Escola Tècnica Superior d'Enginyeria Informàtica
Universitat Politècnica de València

Detección del modelo de vehículo mediante técnicas de aprendizaje profundo

Trabajo Fin de Máster

Máster Universitario en Gestión de la Información

Autor: David Castellano Falcón

Tutor: Carlos Montserrat Aranda

Tutor: César Ferri Ramírez

Curso 2020-2021

Agradecimientos

Lo primero de todo muchas gracias a todos aquellos que me han apoyado, desde mi familia, amigos y compañeros de trabajo

Le doy gracias a mi familia por apoyar todo lo que hago y dejarme hacer siempre lo que quiero. Ya que por eso soy quien soy.

A mi novia por todo lo que me aguanta y el a**poyo** que me da ya que sin ella tampoco sería quien soy. Además, le encanta todos los regalos que le he hecho aplicando visión artificial. La mejor del mundo, las cosas como son.

Gracias a todos los del DMIP por acogerme tan bien, han sido un buen **optimizador** para mi **gradiente**, el cual, aún está lejos de encontrarse en su **óptimo global**, pero creo que con la **tasa de aprendizaje** que ustedes me han aportado la velocidad de mejora ha sido muy **transformadora**. En especial, a Carlos y a César por haber sido quienes más me aguantan y a quienes más lata les doy, aunque creo que casi ninguno de ustedes se queda atrás, lo dicho, ¡muchas gracias a todos!

Chacho y a mis amigos también les doy gracias, faltaba más.

Resumen

En el campo del aprendizaje profundo (*Deep Learning*), la clasificación de imágenes es una de las tareas más exploradas. Asimismo, dentro de esta tarea hay un subdominio cuya principal característica es la dificultad de distinguir las clases ya que éstas son difícilmente distinguibles entre si (especies de pájaros, flores, animales o tipo de vehículos entre otros). En este trabajo se propone un estudio en la clasificación de vehículos al nivel específico de modelo. Para este fin, se utiliza el conjunto de datos de *Cars196* o también llamado *Standford Cars*, y se entrenan redes neuronales profundas analizando distintas configuraciones de arquitecturas (*ResNet, DenseNet, EfficientNet* y *Vit*) con sus pesos pre-entrenados, así como diferentes técnicas de preprocesado de imágenes. También se estudia el efecto de distintas funciones de pérdida, así como distintas estrategias y configuración de valores de ratio de aprendizaje (*learning rate*). La precisión de los modelos obtenidos se acerca a los valores de los últimos trabajos publicados que abordan el problema de la detección automática del modelo de vehículo.

Palabras clave: Deep learning; redes neuronales; clasificación de imagen; Cars196; Standford Cars; data augmentation; learning rate; función de pérdida; ResNet; EfficientNet;ViT; DenseNet

Abstract

In the field of deep learning, image classification is one of the most explored tasks. Likewise, within this task there is a subdomain whose main characteristic is the difficulty of distinguishing the classes since they are difficult to distinguish from each other (species of birds, flowers, animals or types of vehicles, among others). In this work, a study is proposed on the classification of vehicles at the specific model level. For this purpose, the dataset of Cars196 or also called Standford Cars is used, and deep neural networks are trained analyzing different architectures configurations (ResNet, DenseNet, EfficientNet and Vit) with their pre-trained weights, as well as different techniques of image preprocessing. The effect of different loss functions is also studied, as well as different strategies and configuration of learning rate values. The accuracy of the obtained models is close to the values of the latest published works that address the problem of automatic vehicle model detection.

Keywords: Deep learning; neural network; computer vision; Cars196. Standford Cars; data augmentation; learning rate; loss function; ResNet; EfficienNet;ViT; DenseNet

Tabla de contenidos

Índice de figuras

Índice de tablas

Índice de ecuaciones

1. Introducción

Preámbulo

Hoy en día, el estado del arte de la inteligencia artificial, más concretamente en el aprendizaje profundo (*Deep learning*), mejora a una velocidad frenética en la que año tras año y realmente mes a mes la tecnología más avanzada queda rápidamente desactualizada [1]. ¿Y esto a qué es debido? Pues principalmente a dos factores. El primero de ellos a la mejora en los algoritmos que se realizan y el segundo de ellos a la mejora en la capacidad de cómputo. En relación con esta última mejora y por hacer una paralelización a la ley de Moore [2] (la cual nos dice que cada dos años se duplica el número de transistores en un microprocesador), se podría hablar en inteligencia artificial de la ley de Huang [3] nombrada así probablemente por temas de marketing en un título de noticia. Ahora bien, en el año 2019, el director de Nvidia dio una conferencia en la que mostraba las mejoras en rendimiento en sus tarjetas gráficas y donde hace 5 años se necesitaban 600 horas de cómputo para entrenar una *ResNet50* [4], ahora se puede hacer en 2 horas. Obteniendo una mejora de tiempo 300 veces quedando de esta manera la ley de Moore atrás.

En cualquier caso, el objetivo de este preámbulo es hacer al lector consciente de lo rápido que avanza la tecnología y que es posible (por no decir seguro) que cuando se haya empezado a hacer este trabajo ya hayan salido nuevas técnicas que se podrían usar para obtener mejores resultados en los distintos problemas que se van a resolver.

Por otro lado, hay que destacar que gran parte de la revolución vivida en los últimos años dentro del *Deep learning* se lo debemos en sus inicios a los avances del campo de la visión por ordenador y en concreto al desarrollo de un tipo de red neuronal muy importante que está especializada para trabajar con imágenes llamadas Redes neuronales convolucionales [1]. Se hablarán más de ellas en la sección 2. En cuanto a ejemplos de aplicación de visión artificial mediante aprendizaje automático lo tenemos en diversos campos. Por ejemplo, muy recientemente se ha usado visión artificial en imágenes de rayos-X en los pulmones para ayudar en la detección de Covid-19 [5]. Otro ejemplo relacionado con el COVID es la detección automatizada de uso de la mascarilla [6]. Otra forma de utilizar la visión artificial es mediante la segmentación de imagen en ambiente clínico para la detección temprana de enfermedades [7]. Por añadir otra aplicación práctica de la visión artificial uno de ellos es el de la nigromancia digital como el anuncio de Cruzcampo con Lola Flores [8].

Por último y antes de finalizar el preámbulo el trabajo se encuentra en GitHub y en formato abierto en el siguiente enlace https://github.com/dcastf01/TFM-fineGrained-cars196.

Introducción

Una vez terminado este preámbulo y haber visto brevemente unos pequeños ejemplos de aplicaciones que la visión artificial puede hacer se va a proceder a explicar el motivo de este trabajo.

Antiguamente, el único vehículo que se podía comprar era un modelo T de Ford y además siempre era del mismo color ("Cualquier cliente puede tener el coche del color que quiera siempre y cuando sea negro" [9]). Pero hoy en día las cosas son muy distintas, existe una variedad enorme de vehículos, los cuales pueden llegar a ser muy similares entre ellos, aunque se traten de modelos y marcas de coches completamente diferentes. O incluso entre la misma marca y modelo de coche pueden llegar a verse diferentes debido a las modificaciones que se realizan entre la fecha de lanzamiento inicial y la vida útil del vehículo. Tanta variedad entre estos puede provocar que la identificación de estos sea un problema donde en distintas tipologías de empresas pueda ser interesante resolverlo y tener ese dato. Por ejemplo, en parkings privados donde se detecta cada matrícula de vehículos a la entrada, por temas económicos principalmente. Un ladrón podría entrar en el aparcamiento con un coche barato, cambiar la matrícula con un vehículo más caro y marchar con el mismo. Si este aparcamiento privado hubiera asociado la matrícula al modelo, marca y fecha de lanzamiento del vehículo esto hubiera sido detectado antes de que el ladrón se hubiera llevado el coche. Otro ejemplo podría ser el de los lavaderos de vehículos, los cuales podrían automatizar el tipo de lavado en función del vehículo, consiguiendo con ello optimizar lavados y ahorrar costes.

Por tanto, el problema que se va a resolver en el siguiente trabajo es la identificación de vehículos y esto se hará con la utilización de técnicas de aprendizaje profundo (de ahora en adelante *Deep Learning*) y visión artificial. Es un trabajo que podría tener aplicaciones tanto a nivel industrial como de seguridad como se ha observado en los ejemplos anteriormente mencionados.

