarks de Paris, utilizado clásicamente en los proyectos que trabajan sobre imágenes de landmarks. A pesar de ser mucho más pequeño que el dataset de Google y de ya haber sido utilizado en múltiples proyectos, se encontró también un alto porcentaje de imágenes mal clasificadas, por ejemplo imágenes de la copia de la Torre Effeil en Las Vegas u otros Arcos del Triunfo distintos al original de Paris ubicado en la Avda de los Campos Eliseos. Fue descargado de: http://www.robots.ox.ac.uk/~vgg/data/parisbuildings/
- Flirck: Imágenes descargadas de la red social a través de la herramienta para descargas Bulkr. Se seleccionaron imágenes sin restricciones en los permisos.
- Imágenes personales: Imágenes recolectadas en distintos viajes.
- Internet: Para algunas clases en particular que había necesidad de complementar, se buscaban imágenes directamente en la web, siempre tomando en cuenta que no tuviesen restricción de permisos.

Landmarks según la fuente	
Google	182134
Paris	1266
Flirck	984
Personales	262
Web	86
	184732

Tabla 3: Landmarks según la fuente

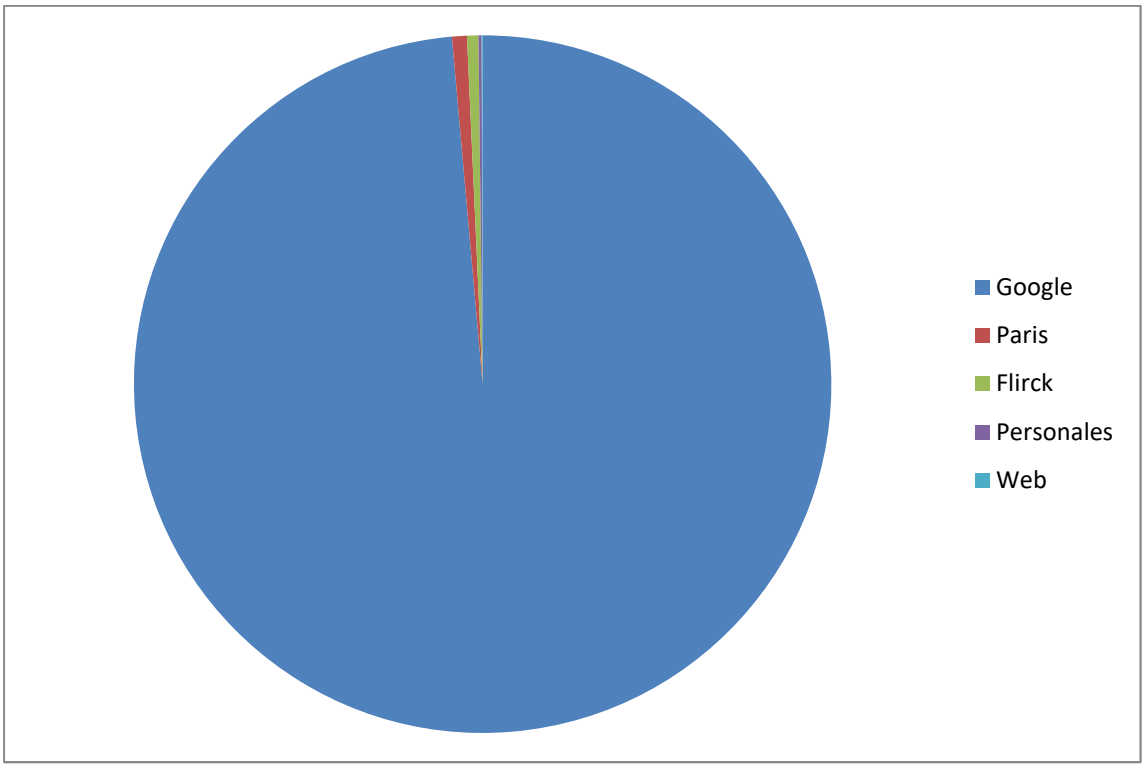


Figura 13: Landmarks según fuente

Clases del dataset	
Alcatraz	8799
Alhambra Palace	17774
Basilica of Saint Peter	49292
Casa Batllo	5139
Charles Bridge	13345
Eiffel	864
Hollywood Sign	581
Itsukushima Shrine	3690
Moulin Rouge	490
National Art Museum of Catalonia	9071
Park Guell	862
Petronas Twin Towers	11090
Rialto Bridge	8433
Roman Coliseum	48867
Rome Pantheon	5436
Statue of Liberty	415
Triomphe	584
	184732

Tabla 4: Distribución de los landmarks en el dataset

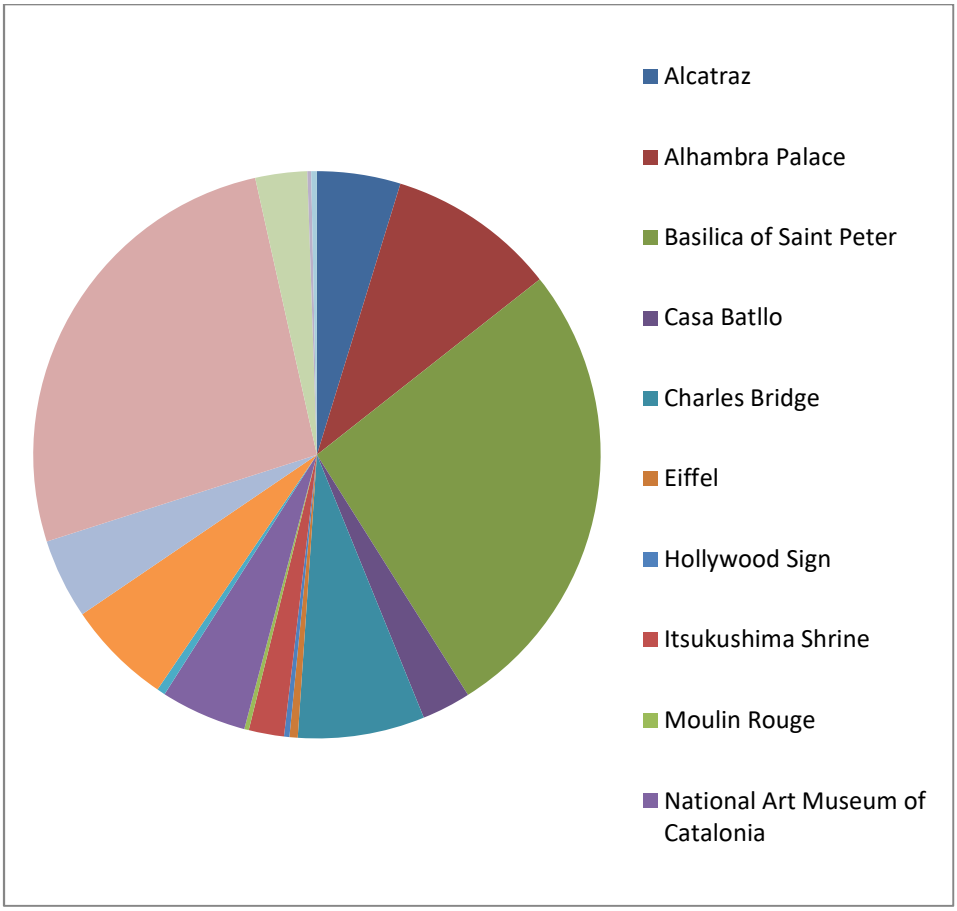


Figura 14: Distribución de los landmarks en el dataset

2.4 Tamaño de las imágenes

Construido el dataset se presentó el problema de cuál era el tamaño adecuado para las imágenes, con dicho fin se realizó una análisis de las ventajas y desventajas de tener tamaños más grandes o pequeños.

- Almacenamiento: A mayor tamaño de imagen, mayores son los requerimientos de almacenamiento
- Precisión del modelo: Normalmente las imágenes más grandes dan mejores valores de precisión
- Overfitting: Imágenes de mayor tamaño pueden tender al overfitting ya que el modelo puede aprender características más especificas
- Requerimientos de memoria: Cuanto más pequeñas son las imágenes menos memoria requiere el modelo
- Batch size: El tamaño debe permitirnos alojar en la memoria de la GPU batches razonables para el entrenamiento (entre 16 y 64 imágenes)
- Filter size: Si se seleccionan tamaños de filtros pequeños (por ejemplo de 3 x 3) e imágenes grandes, además de alargar el tiempo de entrenamiento de la red, se deben tener modelos menos profundos, para que las capas de la red puedan entrar en la GPU

Además de los aspectos mencionados, se debe tener en cuenta que aunque no es obligatorio que las imágenes sean cuadradas, generalmente es una buena idea ya que muchas de las redes dividen la imagen a la mitad. Tomando en cuenta todo lo mencionado, se decidió que el tamaño de las imágenes fuese de 512 x 512, un término medio entre los requerimientos computacionales que generan imágenes más grandes y baja precisión (sobre todo en este caso que es necesario captar detalles) que pueden generar imágenes más pequeñas.

Para redimensionar las imágenes, básicamente tenemos 3 técnicas:

- Squashing: Es una técnica de resampling que ignora el ratio de la imagen y simplemente la redimensiona utilizando algún filtro. Para el caso de downsampling (la situación de la mayoría de las imágenes del dataset, ya que son mayores a 512 x 512) el filtro recomendado es Lanczos (Duchon, 1979).
- Black Borders: Agregar bordes negros para hacer la imagen cuadrada y mantener el ratio y después redimensionarla.
- Center Cropping: Cambia el tamaño de la imagen para que el lado más corto sea de la medida que se quiere obtener, en este caso seria 512, y luego recorta el otro lado para que también quede en 512 y el ratio de la imagen no se modifique.