Objetivo

Ahora centrándonos más en el objetivo principal de este trabajo. El problema principal para abordar consiste en la clasificación de imágenes de vehículos a un nivel muy detallado. Es decir, no solo se indicará la marca/fabricante del vehículo, sino que también se será capaz de identificar la marca/fabricante, el modelo y la fecha de lanzamiento de este. Esto es posible ya que se dispondrá de una clase/etiqueta para cada vehículo, es decir, si un vehículo ha sufrido modificaciones durante las fechas de lanzamiento esta será una nueva etiqueta. Una marca/fabricante puede tener distintos modelos, pero un modelo solo puede ser de una marca/fabricante, y un modelo puede tener distintos vehículos (a clasificar en nuestro problema) con distintas fechas de lanzamiento,

Motivación

Las ganas de aprender cosas nuevas, unidas a las mejoras que se producen en el campo del *Deep learning* y unido a una beca de colaboración otorgada por el DMIP en la Universidad Politécnica de Valencia para el estudio en visión artificial en un proyecto para Istobal [10] (empresa fabricante de trenes de lavado de coches) ha hecho que realicé este trabajo final de máster. Esta oportunidad me ha dado la posibilidad de utilizar distintas tipologías de problema en la visión artificial (clasificación de imagen, detección de objetos, segmentación de imagen, generación de imagen, etc.). Lo cual aprecio ya que estos conocimientos me ayudarán en mi futuro tanto laboral como personal (ya que me ha servido para realizar regalos muy originales).

Estructura del trabajo

La memoria de este proyecto está dividida en 6 secciones/capítulos. En esta primera sección describen los objetivos y la motivación del trabajo. En la siguiente sección se va a enmarcar el problema en el que nos encontramos, el estado del arte y definir en qué área en concreto de la visión artificial se ha trabajado para afrontar los objetivos del problema planteado. Una vez enmarcado el problema se procederá a explicar brevemente las principales arquitecturas que se utilizan en clasificación de imagen, aprovechando las mismas para utilizarlas a posteriori en los experimentos que se detallarán en la sección 3. Para finalizar la segunda sección se trata la importancia de realizar *Data Augmentation* (aumento de datos) y se procede a explicar distintas funciones de pérdida que se usan en los experimentos que se explican a continuación.

En la tercera sección se procede a explicar los experimentos a realizar, los cuales se pueden agrupar de la siguiente manera:

- Experimentos de distintas arquitecturas.
- Experimentos usando modelos pre-entrenados o no
- Experimentos con distintos tipos de *Data Augmentation* o sin *Data Augmentation*.
- Experimentos con distintas tasas/ratios de aprendizaje (de ahora en adelante *learning rate*).
- Experimentos con distintas funciones de pérdida.
- Experimentos con la mejor configuración obtenida de los experimentos anteriores.

Después de esto, en la cuarta sección, se muestra los resultados con una discusión de estos. A continuación, en la quinta sección, se comentan las distintas limitaciones que se tuvieron en la realización de este trabajo y se discuten algunos resultados que aportan valor añadido al lector. Y, para finalizar, se remarcarán las conclusiones obtenidas del trabajo, definiendo en este punto el modelo elegido para resolver el problema de clasificación de hilo fino[1] o también llamado *fine-grained* de vehículos. Y cuáles son los posibles trabajos futuros.

[1] Problema de clasificación donde la apariencia de ciertas categorías están relacionadas (se explicará con más detalle en la sección 2).

2. Estado del arte

En este capítulo, se realizará una revisión de unos conceptos básicos que serán útiles para el lector. A continuación de esto se explicarán que son las redes convolucionales seguido de una evolución temporal del estado del arte en clasificación de imagen desde finales de 2012 hasta 2021. Seguido de esto se explicará un poco la particularidad del problema que se va a resolver. Y finalmente se abordarán distintas técnicas que se utilizan en los distintos experimentos que se realizan en el trabajo.

Conceptos básicos

En este apartado se tratarán conceptos básicos como sobre que es el *Deep Learning* donde además se explicará los elementos básicos del entrenamiento de una red neuronal y la tipología de problemas que soluciona.

Deep Learning

El *Deep Learning* es un subcampo dentro del *Machine Learning*, el cual utiliza redes neuronales (se hablará de las mismas en el siguiente apartado) y cuya principal característica es que puede llegar a obtener mejores resultados que el *Machine Learning* en tareas donde existe una gran cantidad de datos (véase la Ilustración 2.1).

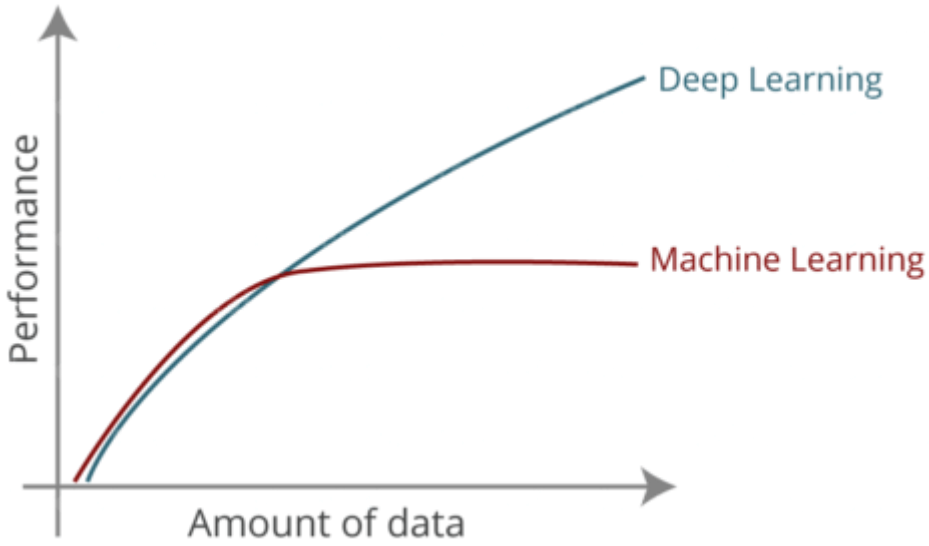

*Ilustración 2.1 Comparación entre Deep Learning y Machi*ne Learning respecto a calidad y cantidad de datos. Fuente: *[11]*

Para entender mejor el *Deep Learning*, se tiene que entender que es una red neuronal, la cual se puede ver como un regresor lineal, exceptuando que en lugar de ser un regresor normalmente son muchos regresores lineales unidos por distintas capas y en distintas capas se utilizan funciones de activación, las cuales ayudan a eliminar parte de la linealidad. Esto les permite a las redes neuronales identificar patrones y clasificar diferentes tipos de información. Siendo usual que las primeras capas aprendan conceptos generales y las últimas capas conceptos más abstractos. Por ejemplo, en el caso

de la imagen de un perro en las primeras capas la red aprendería que son líneas rectas y horizontales y en las últimas capas características como que es un ojo, la nariz, etc [12].

Existen multitud de tipos de redes neuronales con las que se puede crear un sinfín de arquitecturas. Las más destacables serían:

- Redes neuronales multicapas (*MLP* del inglés *Multi Layer Perceptron*) [13].
    - La principal característica de estas redes es que todas las capas se encuentran conectadas permitiendo resolver problemas no lineales, actualmente este tipo de red neuronal es la que se utiliza a la salida de arquitecturas para obtener el resultado final, normalmente una neurona por etiqueta (en un problema de clasificación) o valor (en un problema de regresión) que se desea resolver.
- Redes convolucionales (*CNN* del inglés *Convolutional Neural Network)* [14].
    - Estas redes son las principales que se utilizan hoy en día para resolver problemas relacionados con imagen debido a que tienen una componente que relaciona las variables de forma espacial, en el siguiente apartado se hablara en más detalle de esta red.
- *Transformers* [15].
    - Este tipo de redes se usaron inicialmente en el procesamiento del lenguaje natural, su principal característica es la de relacionar la importancia de los distintos valores entre ellos mediante una capa llamada *Self-Attention* la cual se puede ver más en detalle en su *paper* original [15]. Debido a su versatilidad se está empleando para solucionar todo tipo de problema (texto, audio, imagen, video, generación de contenido…)
- Redes generativas adversariales (*GAN* del inglés *Generative Adversarial Neworks)* [16].
    - Las redes generativas adversariales consisten en usar 2 redes neuronales con objetivos opuestos. Donde una funciona como una red generadora de contenido y la otra red funciona como discriminatoria y se la entrena para discernir del contenido real o el generado por la red generadora. Este tipo de redes se utilizan principalmente para generar nuevo contenido.