No se encontraron estudios que indiquen de forma clara cuál de las 3 técnicas es mejor, pero debido a las características del dataset (algunas veces los landamarks aparecen en extremos de las imágenes) se decidió descartar la tercer opción y generar las 2 primeras para luego probarlas en los modelos.

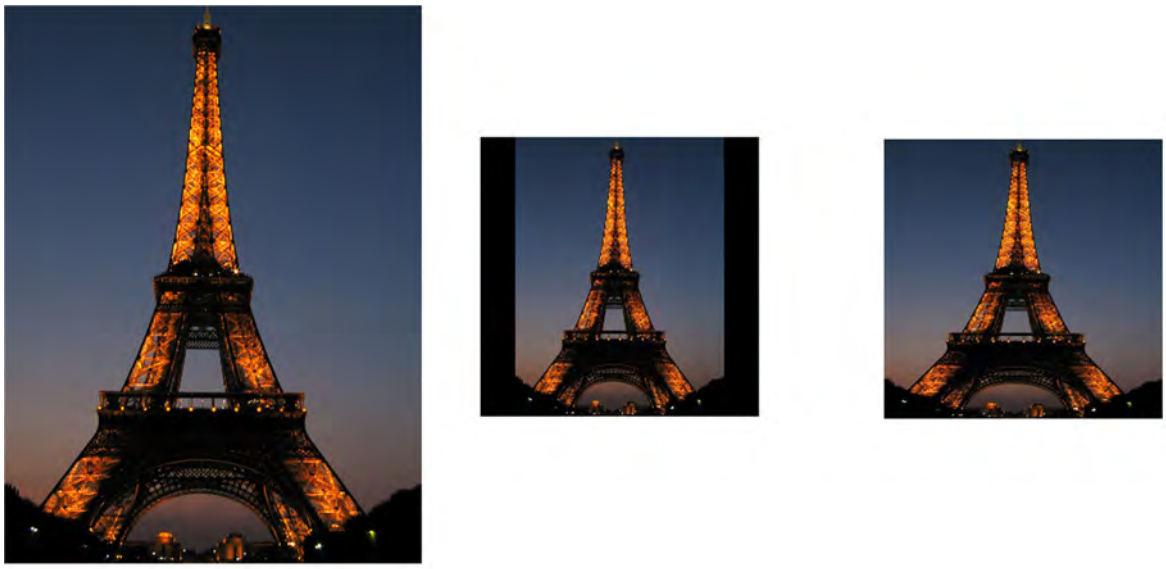


Figura 15: Imagen de la Torre Effeil en los 3 formatos (Original, Black Borders y Squashing)

2.5 Técnicas de validación

El dataset original se dividió en 3 partes distintas:

- Training set: Conjunto de imágenes con las cuales se va a entrenar al algoritmo
- Development set: Conjunto de imágenes que será utilizado para probar los modelos generados e ir ajustando los parámetros o tomando cualquier otra decisión con respecto al algoritmo.
- Test set: Conjunto de imágenes que se utilizaran para evaluar el algoritmo, pero nunca para tomar decisiones con respecto a este o sus parámetros (Ng, 2018).

El tamaño del dev y test set debe ser lo suficientemente grande para que los resultados del desarrollo y la prueba sean representativos del rendimiento del modelo. Por ejemplo si el conjunto de desarrollo tiene solamente 100 ejemplos, la precisión puede variar mucho dependiendo del conjunto de desarrollo elegido. Si bien no hay un número exacto que sea correcto y depende de cada dataset, un conjunto con 10.000 elementos o más podríamos decir que es razonable (Ng, 2018). Por lo que para el caso de este dataset en particular sería suficiente dividirlo en 90/ 10/ 10.

Otro aspecto a destacar, es que la división de los conjuntos se hizo de forma aleatoria para que los 3 mantengan la misma distribución y además se hizo exactamente de la misma forma para los 2 formatos de imágenes para que esto no influya a la hora de compararlos.

3 Experimentación y resultados

En esta sección se desarrollan los modelos y experimentos realizados durante el transcurso del proyecto. Además se muestran los resultados y las justificaciones de los pasos que se fueron siguiendo. El orden seleccionado para la presentación, es el orden cronológico con el que se han ido realizando, finalmente se comentan algunas otras ideas que se pretendían implementar y no fue posible por cuestiones de plazos o capacidad de hardware.

3.1 Transfer Learning

Transfer Learning es un problema de investigación en Deep Learning que se centra en almacenar el conocimiento recogido durante la resolución de un problema, y aplicarlo a otro problema diferente pero de alguna manera relacionado (Jay, n.d.).

Porque utilizar Transfer Learning?

- Dificultad en conseguir la cantidad de datos suficientes para entrenar una red neuronal desde cero.
- Costos de entrenar una red neuronal desde cero, tiempo y recursos computacionales como por ejemplo costosas GPUs.
- Dificultad para encontrar el mejor modelo y los parámetros más adecuados.

Las redes convolucionales son especialmente buenas para aplicar Transfer Learning debido a dos características:

- Los patrones que aprenden son invariables al traslado, esto quiere decir que si por ejemplo aprenden un patrón en la esquina inferior derecha de una imagen, luego lo pueden reconocer en cualquier lado, por ejemplo en la esquina superior izquierda.
- Pueden aprender jerarquías espaciales de patrones, una primera capa convolucional aprenderá pequeños patrones locales como bordes, una segunda capa convolucional aprenderá patrones más grandes hechos con las características de las primeras capas, etc (Chollet, 2018).

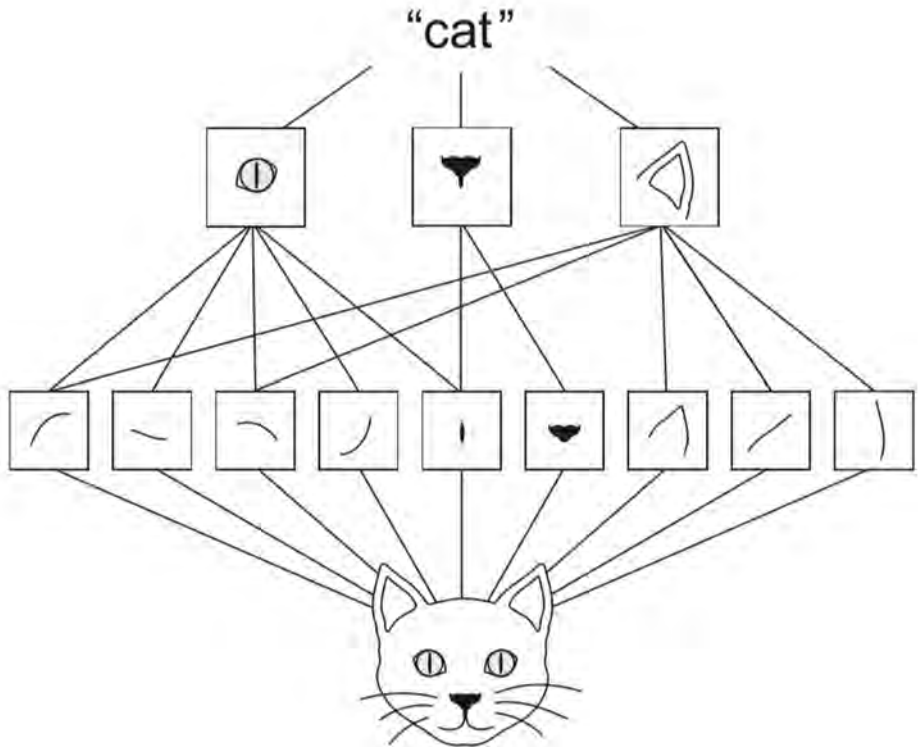


Figura 16: Esquema de cómo funciona una red convolucional (Chollet, 2018)

Los tres mayores escenarios planteados en Transfer Learning son:

- Red convolucional como extractor de características fijas: Se toma una red entrenada previamente con algún dataset grande, como por ejemplo ImageNet, se elimina la última capa completamente conectada y luego se agrega SVM o Softmax y se entrena esta última parte con el nuevo dataset.
- Fine-tunning: Esta segunda estrategia no solo reemplaza y re entrena el clasificador encima de la red convolucional, sino que también afina los pesos de la red convolucional pre entrenada mediante la continuación de la técnica de backpropagation. Se puede aplicar fine-tune en todas las capas de la red convolucional o solo en algunas, lo cual puede ser útil ya que se ha observado que las primeras capas de este tipo de redes aprenden características más genéricas (“Transfer Learning,” n.d.).

Para decidir qué tipo de transfer learning es mejor utilizar en un dataset, se deben tomar en cuenta varios factores, pero los dos más importantes son el tamaño y la similitud con el dataset original con el cual la red fue entrenada:

- El nuevo dataset es pequeño y similar al dataset original: Al ser pequeño el dataset no es conveniente realizar fine-tune, debido a que puede haber riesgos de overfitting. Además al ser similares los datasets, se espera que las características de alto nivel sean similares, por lo tanto se recomienda utilizar la técnica de extracción de características.

- El nuevo dataset es grande y similar al datset original: A diferencia del caso anterior, podemos tener más confianza de no tener problemas de overfitting y manejar la posibilidad de utilizar fine-tune.
- El nuevo dataset es pequeño y muy diferente al dataset original: Debido a que el dataset es pequeño, lo mejor idea entrenar solo un clasificador, pero como además es muy diferente al original, quizá no sea lo mejor ponerlo en el final de la red sino en algún lugar anterior.
- El nuevo dataset es grande y muy diferente al dataset original: En este caso es en el que tiene más sentido aplicar fine-tune, aunque en la práctica generalmente igual nos podemos beneficiar de la inicialización de los pesos de una red pre entrenada (“Transfer Learning,” n.d.).