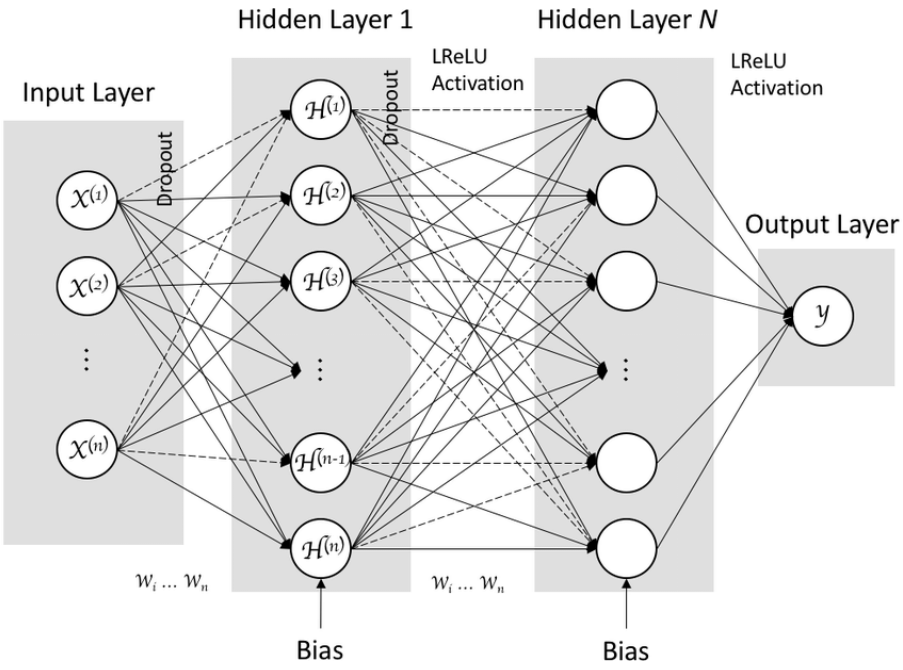

Ilustración 2.2 Red neuronal multicapa (MLP). Fuente: [11]

En la Ilustración 2.2 se puede observar un ejemplo de red neuronal multicapa. En esta red se introducirían los valores en la capa de entrada, cada una de estas llegan a uno de los nodos x^n los cuales se llaman neurona y cada una es un regresor lineal. El resultado de cada neurona pasa a las neuronas de la siguiente capa donde además en algunas capas se podría tener una función de activación, o no, en este caso después de la capa 1 y la capa N se utiliza una *LReLu* (similar a la *ReLu*, la cual transforma los valores negativos a 0). A parte de este tipo de redes neuronales más adelante se verán las redes neuronales convolucionales las cuales están más centradas en problemas de imagen.

En el entrenamiento de estas redes intervienen distintos elementos a tener en cuenta. Para visualizarlo mejor se recomienda apoyarse en la Ilustración 2.3, la cual es un esquema muy útil de aprendizaje auto supervisado, pero es el mejor esquema que se ha encontrado para explicar el entrenamiento de estas redes, además está inspirado en el framework de Pytorch [17].

En primer lugar, y de color rojo se tiene el conjunto de datos con el que se desea entrenar, pueden haber pasado ya por una fase aumentación de datos (este concepto se verá más adelante en esta sección por *Data Augmentation)* o apoyarse en el *Collate Function* para realizarlo al crear el cargador de datos (*Dataloader)*. Este cargador de datos es lo que alimenta el entrenamiento del modelo y está compuesto de lotes de datos del dataset, en nuestro caso lotes de imágenes (de ahora en adelante se le llamará *batch size*). Esto es debido a que entrenar un modelo con un conjunto de imágenes en lugar de una imagen es mucho más rápido ya que se reduce el tiempo en el que se aplica el proceso de optimización (*backpropagation* más adelante se hablará más de él).

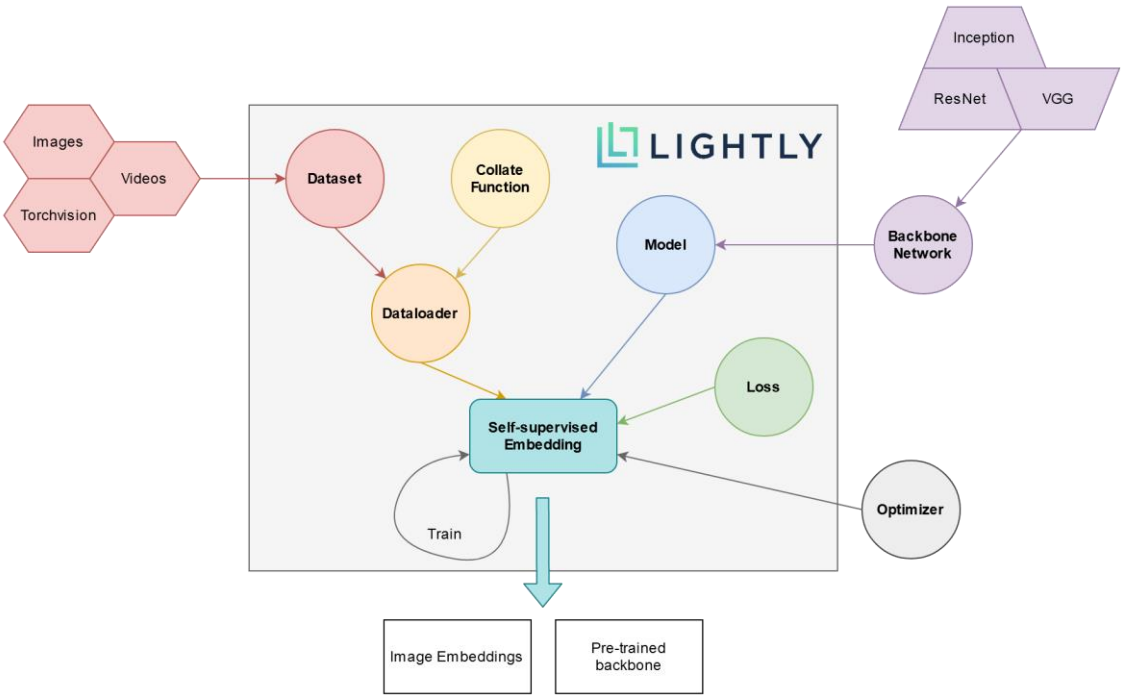

Ilustración 2.3 Esquema de diferentes conceptos en un entrenamiento para un modelo auto supervisado. Fuente: [18].

Por otro lado, es necesario decidir el modelo de red neuronal que se va a utilizar, ya que es en este modelo donde se introducirán los datos (imágenes en este trabajo) y se obtendrá una salida en función del problema elegido. En el caso de un problema de clasificación la salida será un vector de n dimensiones para cada imagen de entrada, donde n es el número de clases y el valor de cada dimensión en n es la probabilidad de que esa sea la clase de la imagen de entrada. Ahora que se dispone del modelo se tiene que seleccionar la función de pérdida que se desea optimizar (más adelante hay una sección de las distintas funciones de pérdida que se utilizarán en el trabajo) y el optimizador a utilizar. La variable más característica de este optimizador es la velocidad de aprendizaje (*learning rate)* donde valores muy altos provocará la falta de convergencia en ningún mínimo y donde valores muy bajos hará que encuentre un mínimo local muy rápido y no le permita mejorar sus resultados (véase la Ilustración 2.4).

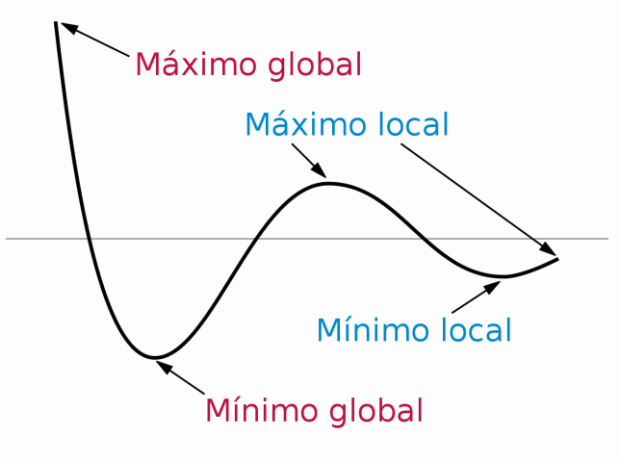

Ilustración 2.4 ejemplo de mínimos y máximo locales y globales. Fuente: [19].

Con todos estos elementos definidos, se procede a explicar en qué consiste el entrenamiento, el cual es en un proceso iterativo por el que todo el conjunto de datos pasa al menos una vez por iteración (esto es un *epoch)* y tiene los siguientes pasos:

1. Se selecciona un lote (*batch*) de datos.
2. Este *batch* se introduce en el modelo.
3. Se extraen las predicciones del modelo y se comparan con los valores reales mediante la función de pérdida.
4. Mediante el optimizador se actualizan los pesos del modelo para disminuir el valor obtenido de la función de pérdida.
5. Se repite el proceso con todo el dataset, de esta manera se completa un epoch.
6. Se repite el proceso el número de epoch que se hayan definido en el entrenamiento

Una vez finalizado este entrenamiento se usa un conjunto de datos de las mismas características donde el modelo no haya visto ninguno de estos datos. Y se aplica diversas métricas, depende del problema que se esté resolviendo, en el caso de este trabajo esta métrica será el *accuracy* debido a que el conjunto de imágenes está balanceado. Esta es una métrica la cual describe lo bien que funciona el modelo en todas las clases [20]. Se calcula como la relación entre el número de predicciones correctas y el número total de predicciones, siendo el resultado un porcentaje de acierto (veasé la Ecuación 2.1).

Ecuación 2.1 Formula del accuracy. Fuente: [20].

$$Accuracy = \frac{True_{positive} + True_{negative}}{True_{positive} + True_{negative} + False_{positive} + False_{negative}}$$

Con este segundo análisis se puede verificar si el modelo funciona correctamente o no y además se puede comparar entre distintos modelos cual funciona mejor.

Se desea también insistir que la flexibilidad y la potencia que aportan las redes neuronales han hecho que se apliquen en distintos campos (Procesamiento de lenguaje natural, reconocimiento de voz, detección de objetos…) como nos indica [21].

Tipología de problemas en Deep Learning

En [22], página ampliamente utilizada por la comunidad de investigadores en *Deep Learning*, el estado del arte se divide en distintas secciones que contienen distintas tareas y estas tareas disponen de distintos *benchmark*, los cuales están compuestos por un dataset y una métrica determinada. Siendo las secciones más destacables la de visión por ordenador y procesamiento de lenguaje natural. Dentro de la sección de visión por ordenador se puede observar que existen una gran multitud de tareas (1079) diferentes donde se destacan las siguientes:

- Clasificación de imágenes.
    - Esta es una tarea fundamental que intenta comprender una imagen completa como un todo. El objetivo es clasificar la imagen asignándole a una etiqueta específica [22].
- Detección de objetos.

    - Consiste en detectar instancias de objetos de una determinada clase dentro de una imagen, los métodos de última generación se pueden clasificar en dos tipos: métodos de una etapa y métodos de dos etapas [22]. Los de primera etapa priorizan la velocidad de inferencia (*YOLO* [23] y *SSD* [24] entre otros) y los de segunda etapa los cuales priorizan el *accuracy* respecto a la velocidad (*Faster R-CNN* [25] por ejemplo).
- Segmentación de imágenes.
    - Es la tarea de identificar la pertenencia de un píxel a una clase determinada, su uso más característico se encuentra en la medicina para la detección de tumores. También se está usando entre otras cosas para investigaciones relacionadas con los coches autónomos

Hay que destacar que dentro de estas tareas existen otro tipo de subtareas más detalladas, las cuales tienden a tener sus particularidades. En el caso de clasificación de imagen, el cual es el problema (a grandes rasgos) que se resuelve en este trabajo, se pueden ver las siguientes subtareas:

- Clasificación de imágenes de pocos disparos (*Few-Shot Image Classification*).
    - Esta subtarea consiste en realizar la clasificación de imágenes con solo unos pocos ejemplos para cada categoría. La principal aplicación que se busca con esto es poder utilizar redes neuronales con poca cantidad de imágenes de entrenamiento [22].
- Clasificación de imágenes de hilo fino (*Fine-Grained Image Classification*).
    - Este problema en concreto es el que se resuelve en este trabajo y se centra en diferenciar entre clases de objetos difícilmente distinguibles al menos que se sea un experto (especies de animales,flores, pájaros, etc) [22]. Más adelante se hablará un poco más en detalle de este problema.
- Clasificación de imágenes semi-supervisadas (*Semi-Supervised Image Classification).*
    - Utilización de datos no etiquetados y datos etiquetados para aumentar el rendimiento de la clasificación [22]. La idea principal es tener una red neuronal que haya aprendido a diferencia distintos patrones de las imágenes (inicialmente todas sin etiquetar). Para posteriormente, mediante un proceso de re-entrenamiento, con imágenes etiquetadas (un porcentaje pequeño del dataset, usualmente un 1% o 10% del mismo) el modelo obtenga buenos resultados.

Como se ha observado existen una multitud de problemas que se resuelven en el campo del *Deep Learning* y se ruega visitar [26] para ver más detenidamente los principales problemas que existen. Pero este trabajo se centrará en clasificación de imagen, más concretamente en clasificación de hilo fino (más adelante se explicará la particularidad de un problema de clasificación de hilo fino con un poco más de detalle). Pero antes de proceder a explicar el problema de clasificación de imágenes se explicará al lector con algo más de detalle qué es una red neuronal convolucional y su diferencia con las redes neuronales multicapas en los problemas de imagen.

Redes neuronales convolucionales[2]

Si para el procesamiento de una imagen 2D se utilizara una *MLP* se estaría usando cada píxel de la imagen como si fuera una variable independiente, es decir, como si fuera un vector plano y perdiendo, de este modo, la información de la posición del píxel en la imagen y su relación con los vecinos más próximos. Por otro lado, en una red convolucional, al igual que para el ojo humano, para la interpretación de la imagen se tiene en cuenta tanto su posición como la distribución de brillos en la zona en la que se encuentra (i.e. el brillo de los píxeles vecinos) aportando una información mucho más rica a la hora de llevar a cabo interpretaciones de las imágenes.

Por tanto, las redes neuronales convolucionales a diferencia de las multicapas están diseñadas para aprovechar la estructura espacial de las imágenes. De este modo, se aprovecha de la relación que existe entre un píxel y sus píxeles vecinos. De este modo esta relación que existe entre los píxeles les permite detectar estructuras, formas y patrones.

Una red convolucional es un tipo de red neuronal que se caracteriza por aplicar un tipo de capa donde se realiza una operación matemática conocida como convolución [1] La convolución generará nuevos píxeles utilizando una matriz de números (llamados filtros o *kernel*) sobre cada píxel y sus vecinos. Esto da como resultado una nueva imagen (ver Ilustración 2.5), como cada capa de la red puede tener distintos filtros el resultado de los mismos se puede denominar mapas de características. En cuanto a quién decide los valores de estos filtros, es la propia red neuronal en la fase de entrenamiento quien decide sus valores en función del *backpropagation* y la función de pérdida elegida (más adelante se profundizará un poco más sobre esto).

Ilustración 2.5 Imagen obtenida después de aplicar un filtro convolucional. Fuente: [12].

[2] Si se desea más información sobre este tipo de redes se ruega ver el siguiente video [12].

Clasificación de imágenes

Una vez obtenida una idea general de qué son las redes convolucionales se va a explicar el estado del arte de la clasificación imágenes. La clasificación de imágenes es una tarea fundamental que intenta comprender una imagen completa como un todo. El objetivo es clasificar la imagen asignándole una etiqueta específica. Normalmente, la clasificación de imágenes se refiere a imágenes en las que solo aparece y se analiza un objeto. [22]

Hoy en día para esta tarea se utilizan principalmente redes neuronales convolucionales profundas. Y fue a partir de la competición de Imagenet de 2012 [27], con la llegada del modelo *AlexNet* [28], cuando el número de investigaciones en esta tarea y en este campo (visión artificial) aumentó considerablemente, (ver Ilustración 2.6. Actualmente se consigue un *accuracy* (se utilizará la palabra *accuracy* en lugar de precisión para no confundir con precisión y para usar la nomenclatura de la literatura) en Top 1[3] cercanos al 90% en el *dataset* (conjunto de datos) de Imagenet, el cual tiene 1000 clases diferentes.