Si bien los conceptos de tamaño y similitud no se definen de una forma en que se puedan cuantificar de forma precisa, podríamos decir que el tamaño del dataset del proyecto es mediano. Ya que es grande comparado con datasets con unas pocas miles de imágenes, por ejemplo el dataset de Paris y pequeño comparado con datasets con más de 1.000.000 de imágenes como el de Google.

Keras ofrece distintas redes neuronales entrenadas con ImageNet, por lo que se investigó este dataset para determinar la similitud con el dataset del proyecto. Si bien ImageNet tiene una gran variedad de clases que supera ampliamente al dataset del proyecto, dentro de estas clases incluye clases como: edificaciones, puentes, torres, fuentes, etc. Incluso se encontraron imágenes de la mayoría de las clases del presente proyecto, por lo que podemos establecer que hay similitud.

Figura 17: Ejemplos de imágenes de ImageNet (“ImageNet,” n.d.)

Por lo que de acuerdo al tamaño del dataset, su similitud con ImageNet y cuestiones de optimización de tiempo y recursos, se decidió comenzar a experimentar con la técnica de extracción de características de transfer learning.

3.1.1 ResNet50

ResNet es el nombre corto para Residual Network, fue presentada por el equipo de Microsoft en 2015 en la competición ILSVRC, resultando ganadora y superando por primera vez el rendimiento humano. El aprendizaje residual surgió como una necesidad para facilitar el entrenamiento de las redes cada vez más profundas, ResNet inserta conexiones de acceso directo que permiten conectar diferentes capas de la red (He, Zhang, Ren, & Sun, 2015).

Para el proyecto se utilizó la versión implementada por Keras de ResNet50 (50 capas), pre entrenada con ImageNet:

```
from tensorflow.python.keras.applications import ResNet50
from tensorflow.python.keras.models import Sequential
from tensorflow.python.keras.layers import Dense, Flatten, GlobalAveragePooling2D
from tensorflow.python.keras.preprocessing.image import ImageDataGenerator
from tensorflow.python.keras.callbacks import TensorBoard, EarlyStopping, ModelCheckpoint
import numpy as np
import tensorflow as tf
import random as rn

# The below is necessary in Python 3.2.3 onwards to have reproducible behavior for certain hash-based operations.

import os
os.environ['PYTHONHASHSEED'] = '0'

# The below is necessary for starting Numpy generated random numbers in a well defined initial state.
np.random.seed(42)

# The below is necessary for starting core Python generated random numbers in a well-defined state.
rn.seed(12345)

# Force TensorFlow to use single thread. Multiple threads are a potential source of non-reproducible results.
session_conf = tf.ConfigProto(intra_op_parallelism_threads=1, inter_op_parallelism_threads=1)
from keras import backend as K

# The below tf.set_random_seed() will make random number generation in the TensorFlow backend have a well-defined initial state.
tf.set_random_seed(1234)

sess = tf.Session(graph=tf.get_default_graph(), config=session_conf)
K.set_session(sess)

num_classes = 17
image_size = 512

my_new_model = Sequential()
my_new_model.add(ResNet50(include_top=False, pooling='avg', weights='imagenet'))
my_new_model.add(Dense(num_classes, activation='softmax'))

# Say not to train first layer (ResNet) model. It is already trained
my_new_model.layers[0].trainable = False

my_new_model.compile(optimizer='sgd', loss='categorical_crossentropy', metrics=['accuracy'])

data_generator_train = ImageDataGenerator(rescale=1./127.5)
data_generator_test = ImageDataGenerator()

train_generator = data_generator_train.flow_from_directory(
        '../TFM/Dataset_Resize_Split_Train',
        target_size=(image_size, image_size),
        batch_size=32,
        class_mode='categorical')

validation_generator = data_generator_test.flow_from_directory(
        '../TFM/Dataset_Resize_Split_Dev',
        target_size=(image_size, image_size),
        class_mode='categorical')

tbCallBack =  TensorBoard(log_dir='/Tensorboard/Graph_ResNet50_Final', histogram_freq=0, write_graph=True, write_images=True)
early_stopping = EarlyStopping(monitor='val_acc', patience=3)

my_new_model.fit_generator(
        train_generator,
        epochs=20,
        callbacks=[tbCallBack, early_stopping, ModelCheckpoint('model_ResNet50.h5', monitor='val_acc', save_best_only=True)],
        validation_data=validation_generator)

K.clear_session()
```

Comentarios generales de la implementación:

- Se insertó código con el fin de asegurar que los resultados fuesen reproducibles, para poder determinar si los cambios en el rendimiento del modelo eran debido a una modificación o simplemente al resultado de una nueva prueba aleatoria. Este criterio se mantuvo para todos los modelos utilizados durante el proyecto.
- EarlyStopping: Detiene el entrenamiento cuando la métrica monitoreada, en este caso la precisión, deja de mejorar en tres epochs (no tienen por qué ser consecutivas). Esta técnica también se aplicó en todos los modelos, para optimizar los tiempos de entrenamiento.
- ModelCheckpoint: Guarda el modelo generado después de cada epoch, en este caso se agregó un parámetro para que solamente conserve el mejor modelo generado, el cual se define de acuerdo a su precisión. Al igual que en el caso anterior, esta técnica se utilizó para todos los modelos.
- TensorBoard: Es una herramienta de visualización provista por TensorFlow, para utilizarla con Keras es necesario indicar donde se debe guardar el archivo con los datos. TensorBoard fue utilizado para visualizar los resultados de todos los modelos.
- Optimizador SGD (Stochastic gradient descent): Es un método iterativo para optimizar la función objetivo (loss function) que realiza una aproximación estocástica de la optimización por descenso de gradiente.
- Categorical cross entropy: Esta función mide el rendimiento de un modelo de clasificación cuyo resultado es una probabilidad entre el 0 y 1. Su valor aumenta a medida que la probabilidad se aleja de la etiqueta resultado. En este caso se selecciona la opción categorical por que el modelo predice categorías.
- Batch size: Un lote de imágenes generalmente aproxima la distribución del dataset mejor que una sola imagen, pero cuanto más grande el lote es necesaria más memoria. En general se recomienda utilizar lotes de entre 16 y 64 elementos, en este caso se selecciono 32.

Durante el desarrollo del modelo se fueron realizando diferentes pruebas, algunas de las cuales resultaron en mejoras y buenos resultados y otras no. A continuación se detallan cuales implementaciones sirvieron y cuales por el contrario no mejoraron el rendimiento o incluso lo empeoraron.

Lo que funciono:

- Las imágenes redimensionadas con la técnica de squashing
- El optimizador Stochastic gradient descent
- Re escalar las imágenes del dataset de entrenamiento
- Utilizar el factor 1./127.5 para re escalar

Lo que no funciono:

- Re escalar las imágenes del dataset de validación
- Utilizar el factor 1./255 para re escalar

- Las imágenes redimensionadas con la técnica de black borders
- Utilizar la técnica de data augmentation

A continuación se muestran los resultados obtenidos por el modelo que tuvo mejor rendimiento, esto fue definido de acuerdo al valor de precisión resultante de aplicar el modelo sobre el conjunto de validación.

```
Found 147785 images belonging to 17 classes.
Found 18473 images belonging to 17 classes.
Epoch 1/20
4619/4619 [==============================] - 2895s 627ms/step - loss: 0.2491 - acc: 0.9411 - val_loss: 0.4743 - val_acc: 0.8375
Epoch 2/20
4619/4619 [==============================] - 2890s 626ms/step - loss: 0.1050 - acc: 0.9756 - val_loss: 0.4367 - val_acc: 0.8510
Epoch 3/20
4619/4619 [==============================] - 2899s 628ms/step - loss: 0.0807 - acc: 0.9811 - val_loss: 0.4329 - val_acc: 0.8544
Epoch 4/20
4619/4619 [==============================] - 2887s 625ms/step - loss: 0.0691 - acc: 0.9837 - val_loss: 0.4112 - val_acc: 0.8628
Epoch 5/20
4619/4619 [==============================] - 2905s 629ms/step - loss: 0.0610 - acc: 0.9854 - val_loss: 0.4033 - val_acc: 0.8678
Epoch 6/20
4619/4619 [==============================] - 2918s 632ms/step - loss: 0.0550 - acc: 0.9868 - val_loss: 0.4164 - val_acc: 0.8634
Epoch 7/20
4619/4619 [==============================] - 2924s 633ms/step - loss: 0.0510 - acc: 0.9878 - val_loss: 0.3910 - val_acc: 0.8741
Epoch 8/20
4619/4619 [==============================] - 2931s 635ms/step - loss: 0.0476 - acc: 0.9889 - val_loss: 0.3823 - val_acc: 0.8766
Epoch 9/20
4619/4619 [==============================] - 2910s 630ms/step - loss: 0.0447 - acc: 0.9895 - val_loss: 0.3917 - val_acc: 0.8746
Epoch 10/20
4619/4619 [==============================] - 2912s 630ms/step - loss: 0.0425 - acc: 0.9899 - val_loss: 0.3923 - val_acc: 0.8751
Epoch 11/20
4619/4619 [==============================] - 2933s 635ms/step - loss: 0.0401 - acc: 0.9903 - val_loss: 0.3948 - val_acc: 0.8743
```