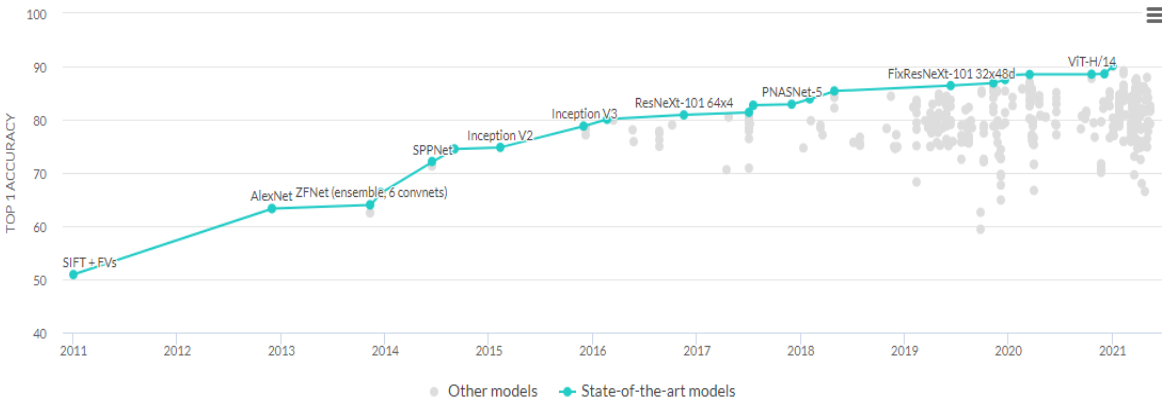

Ilustración 2.6 Estado del arte en clasificación de imagen. Fuente: [22].

Además de la mejora en el *accuracy*, el avance de estos estudios ha permitido demostrar que a mayor número de capas profundas supone mejores resultados. Eso sí, a costa de mayor necesidad de cómputo o mayor tiempo de entrenamiento. Ahora bien, uno de los grandes problemas del *Deep Learning* es el *backpropagation*, ya que el gradiente que se aplica para actualizar los parámetros se puede perder a medida que recorren las capas inferiores[4] de la red neuronal al hacerse infinitamente pequeño [29].

Para solucionar el problema del gradiente, en 2015 se creó el modelo *ResNet* [4]. Éste introduce "atajos" llamados *skip-connections* a través de la red neuronal para facilitar la propagación de valores mediante descenso de gradiente. En la Ilustración 2.7 se puede visualizar fácilmente un ejemplo de este tipo de "atajos". Está experimentalmente validado que estas conexiones ayudan a la convergencia del modelo ya que permite a la propagación del gradiente omitir alguna capa en la red neuronal provocando que el gradiente llegue a capas más profundas. Si no se hiciera esto el gradiente se vuelve muy pequeño (incluso podría ser nulo) a medida que se acerca a las primeras capas, llegando,

[3] La precisión del modelo con su primera predicción.
[4] La entrada a la red neuronal es la capa inferior y la salida es la capa superior.

en algunas situaciones a no aplicar ninguna actualización a los pesos de las primeras capas [30].

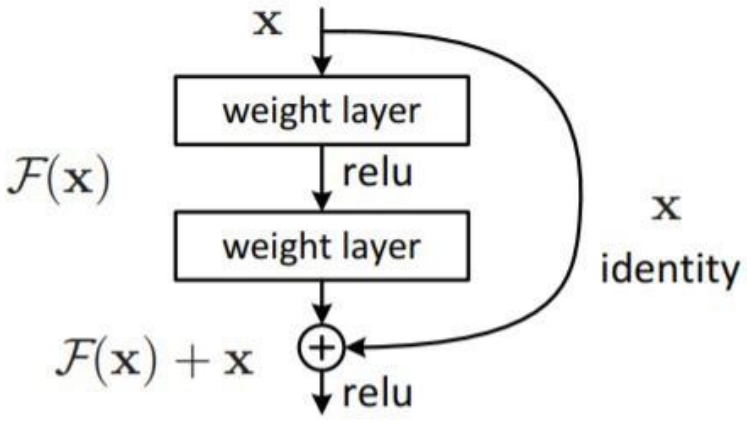

Ilustración 2.7 Bloque residual. Fuente: [4].

Más adelante, en 2018, se crea el modelo *DenseNet* [31], con la idea de crear un modelo más profundo, con mejores resultados y más eficiente. La idea principal era que cada capa estuviera conectada a todas las demás capas en forma de retroalimentación. Sería algo similar al bloque residual de la *ResNet*, pero aplicado a todas las capas (véase la Ilustración 2.8).

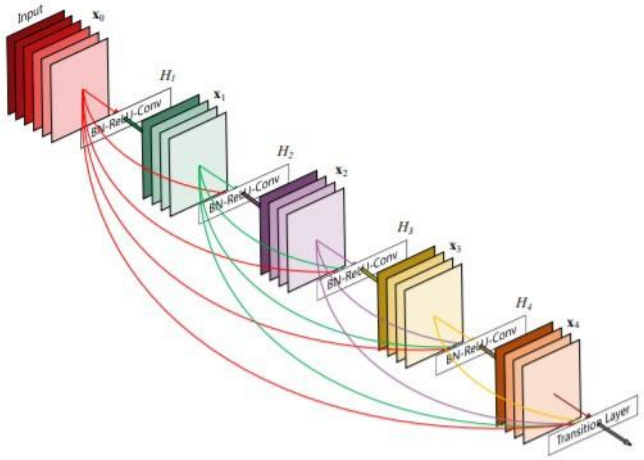

Ilustración 2.8 Arquitectura de DenseNet. Fuente: [31].

Todos estos avances en la competición de Imagenet han hecho que muchos investigadores hayan estado intentando crear arquitecturas cada vez mejores (en relación con el *accuracy* de los modelos). Ahora bien, no todos los modelos se centran solo en mejorar esta métrica. En el caso de *EfficientNet* [32], p.e., esta arquitectura creada en 2020 tiene como principal objetivo el escalado de la arquitectura de forma eficiente consiguiendo, además, mejorar el estado del arte. Para ello, esta aproximación se apoya en 3 dimensiones: la profundidad, el ancho y la resolución. La profundidad consiste en el número de capas de la red. El ancho es el número de canales que existen en una capa convolucional. Finalmente, la resolución es el tamaño de la imagen de entrada. Para ver más información véase la página 62 en los anexos.

Finalmente, cabe destacar el *Visión Transformer* (*ViT* en adelante). Su principal innovación en visión artificial es utilizar una arquitectura codificador-decodificador (*encoder-decoder)* que ha estado dando buenos resultados en problemas de *Natural Language Process* (*NLP* o proceso de lenguaje natural en español). Para ello se siguen los siguientes pasos:

1. Dividir la imagen en diferentes recortes.
2. A cada recorte se le añade información de donde se encuentra el corte.
3. A continuación, estas imágenes se transforman en vectores plano junto con la información de la posición.
4. Este vector plano se pasa por el encoder (codificador)
5. Al final del encoder se utiliza una red multicapa para obtener la clase

Se puede observar la arquitectura en la Ilustración 2.9. También se desea comentar que en el *ViT* original cada corte es independiente del resto, es decir, no existe solape entre los cortes. Pero en este trabajo, uno de los modelos que se usará en los experimentos si dispondrá de solape entre imágenes, lo cual provocará que el vector de entrada al Transformer tenga una dimensionalidad mayor. Más información sobre estas arquitecturas puede ser encontrada en los anexos.