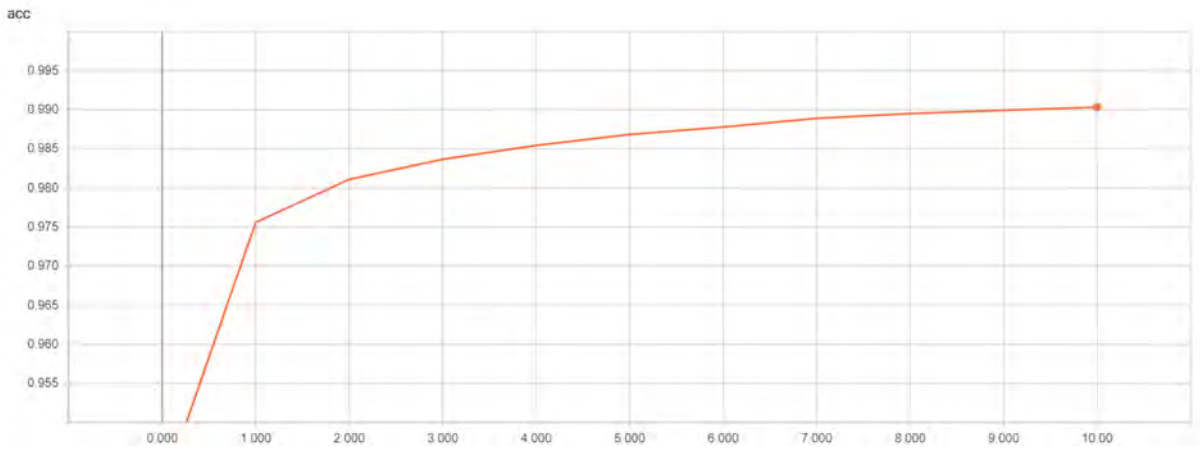


Figura 18: Accuracy del modelo sobre el dataset de entrenamiento por epoch

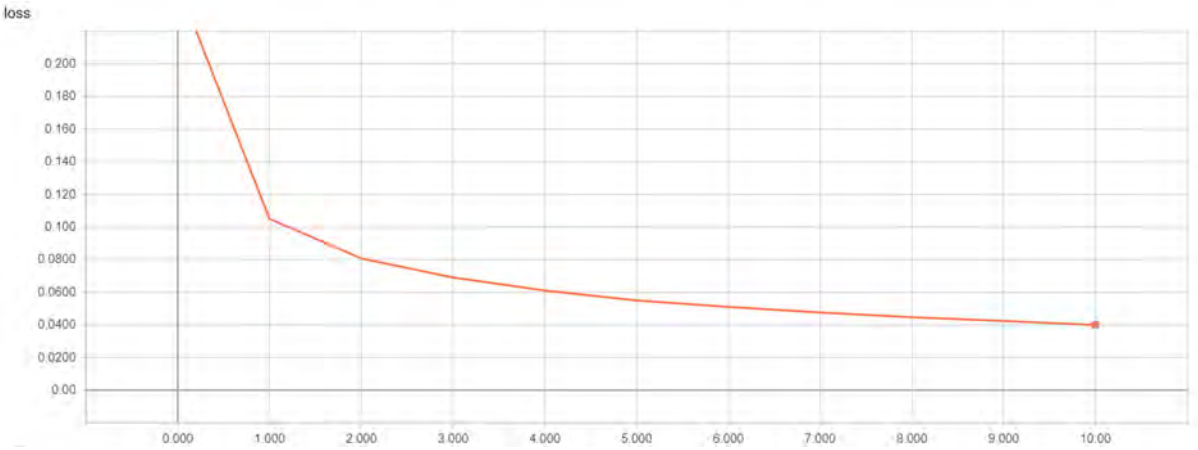


Figura 19: Evolución de la función objetivo (cross entropy loss) sobre el dataset de entrenamiento

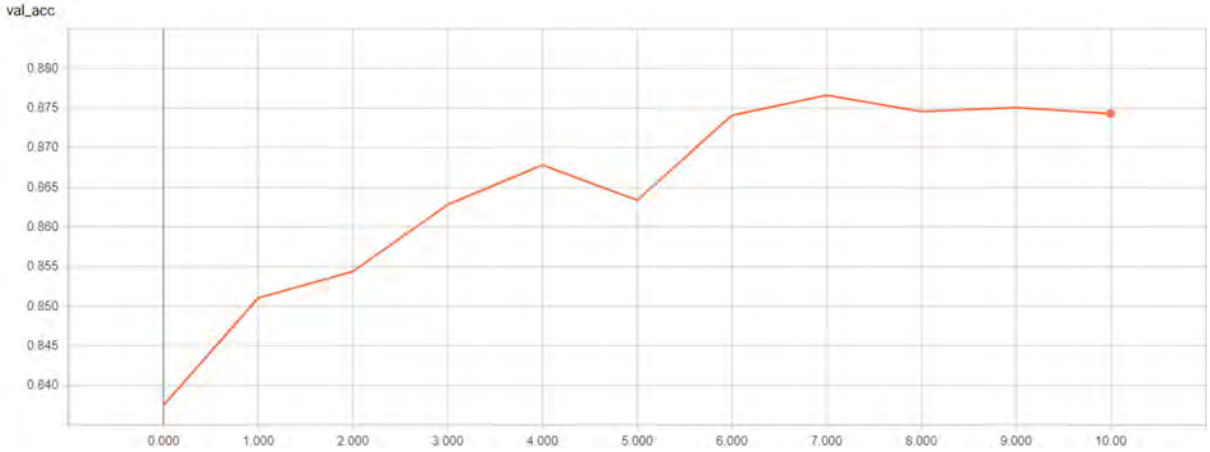


Figura 20: Accuracy del modelo sobre el dataset de validación por epoch

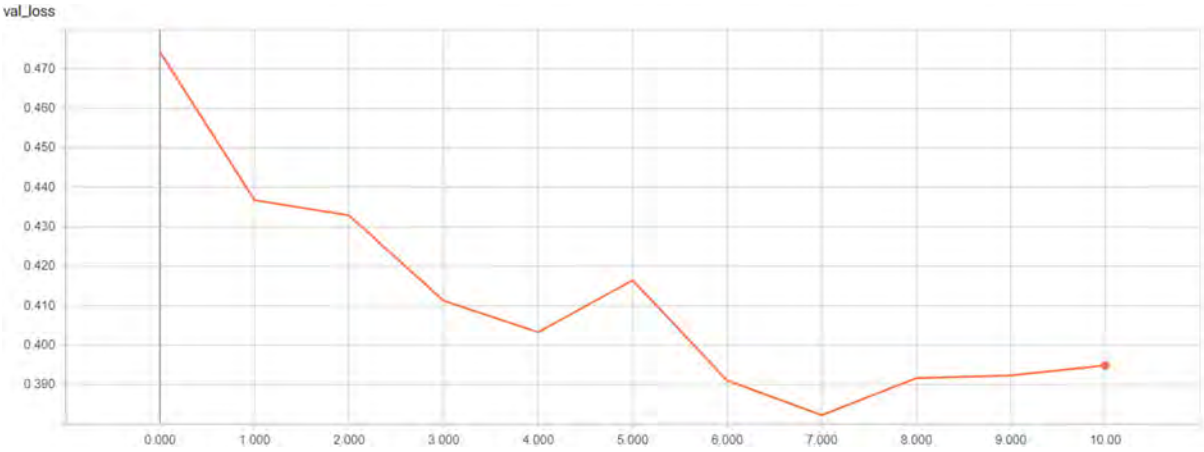


Figura 21: Evolución de la función objetivo (cross entropy loss) sobre el dataset de validación

Comentarios sobre los resultados:

- El mejor modelo se obtuvo en el step 7, luego de entrenarse la red durante 5 horas y 30 minutos, con una accuracy de 0,8766 sobre el dataset de validación
- A partir del step 7 comienza lentamente a producirse overfitting generando una caída del valor de accuracy sobre el dataset de validación
- El modelo entrena correctamente, manteniendo un gap de aproximadamente 10 puntos entre las métricas de accuracy de ambos datasets
- La curva de disminución de la función de loss presenta visiblemente más saltos para el dataset de validación

3.1.2 VGG19

VGG fue el modelo ganador en la competición ILSVRC de 2014, su mayor contribución fue la evaluación de redes más profundas utilizando pequeños filtros convolucionales (Simonyan & Zisserman, 2014).

Para el proyecto se utilizó la versión implementada por Keras de VGG19 (19 capas), pre entrenada con ImageNet:

```
from tensorflow.python.keras.applications.vgg19 import VGG19
from tensorflow.python.keras.models import Sequential
from tensorflow.python.keras.layers import Dense, Flatten, GlobalAveragePooling2D
from tensorflow.python.keras.preprocessing.image import ImageDataGenerator
from tensorflow.python.keras.callbacks import TensorBoard, EarlyStopping, ModelCheckpoint
import numpy as np
import tensorflow as tf
import random as rn

# The below is necessary in Python 3.2.3 onwards to have reproducible behavior for certain hash-based operations.

import os
os.environ['PYTHONHASHSEED'] = '0'

# The below is necessary for starting Numpy generated random numbers in a well-defined initial state.
np.random.seed(42)

# The below is necessary for starting core Python generated random numbers in a well-defined state.
rn.seed(12345)

# Force TensorFlow to use single thread. Multiple threads are a potential source of non-reproducible results.
session_conf = tf.ConfigProto(intra_op_parallelism_threads=1, inter_op_parallelism_threads=1)
from keras import backend as K

# The below tf.set_random_seed() will make random number generation in the TensorFlow backend have a well-defined initial state.
tf.set_random_seed(1234)

sess = tf.Session(graph=tf.get_default_graph(), config=session_conf)
K.set_session(sess)

num_classes = 17
image_size = 512


my_new_model = Sequential()
my_new_model.add(VGG19(include_top=False, pooling='avg', weights='imagenet'))
my_new_model.add(Dense(num_classes, activation='softmax'))

# Say not to train first layer model. It is already trained
my_new_model.layers[0].trainable = False

my_new_model.compile(optimizer='sgd', loss='categorical_crossentropy', metrics=['accuracy'])

data_generator_train = ImageDataGenerator()
data_generator_test = ImageDataGenerator()

train_generator = data_generator_train.flow_from_directory(
        '../TFM/Dataset_Resize_Split_Train',
        target_size=(image_size, image_size),
        batch_size=24,
        class_mode='categorical')

validation_generator = data_generator_test.flow_from_directory(
        '../TFM/Dataset_Resize_Split_Dev',
        target_size=(image_size, image_size),
        batch_size=24,
        class_mode='categorical')

tbCallBack =  TensorBoard(log_dir='/Tensorboard/Graph_VGG19_Final', histogram_freq=0, write_graph=True, write_images=True)
early_stopping = EarlyStopping(monitor='val_loss', patience=2)

my_new_model.fit_generator(
        train_generator,
        epochs=20,
        callbacks=[tbCallBack, early_stopping, ModelCheckpoint('model_VGG19.h5', monitor='val_acc', save_best_only=True)],
        validation_data=validation_generator)

K.clear_session()
```