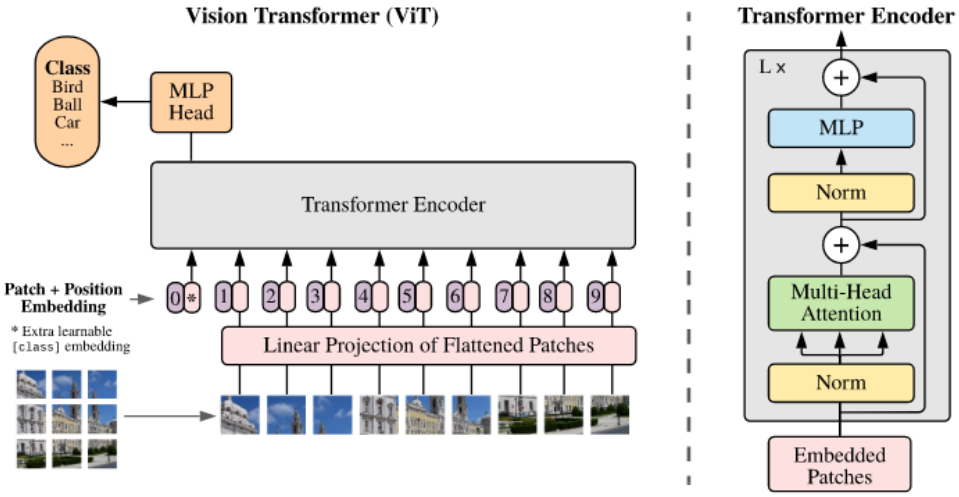

Ilustración 2.9 Arquitectura de Vit. Fuente: [33].

Como ya se ha mencionado, el problema que se plantea solucionar en este proyecto se enmarca en clasificación de hilo fino (a partir de ahora *fine-grained* [34]). Estos problemas se diferencian de la clasificación de imagen "estándar" en que las clases a predecir son complejas de diferenciar incluso para un experto en el área. Por ejemplo, especies de aves, flores o nuestro problema, la identificación de marca/modelo y fecha de lanzamiento de vehículos. En la Ilustración 2.10 puede observarse una comparación de los dos tipos de problema.

Ilustración 2.10 Problema de clasificación de hilo fino (clasificar entre tipo de cuervo) vs clasificación "estándar" (clasificar entre cuervo o vaca). Fuente: [35].

La clasificación de hilo fino suele tratarse de forma un poco distinta a la clasificación "estándar" ya que, en lugar de buscar características generales en las imágenes en estos problemas, se tiende a buscar la característica que las diferencia dentro de las clases. Por ejemplo, en nuestro problema, la parte frontal del vehículo suele ser lo que mejor permite diferenciar el tipo de vehículo.

Igualmente, se ha de recalcar que los principales retos de la visión artificial para ambos tipos de problemas residen en la cantidad de datos que son necesarios. Ya que si se usan pocas imágenes, como se hablará más adelante, la red puede memorizar las imágenes que se utilizan en lugar de aprender conceptos abstractos de las mismas para ello se utilizan técnicas de aumentación de datos (de ahora en adelante *Data Augmentation*).

Pero este no es el único reto, la elección de la velocidad de aprendizaje (*learning rate* de ahora en adelante) también es un reto como se observará en los experimentos. Además de todo lo anterior, depende del problema a resolver y de cómo se desee abordar la elección de una función de pérdida que la red tenga como objetivo mejorar puede variar mucho los resultados, los tipos de funciones de pérdida que se usarán en este trabajo se abordarán con más detalle más adelante.

¿Qué es y para qué sirve el *Data Augmentation*?

Como se ha comentado recientemente las redes neuronales son muy demandantes de datos ya que necesitan de una gran cantidad de imágenes para que puedan adaptar los pesos internos de forma que aprenda de verdad, es decir, que sean capaces de llevar a cabo generalizaciones a partir de los ejemplos de entrada. Si el modelo no es capaz de generalizar, existe el problema de que sobreajuste (*overfitting*[5]) y, por tanto, no sea capaz de funcionar adecuadamente con nuevos ejemplos. Si un modelo generaliza bien, será capaz de clasificar correctamente una imagen que no haya visto previamente (i.e. no la haya visto durante el entrenamiento del modelo). Hay distintas técnicas que se pueden emplear para evitar el problema del sobreajuste. Uno de ellos es usar redes pre-entrenadas, es decir, transferir los pesos del mismo modelo, pero de otro problema/dominio al modelo que se va a utilizar (p.e. con el modelo entrenado en Imagenet con 1000 clases [28]). Estos pesos se transfieren a nuestro modelo y, mediante un entrenamiento/refinamiento selectivo, reajustar los pesos necesarios. La principal ventaja de esto es que se necesitan mucho menos datos para reajustar la red a un problema específico. Esta práctica es llamada transferencia de aprendizaje (de ahora en

[5] Memorizar los datos de entrenamiento.

adelante *transfer learning*). Esto permite acelerar los procesos de aprendizaje (se necesitan muchas menos imágenes para reentrenar una red) pero aún existe el problema del *overfitting* o sobreajuste.

Para evitar el problema del *overfitting*, durante el proceso de refinamiento de redes pre-entrenadas, también se tiende a aplicar *Data Augmentation*. Esto consiste en aumentar la cantidad de imágenes de las que se dispone aplicando diversas transformaciones o combinaciones de estas. Algunas de estas transformaciones se pueden observar en la Ilustración 2.15 donde ven distintos ejemplos de *Data Augmentation* los cuales solo modifican los valores de los píxeles. Aparte de estas existen otros ejemplos de transformación son:

- Redimensionamiento de imagen.
    - Redimensionar el tamaño de la imagen al valor que se desea, tanto ampliando la imagen como reduciéndola (véase la Ilustración 2.11).
- Corte aleatorio (a partir de ahora *random crop*).
    - Cortar la imagen en una posición aleatoria a un tamaño determinado, muy útil para reducir la dimensionalidad de la imagen sin perder calidad (véase la Ilustración 2.12).
- Rotación de imagen.
    - Girar la imagen la cantidad de grados que se decida, el valor puede ser aleatorio (véase la Ilustración 2.13).
- Voltear la imagen.
    - Voltear la imagen eje que se desee (véase la Ilustración 2.14).

Todas estas transformaciones se pueden combinar para obtener un número muy amplio de imágenes de entrada con solo una imagen original, además es debido a estas transformaciones que el área de aprendizaje auto supervisado está en auge.

Ilustración 2.11 ejemplo de función Resize de Pytorch. Fuente: [17].

Ilustración 2.12 ejemplo de función RandomCrop de Pytorch. Fuente: [17].

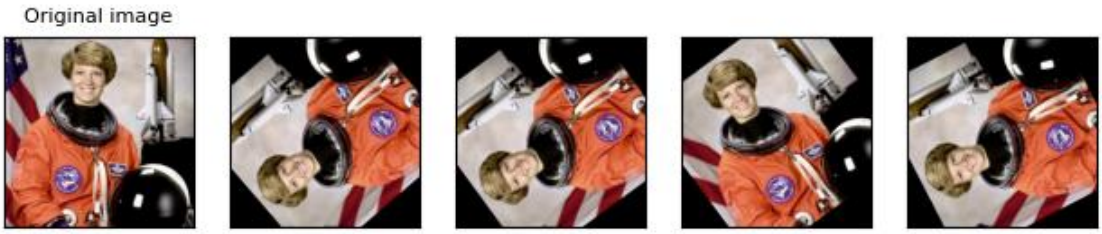

Ilustración 2.13 ejemplo de función RandomRotation de Pytorch. Fuente: [17].

Ilustración 2.14 ejemplo de función RandomHorizontalFlip de Pytorch. Fuente: [17].

Ilustración 2.15 Conjunto de imágenes que han sufrido de augmentation. Fuente: [36].

En los experimentos que se presentan en este trabajo se ha aplicado *Autoaugment* [37] Ésta es una librería que aplica en cada iteración del entrenamiento una política distinta de "*Data Augmentation*". En el apéndice A de [37] se puede encontrar más información al respecto. Además, en la Ilustración 2.16 se puede observar distintas políticas aplicadas a la misma imagen usando *Autoaugment*.

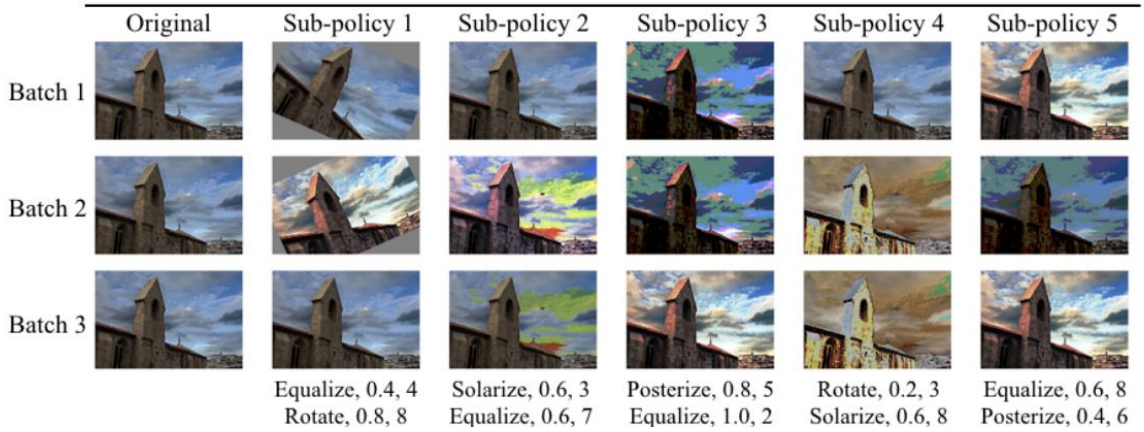

Ilustración 2.16 ejemplo de distintos batch de imágenes usando distintas sub políticas de Autoaugment. Fuente: [37].

Por otro lado, para aumentar aún más el número y variedad de las imágenes de entrada, también se aplicará *CutMix* [38] Esta librería es la combinación de *Mixup* [39] y un *Cutout*. En la Ilustración 2.17 se puede observar que *Mixup* es la combinación de dos imágenes, por lo cual su etiqueta es la combinación de estas dos imágenes. Por otro lado, *Cutout* permite generar más imágenes mediante la eliminación de una parte de las imágenes originales.

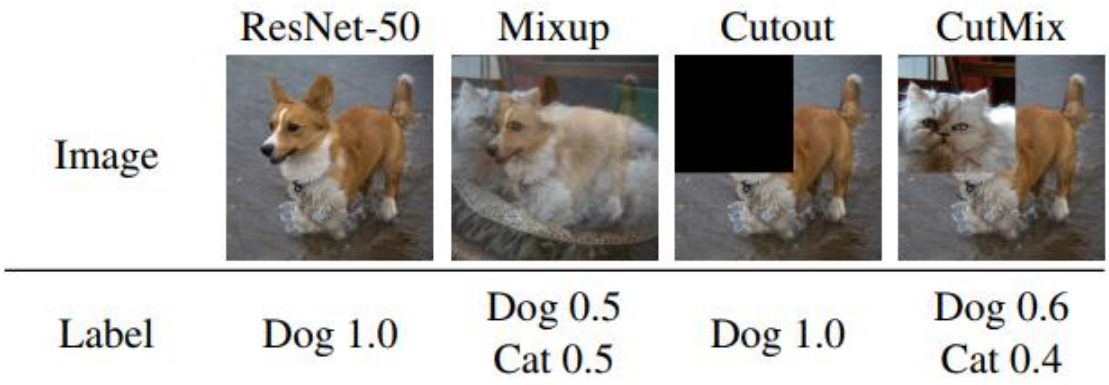

Ilustración 2.17 Ejemplos de aplicar Mixup, Cutout y la combinación, CutMix. Fuente: [38].

Con estas técnicas se pretende disminuir el *overfitting* que se pueda producir en las distintas redes neuronales.

Funciones de pérdida

Como se ha comentado anteriormente, las funciones de pérdida son necesarias porque su minimización es el objetivo que tiene la red neuronal a la hora de entrenar. De esta manera, a grandes rasgos, las redes neuronales actualizan sus parámetros (aprenden) a partir de la propagación de los cambios necesarios para la minimización de las funciones mencionadas. Esta propagación, habitualmente, se lleva a cabo mediante la técnica conocida como "*backpropagation*".

Existen muchas funciones de pérdida y su uso depende del problema y cómo se desee abordarlo. En este trabajo en concreto se centra en 3 funciones de pérdidas:

- La función de pérdida *Crossentropy* (entropía cruzada) [17].
- *Contrastive Loss* (contrastivo en español, aunque es mejor planteárselo como función de similitud de imágenes) [40].
- *Triplet Loss* (la función de pérdida de tripleta) [41].

Se han elegido estas 3 funciones de pérdida debido a que la *Crossentropy* es la función de pérdida estándar y más utilizada para todos los problemas de clasificación en visión artificial. En cuanto a la *Contrastive Loss* y la *Triplet Loss* se han escogido debido a que entre sus cualidades (las cuales se nombran con más detalle más adelante) se encuentra aproximar imágenes similares dentro de un espacio latente en el caso de ambas o en el caso de la *Triplet loss* también aleja las imágenes diferentes dentro de ese espacio.

La entropía cruzada es una medida que te da la diferencia entre dos o más distribuciones de probabilidad. La fórmula de esta función de pérdida se puede apreciar en la Ecuación 2.2.

Ecuación 2.2 formula de la entropía cruzada. Fuente: [42].

$$L_{\text{cross-entropy}}(\hat{\mathbf{y}}, \mathbf{y}) = -\sum_{i} y_i \log(\hat{y}_i)$$

Tabla 2.1 Ejemplo de datos para explicar la entropía cruzada. Fuente: elaboración propia.

Etiquetas	A	B	C
Distribución de la etiqueta real	1	0	0
Distribución predicha por el modelo	0.7	0.1	0.2

La información presente en la Tabla 1 se utilizará para ilustrar el funcionamiento del *Crossentropy*. En esta tabla se muestra un ejemplo artificial muy sencillo de etiquetado real (fila 1) y la predicción llevada a cabo por una red neuronal convolucional con una última capa de tipo *SoftMax*[6] (fila 2). En este caso, la etiqueta de la imagen es A y el modelo ha predicho que solo existe un 70% de probabilidad de que la imagen pertenezca a A, un 10% de probabilidad que pertenezca a B y un 20% que sea C. Aplicando la fórmula el resultado sería el siguiente *Loss*=-(1*$\log_2$(0.7) +0*$\log_2$(0.1) +0*$\log_2$(0.2)) = 0.5145. Optimizando el modelo con esta fórmula se consigue optimizar que el modelo prediga con la mayor probabilidad posible la clase correspondiente.

Respecto a *Contrastive Loss* y *Triplet Loss* [41] , éstas están muy relacionadas entre ellas. La idea principal de estas dos funciones es crear un espacio latente, en el cual las clases más similares estén más adyacentes entre ellas. Para entender un poco mejor qué es el espacio latente en la Ilustración 2.18 se puede observar el espacio latente del dataset MNIST[7] el cual se ha transformado a un espacio en 2D mediante una reducción de variable con t-SNE. En la imagen se puede apreciar que existen distintos clústeres correspondientes a cada clase. La idea de *Contrastive Loss* y *Triplet Loss* es crear este espacio latente. La idea principal de estas dos funciones es crear un espacio latente, en el

[6] Esta capa permite transformar los valores del espacio latente anterior en un vector donde sus valores sumen 1 por lo que sirve para ver la probabilidad de que una etiqueta sea correcta en la predicción.
[7] Dataset que consiste en dígitos entre el 0 y el 9 escritos a mano.

cual las clases más similares estén más próximas entre ellas. Para entender un poco mejor que es el espacio latente, en la Ilustración 2.18 se puede observar el espacio latente del dataset MNIST el cual se ha transformado a un espacio en 2D mediante una reducción de variable con t-SNE. En la imagen se puede apreciar que existen distintos clústeres correspondientes a cada clase. El objetivo de *Contrastive Loss* y *Triplet Loss* es crear este espacio latente maximizando las distancias entre las distintas clases.