Este modelo se implementó con las mismas técnicas y parámetros que ResNet50, salvo el tamaño de los lotes, en este caso 32 daba problemas de memoria por lo que se usó 24 y que VGG19 dio mejores resultados sin re escalar las imágenes.

```
Using TensorFlow backend.
Found 147785 images belonging to 17 classes.
Found 18473 images belonging to 17 classes.
Epoch 1/20
6158/6158 [==============================] - 5100s 828ms/step - loss: 0.1847 - acc: 0.9479 - val_loss: 0.1019 - val_acc: 0.9694
Epoch 2/20
6158/6158 [==============================] - 5093s 827ms/step - loss: 0.0912 - acc: 0.9730 - val_loss: 0.0884 - val_acc: 0.9730
Epoch 3/20
6158/6158 [==============================] - 5089s 826ms/step - loss: 0.0767 - acc: 0.9771 - val_loss: 0.0715 - val_acc: 0.9779
Epoch 4/20
6158/6158 [==============================] - 5102s 829ms/step - loss: 0.0697 - acc: 0.9792 - val_loss: 0.0656 - val_acc: 0.9798
Epoch 5/20
6158/6158 [==============================] - 5119s 831ms/step - loss: 0.0647 - acc: 0.9806 - val_loss: 0.0654 - val_acc: 0.9805
Epoch 6/20
6158/6158 [==============================] - 5092s 827ms/step - loss: 0.0611 - acc: 0.9813 - val_loss: 0.0678 - val_acc: 0.9798
Epoch 7/20
6158/6158 [==============================] - 5115s 831ms/step - loss: 0.0581 - acc: 0.9825 - val_loss: 0.0699 - val_acc: 0.9786
```

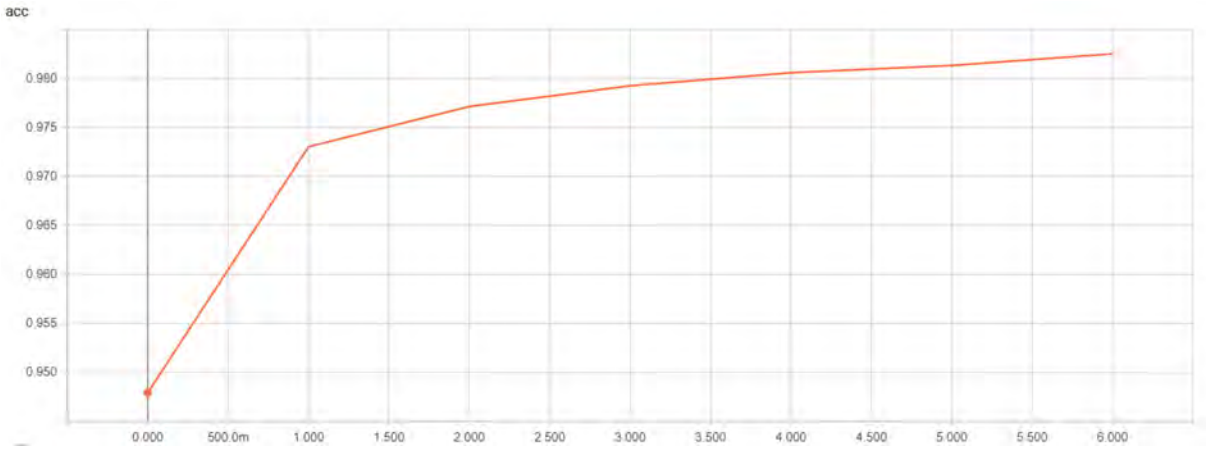


Figura 22: Accuracy del modelo sobre el dataset de entrenamiento por epoch

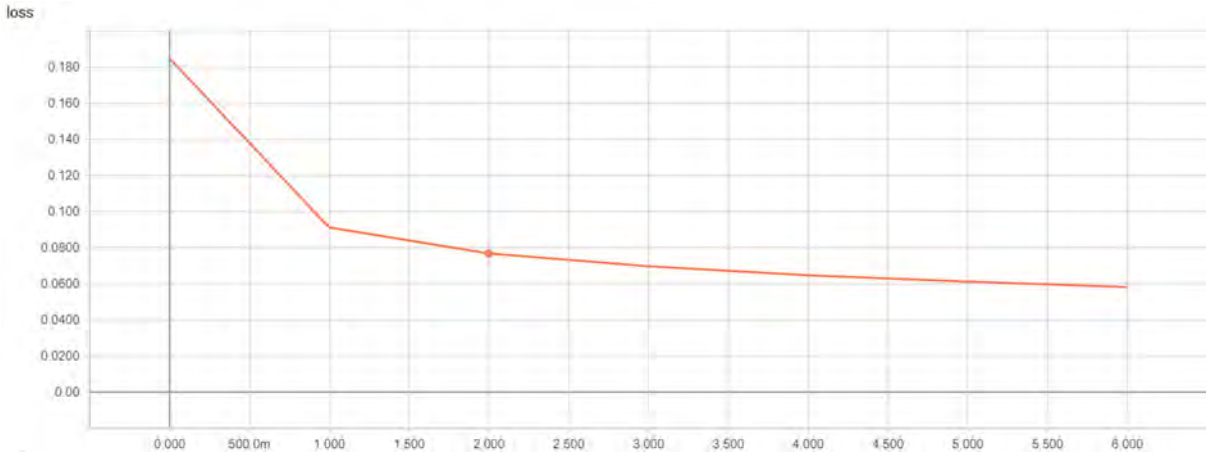


Figura 23: Evolución de la función objetivo (cross entropy loss) sobre el dataset de entrenamiento

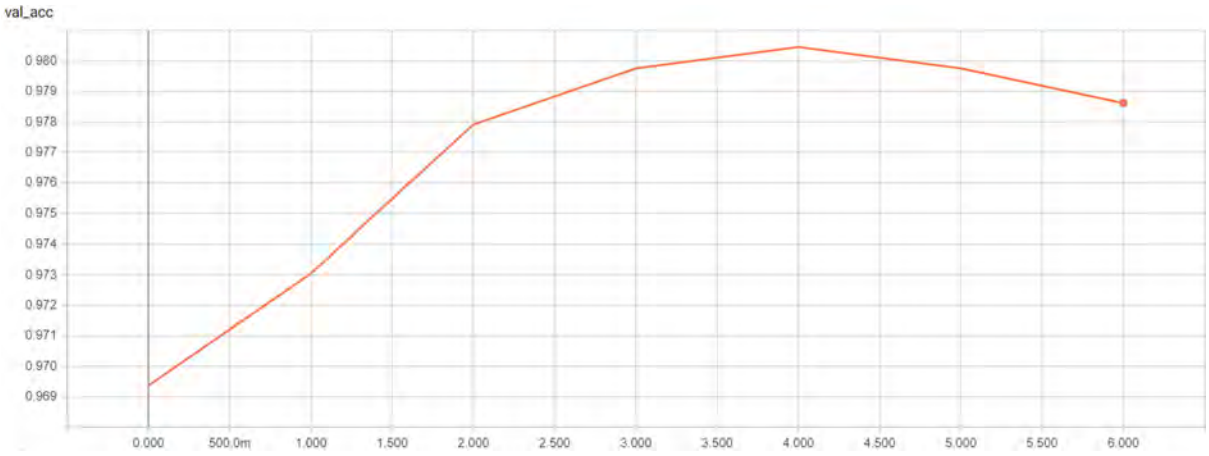


Figura 24: Accuracy del modelo sobre el dataset de validación por epoch

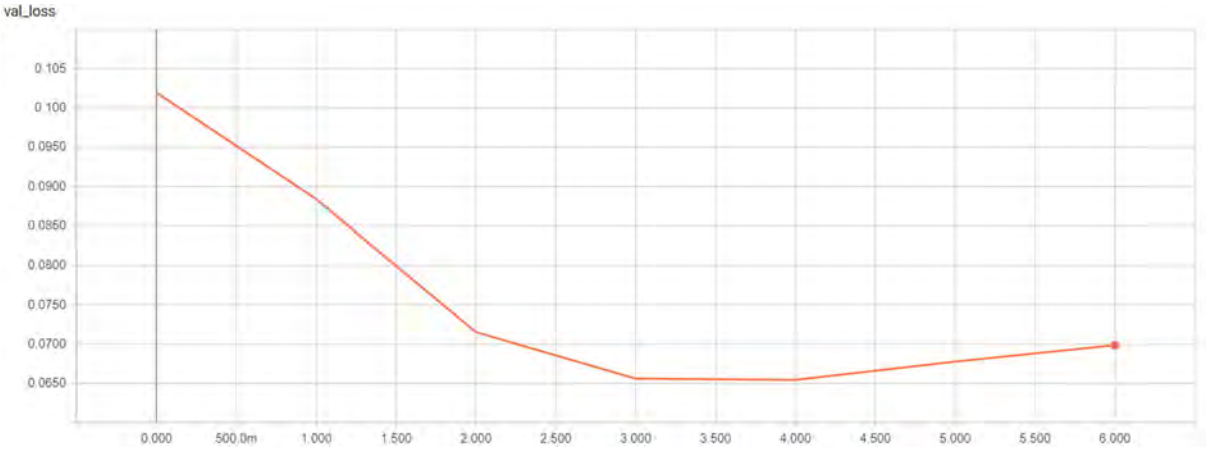


Figura 25: Evolución de la función objetivo (cross entropy loss) sobre el dataset de validación

Desde la primer epoch este modelo mostro buenos resultados, obteniendo su máximo en la quinta iteración con una accuracy de 0.9805.

3.2 Modelo desde cero

Otro de los experimentos realizados fue la construcción de una red neuronal convolucional desde cero, se inició con un modelo basado en una red sencilla poco profunda y se fueron agregando capas y aplicando diferentes técnicas para mejorar su rendimiento. Cabe destacar que por problemas de capacidad de memoria se debió reducir el tamaño de las imágenes a 256 x 256 pixeles y no se pudieron utilizar filtros mayores a 128.

```
sess = tf.Session(graph=tf.get_default_graph(), config=session_conf)
session_conf.gpu_options.allow_growth = True
session_conf.gpu_options.allocator_type ='BFC'
K.set_session(sess)

num_classes = 17
image_size = 256

model = Sequential()
model.add(Conv2D(32, kernel_size=(3, 3), input_shape=(image_size,image_size,3), kernel_initializer='glorot_uniform'))
model.add(Activation("relu"))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(32, (3, 3), kernel_initializer='glorot_uniform'))
model.add(Activation("relu"))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(32, (3, 3), kernel_initializer='glorot_uniform'))
model.add(Activation("relu"))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(64, (3, 3), kernel_initializer='glorot_uniform'))
model.add(Activation("relu"))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(64, (3, 3), kernel_initializer='glorot_uniform'))
model.add(Activation("relu"))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Flatten())
model.add(Dense(128, kernel_regularizer=regularizers.l2(0.01), activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(num_classes, activation='softmax'))

model.compile(optimizer=optimizers.Adam(lr=1e-3, beta_1=0.9, beta_2=0.999), loss='categorical_crossentropy', metrics=['accuracy'])

model.summary
```

```
data_generator_train = ImageDataGenerator(rescale=1./127.5,
                                          rotation_range=20,
                                          width_shift_range=0.2,
                                          height_shift_range=0.2,
                                          horizontal_flip=True)
data_generator_test = ImageDataGenerator(rescale=1./127.5)

train_generator = data_generator_train.flow_from_directory(
        '../TFM/Dataset_Resize_Split_Train',
        target_size=(image_size, image_size),
        batch_size=32,
        class_mode='categorical')

validation_generator = data_generator_test.flow_from_directory(
        '../TFM/Dataset_Resize_Split_Dev',
        target_size=(image_size, image_size),
        batch_size=32,
        class_mode='categorical')

tbCallBack =  TensorBoard(log_dir='/Tensorboard/My_Model_8', histogram_freq=0, write_graph=True, write_images=True)
early_stopping = EarlyStopping(monitor='val_acc', patience=3)

model.fit_generator(
        train_generator,
        epochs=20,
        callbacks=[tbCallBack, early_stopping, ModelCheckpoint('model_8.h5', save_best_only=True)],
        validation_data=validation_generator)

K.clear_session()
```

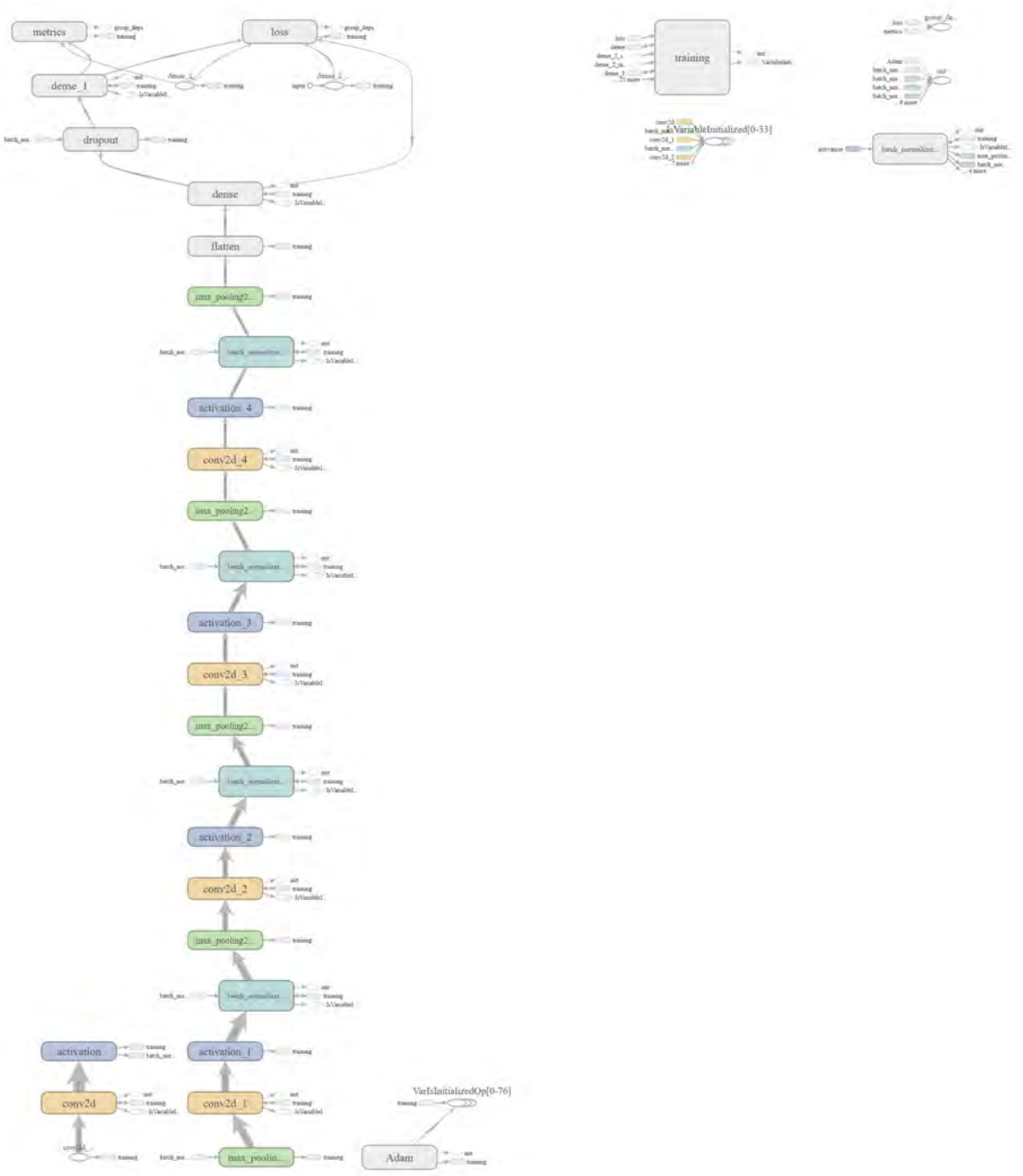


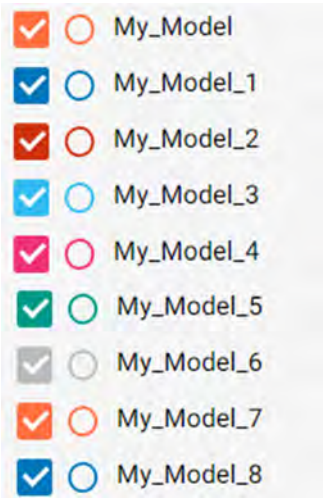
Figura 26: Grafo de Modelo Final

En la imagen y código superior se puede ver la estructura de versión final de la red neuronal construida, a continuación se explican los detalles de su implementación:

- La red cuenta con 5 capas convolucionales y una capa completamente conectada, el modelo inicial contaba con 2 capas convolucionales y una densamente conectada, a medida que se fueron añadiendo capaz al modelo este fue aumentando su precisión.
- La activación utilizada fue RELU (Rectified Linear Unit) ya que en la actualidad es la más recomendada por que converge más rápido que Sigmoid y Tanh

- Se utilizaron diferentes técnicas para evitar overffiting:
  -Data augmentation
  -Regularización L2
  -Dropout y Batch Normalization, primero se incorporó dropout luego de cada capa lo cual mejoro el modelo y luego se dejó dropout solo en la capa densamente conectada y en las capas convolucionales se cambió dropout por Batch Normalization lo que mejoro aún más el modelo.
- Se re escalaron las imágenes con el factor 1./127.5, primero se había utilizado esta técnica solamente con las imágenes de entrenamiento lo que dio muy malos resultados, al agregarse en las imágenes de validación se vio una mejora significativa.
- Como técnica de reducción de muestreo se utilizó Max Pooling.
- Como método de optimización primero se utilizó SGD y luego se cambió por Adam (Kingma & Ba, 2014), lo que mejoro los resultados
- En lugar de utilizar la inicialización de pesos aleatoria de las capas de la red, se agregó el inicializador Glorot normal, más conocido como Xavier (Glorot & Bengio, 2010)

Se construyeron y entrenaron 8 modelos antes de llegar a la versión final, la cual alcanzo una accuracy de 0,9592. Debajo se muestran graficadas las métricas de los 9 modelos generados (My_Model_8 corresponde a la versión final).



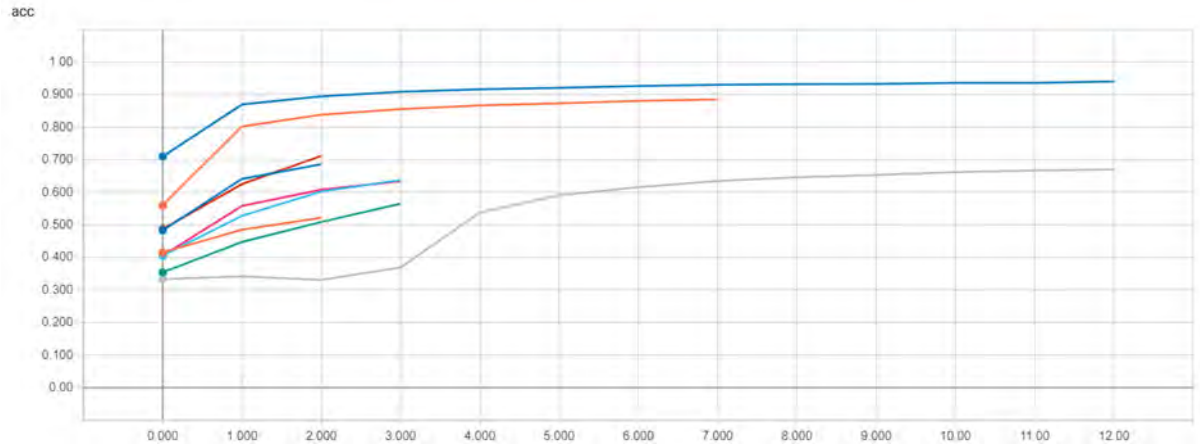


Figura 27: Accuracy de los modelos sobre el dataset de entrenamiento por epoch

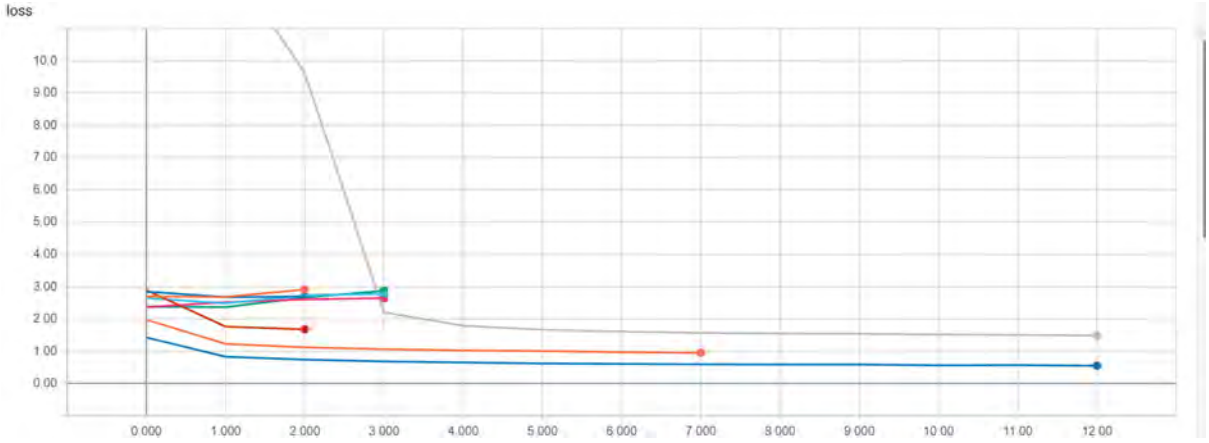


Figura 28: Evolución de la función objetivo (cross entropy loss) sobre el dataset de entrenamiento

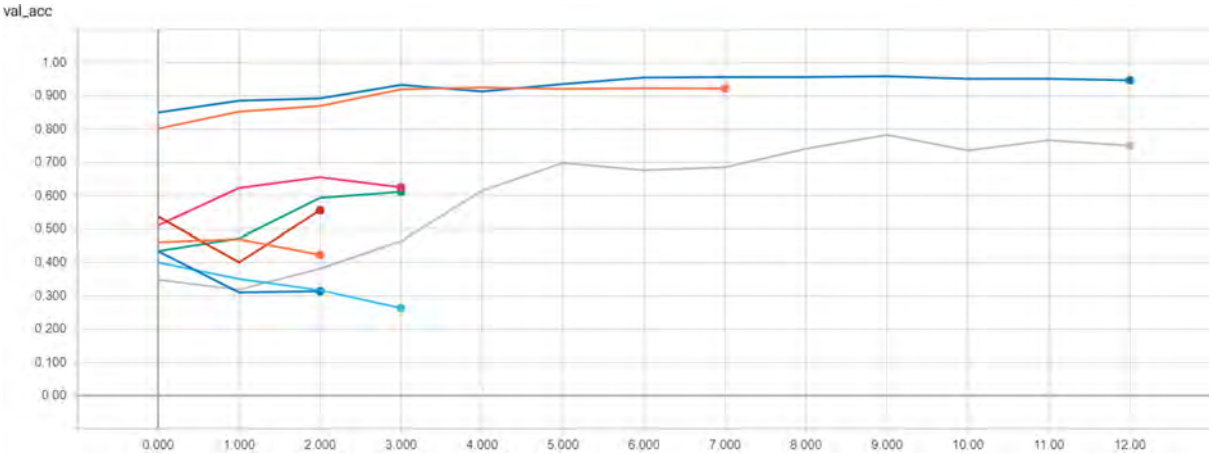


Figura 29: Accuracy de los modelos sobre el dataset de validación por epoch

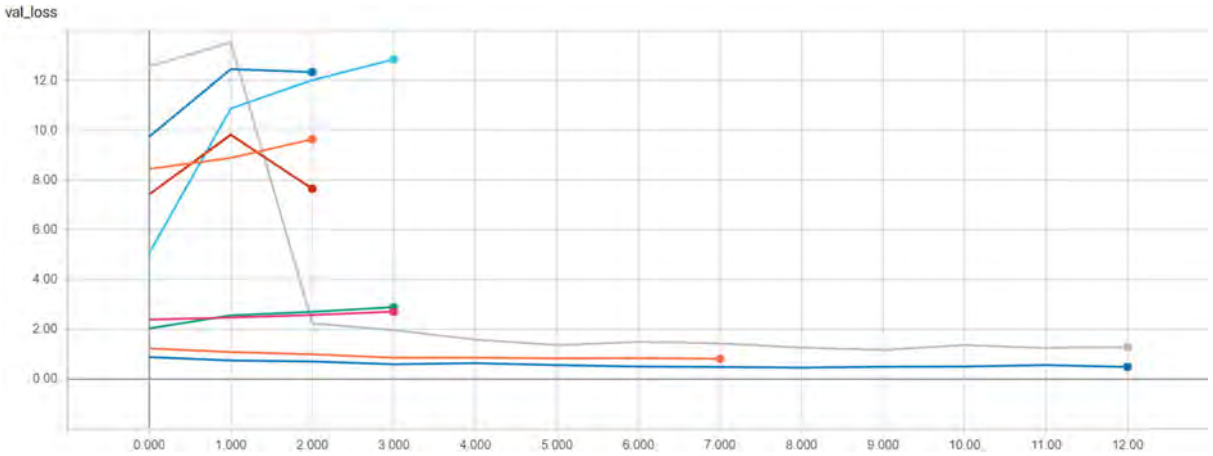


Figura 30: Evolución de la función objetivo (cross entropy loss) sobre el dataset de validación

3.3 Comparación de modelos y resultado final

Como se puede apreciar en el gráfico de la figura 31 el modelo VGG19 fue el que obtuvo la accuracy más alta sobre el dataset de validación, seguido por el modelo creado desde cero.

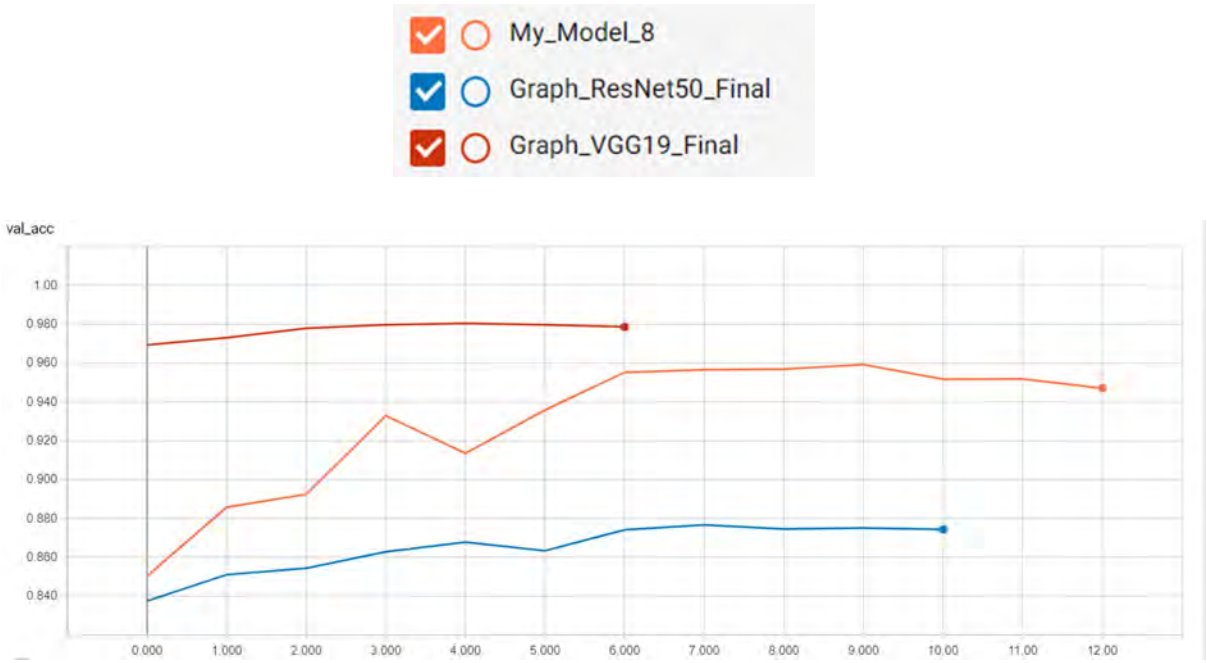


Figura 31: Comparación de la accuracy de los 3 modelos sobre el dataset de validación

Por último, se seleccionó el modelo VGG19, debido a que fue el que tuvo mejor accuracy sobre los datos de validación, para ser evaluado con el dataset de test y se obtuvo un 97,97% de accuracy.

4 Enseñanzas y conclusiones

A través del desarrollo del presente proyecto se obtuvieron las siguientes enseñanzas y conclusiones.

- Importancia de mirar los datos: Mas allá de mirar los datos de forma global a través de la exploración estadística, es importante tomarse un tiempo para mirarlos de forma más detallada, de esta manera vamos a encontrar aspectos fundamentales para el desarrollo del modelo que no se pueden percibir de forma global.
- Tiempo de construcción del dataset: A pesar de que hoy en día existen herramientas que nos ayudan en su construcción, para obtener un dataset de calidad todavía sigue siendo importante el etiquetado o la verificación manual a través de un ser humano, lo cual es muy costoso desde el punto de vista del tiempo.
- Hardware: Para proyectos de deep learning y especialmente de computer visión es fundamental contar con el hardware necesario, no tener GPU o memoria suficiente pueden hacer inviable su realización.
- Importancia de la experimentación: Si bien la experimentación siempre es importante en cualquier proyecto, más aún lo es cuando hablamos de deep learning donde muchas veces es difícil saber qué es lo que las redes están aprendiendo. En la práctica vemos que técnicas que deberían mejorar el rendimiento en teoría muchas veces no lo hacen o viceversa, lo que funciona para un problema no tiene por qué funcionar para otro.
- Respaldos/ trazabilidad/ reproducibilidad: Como en cualquier proyecto es importante tener la información y modelos respaldados, pero además es importante poder mantener cierta trazabilidad. Por ejemplo si un mismo dataset de imágenes lo tenemos en tres formatos diferentes, la misma imagen en cada dataset debe mantener el mismo código, esto va a ser de utilidad cuando necesitemos hacer comparaciones. Otro punto importante es que todos los scripts sean reproducibles con el fin de poder diferenciar los cambios de rendimiento por mejoras o por la aleatoriedad de los datos ingresados al modelo.
- Tiempos: Es fundamental en proyectos de deep learning tener en cuenta los tiempos de entrenamiento de las redes, ya que estos pueden ser muy largos y son inevitables.
- No reinventar la rueda: En la actualidad hay muchos problemas que ya han sido resueltos, además existen técnicas como transfer learning que nos permiten aplicar en nuestros dataset modelos exitosos y entrenados con millones de datos. Por lo que generalmente, a no ser que nuestro problema sea muy específico es mejor comenzar utilizando estas técnicas.
- Deep Learning: Es una técnica muy poderosa que está revolucionando el machine learning en la actualidad, debido a sus logros y ventajas, fundamentalmente en rubros como la visión por computador. Pero para mejorar su utilización es importante conocer cómo funcionan, aspecto sobre el cual todavía queda mucho por recorrer e investigar.

5 Lineamientos futuros

Debido a falta de tiempo e inclusive de recursos de hardware hubieron diferentes aspectos del proyecto que no llegaron a implementarse y se desea hacerlo en un futuro. A continuación se mencionan los principales:

- Ampliar el dataset: El objetivo en una segunda etapa es introducir nuevas clases pertenecientes a otros landmarks, para ampliar el dominio del modelo.
- Dataset abierto: Otro objetivo que no llego a ser implementado, fue el de trabajar con un dataset abierto, o sea que el modelo reciba imágenes no pertenecientes a ninguna de las clases y no las clasifique en alguna de las clases del dataset, sino que identifique que es una clase desconocida. Se llegaron a seleccionar y estudiar dos técnicas OpenMax (Lyang, Li, & Srikant, 2018) y Odin (Bendale & Boult, 2015).
- Zca whitening: En los modelos para visión por computador se recomienda realizar un pre procesamiento de las imágenes con esta técnica, se realizaron varios intentos, pero ninguno fue exitoso debido a que no era suficiente la capacidad de memoria de la GPU (8GB). Investigando el problema, se encontraron varias referencias a la necesidad de tener amplia memoria para poder aplicar zca whitening en el pre procesamiento de imágenes.
- SENets (Squeeze and Excitation Networks): Fue el modelo ganador de la competencia ILSVRC de 2016, este trabajo se enfoca en la relación del canal y propone una nueva unidad arquitectónica, llamada bloque “Squeeze and Excitation”, el cual recalibra las interdependencias entre los canales de la red (Hu, Shen, & Sun, 2017).
- Pruebas: Algunas de las pruebas realizadas, por ejemplo entrenar con los dos formatos de imágenes, se hicieron en solo un modelo, dichas pruebas se podrían extender a los otros modelos para ver si arrojan los mismos resultados. Además también se podría jugar aun un poco más con los parámetros de los modelos en busca de mejorar su rendimiento. El principal problema fue que cada modificación debía probarse sola, para saber si realmente estaba generando algún cambio, esto significaba entrenar el modelo cada vez lo que consumía muchas horas.
- Visualizaciones: Se utilizó la herramienta TensorFlow para las visualizaciones, pero de forma relativamente básica, en un futuro se podría aprovechar más su potencial. Además se investigó código para mostrar que es lo que está reconociendo la red en cada capa, pero nuevamente por cuestiones de cronograma no fue posible llegar a implementarlo.

Bibliografía

Alom, M. Z., Taha, T. M., Yakopcic, C., Westberg, S., Hasan, M., Van Esesn, B. C., … Asari, V. K. (2018). The History Began from AlexNet: A Comprehensive Survey on Deep Learning Approaches. Retrieved from http://arxiv.org/abs/1803.01164

Bendale, A., & Boult, T. (2015). Towards Open Set Deep Networks. https://doi.org/10.1109/CVPR.2016.173

Chollet, F. (2018). Deep Learning with Python.

Dodge, S., & Karam, L. (2016). Understanding how image quality affects deep neural networks. *2016 8th International Conference on Quality of Multimedia Experience, QoMEX 2016*. https://doi.org/10.1109/QoMEX.2016.7498955

Duchon, C. E. (1979). Lanczos Filtering in One and Two Dimensions. *Journal of Applied Meteorology*. https://doi.org/10.1175/1520-0450(1979)018<1016:LFIOAT>2.0.CO;2

Glorot, X., & Bengio, Y. (2010). Understanding the difficulty of training deep feedforward neural networks. *Pmlr*, *9*, 249–256. https://doi.org/10.1.1.207.2059

He, K., Zhang, X., Ren, S., & Sun, J. (2015). Deep Residual Learning for Image Recognition.

Hu, J., Shen, L., & Sun, G. (2017). Squeeze-and-Excitation Networks, 1–11. Retrieved from http://arxiv.org/abs/1709.01507

ImageNet. (n.d.). Retrieved from http://image-net.org/

ImageNet Wikipedia. (n.d.). Retrieved from https://en.wikipedia.org/wiki/ImageNet

Jay, P. (n.d.). Transfer Learning using Keras. Retrieved from https://medium.com/@14prakash/transfer-learning-using-keras-d804b2e04ef8

Jia Deng, Wei Dong, Socher, R., Li-Jia Li, Kai Li, & Li Fei-Fei. (2009). ImageNet: A large-scale hierarchical image database. *2009 IEEE Conference on Computer Vision and Pattern Recognition*, (June), 248–255. https://doi.org/10.1109/CVPRW.2009.5206848

Kaggle. (n.d.). Retrieved from www.kaggle.com

Kingma, D. P., & Ba, J. (2014). Adam: A Method for Stochastic Optimization, 1–15. https://doi.org/http://doi.acm.org.ezproxy.lib.ucf.edu/10.1145/1830483.1830503

Krizhevsky, A., Sutskever, I., & Geoffrey E., H. (2012). ImageNet Classification with Deep Convolutional Neural Networks. *Advances in Neural Information Processing Systems 25 (NIPS2012)*, 1–9. https://doi.org/10.1109/5.726791

Lyang, S., Li, Y., & Srikant, R. (2018). Enhancing the reliability of out-of-distribucion image detection in neural networks.

Microsoft Virtual Machines. (n.d.). Retrieved from https://azure.microsoft.com/en-us/services/virtual-machines/data-science-virtual-machines/

Ng, A. (2018). Train/ Dev/ Test. In *Machine Learning Yearning* (pp. 14–19).

Noh, H., Araujo, A., Sim, J., Weyand, T., & Han, B. (2017). Large-Scale Image Retrieval with Attentive Deep Local Features. *Proceedings of the IEEE International Conference on Computer Vision*, *2017–Octob*, 3476–3485. https://doi.org/10.1109/ICCV.2017.374

NVidia. (n.d.). Retrieved from https://developer.nvidia.com/cuda-gpus

Python Wikipedia. (n.d.). Retrieved from https://es.wikipedia.org/wiki/Python

Simonyan, K., & Zisserman, A. (2014). Very Deep Convolutional Networks for Large-Scale Image Recognition, 1–14. https://doi.org/10.1016/j.infsof.2008.09.005

Tensorflow. (n.d.). Retrieved from https://www.tensorflow.org/

Transfer Learning. (n.d.). Retrieved from http://cs231n.github.io/transfer-learning/

Wu, S., Zhong, S., & Liu, Y. (2017). Deep residual learning for image steganalysis. *Multimedia Tools and Applications*, 1–17. https://doi.org/10.1007/s11042-017-4440-4

Zheng, Y. T., Zhao, M., Song, Y., Adam, H., Buddemeier, U., Bissacco, A., … Neven, H. (2009). Tour the World: Building a web-scale landmark recognition engine. *2009 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*

Workshops, CVPR Workshops 2009, *2009 IEEE*, 1085–1092. https://doi.org/10.1109/CVPRW.2009.5206749