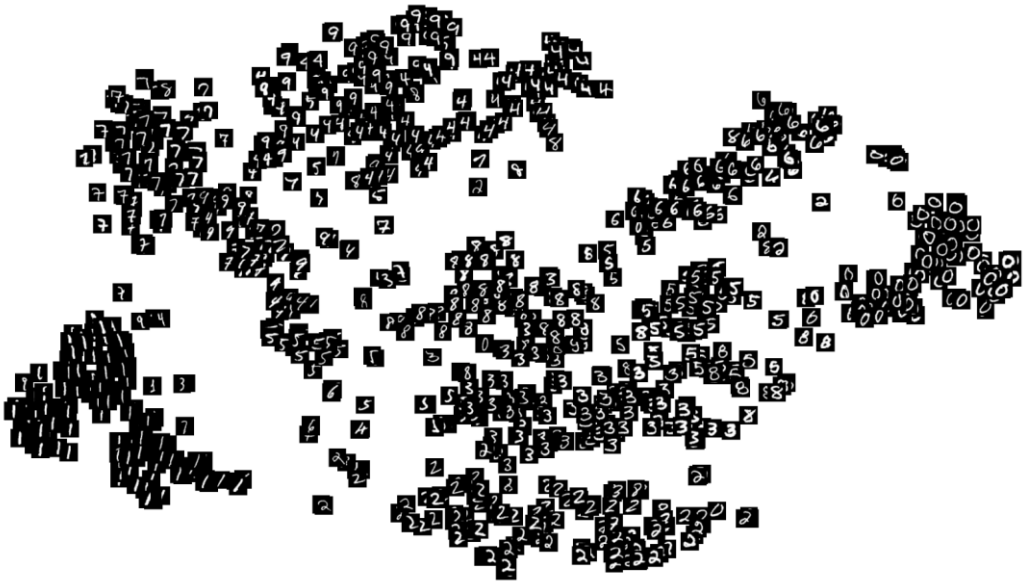

Ilustración 2.18 plano de espacio latente resumido mediante t-SNE en dos dimensiones. Fuente: [43].

En el caso de *Contrastive Loss* se utilizará el mismo planteamiento que se utiliza en *Simsiam* [44]. Éste consiste en utilizar redes siamesas (es decir, redes que comparten sus pesos), donde la misma imagen se le aplica dos transformaciones distintas mediante" *Data Augmentation* ". A continuación, estas imágenes se pasan por la red neuronal, la cual funciona como un *encoder* (codificador). Una de ellas se la pasa por una capa multiperceptron y entre ambas salidas se calcula la similitud, usando la distancia del coseno. Esta función de pérdida no necesita etiquetas y está planteada para aprendizaje auto supervisado (más información en el siguiente *survey* [45]).

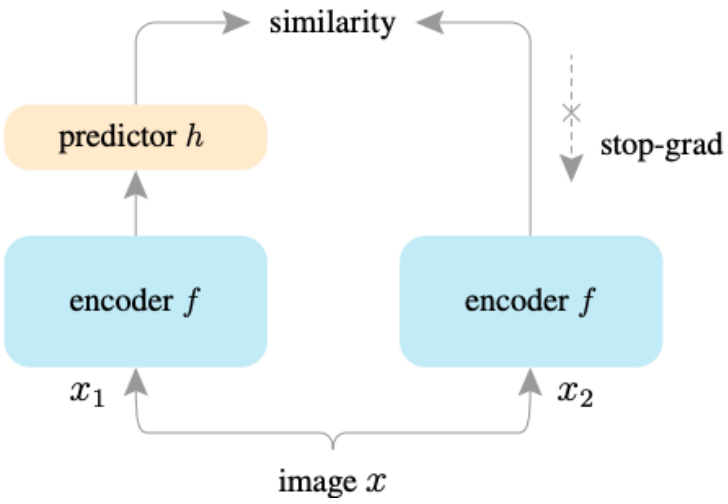

Ilustración 2.19 Arquitectura de Simsiam. Fuente [44].

En el caso de *Triplet Loss* su objetivo es reducir la distancia entre la etiqueta positiva y la imagen "ancla" y, además, alejar la imagen con etiqueta negativa de la imagen ancla. En la Ilustración 2.20 se puede apreciar perfectamente su funcionamiento. En la Ilustración 2.21 se puede observar también su funcionamiento en un plano en 2D.

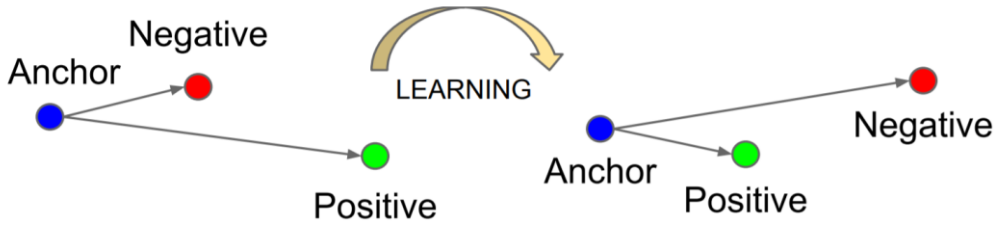

Ilustración 2.20 Triplet loss minimiza la distancia entre la etiqueta ancla y la etiqueta positiva y a la vez maximiza la distancia entre el ancla y la etiqueta negativa. Fuente: [41].

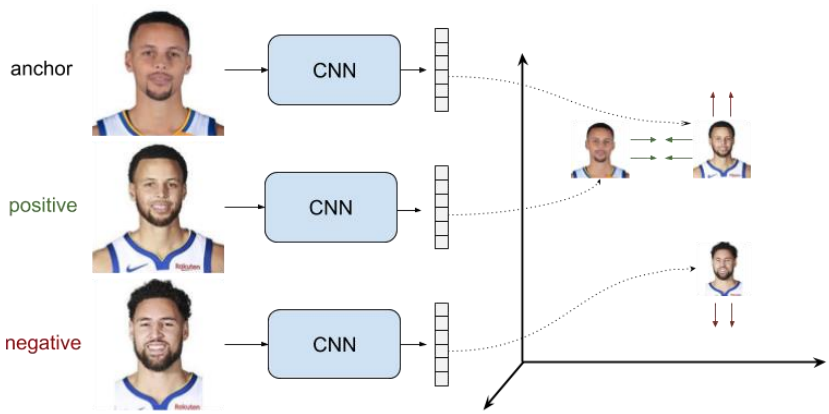

Ilustración 2.21 funcionamiento de Triplet Loss en un espacio en 2D. Fuente [46].

Por último, cabe mencionar que estas funciones de pérdida no se tienen porque utilizar de forma independiente. En la Ilustración 2.22, p.e., se aplica *Contrastive Loss* para ordenar el espacio latente y luego *Crossentropy* para darle peso a la correcta predicción de la clase.

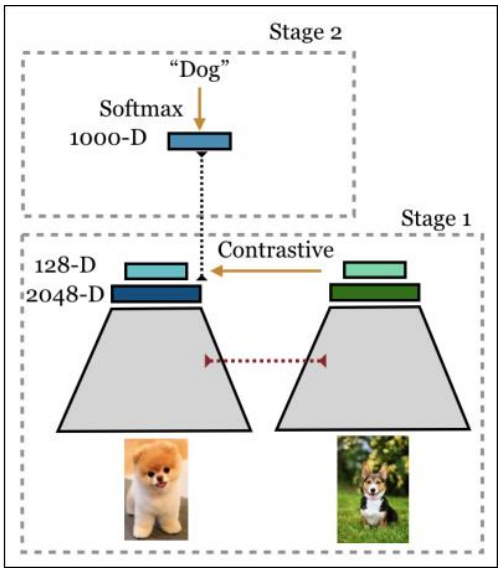

Ilustración 2.22 Utilización de diversas funciones de pérdida en el mismo problema. Fuente: [40].

Trabajos relacionados con fine-grained en detección de vehículos

En cuanto a trabajos relacionados que estén enfocados en concreto a solucionar el problema objetivo de este trabajo (clasificación de vehículos al más bajo nivel) nos podemos encontrar estrategias diversas. Una de ellas se aplica en [47] cuyo planteamiento consiste en extraer las características más representativas del objeto de interés, desde las partes más globales a las más locales. En la Ilustración 2.23 se puede observar el flujo de tareas propuesto con este objetivo. En este caso, el flujo de tareas consiste en utilizar las redes convolucionales (*CNN*) para extraer características globales y detectar las subregiones diferenciadoras de clase de forma automática. Para ello hace uso de la última capa de la *CNN* para extraer un mapa de características y de estas quedarse con las más relevantes. En base a esto, determina las zonas de la imagen que corresponde a estas características y las reintroduce en una *CNN* para continuar el análisis. Este proceso se sigue a 3 niveles de detalle (ver ilustración 3). El resultado de cada *CNN* en cada nivel da como resultado un vector de características. Estos tres vectores de características resultantes de cada *CNN* se usarán de entrada en un *Support Vector Machines* (*SVM*) que llevará a cabo la clasificación final. Este trabajo tuvo un *accuracy* de 98,29% en el dataset de Compcars [48] pero su objetivo era solo la marca y el modelo del vehículo, por lo que no incluía la fecha de lanzamiento a diferencia de nuestra aproximación.

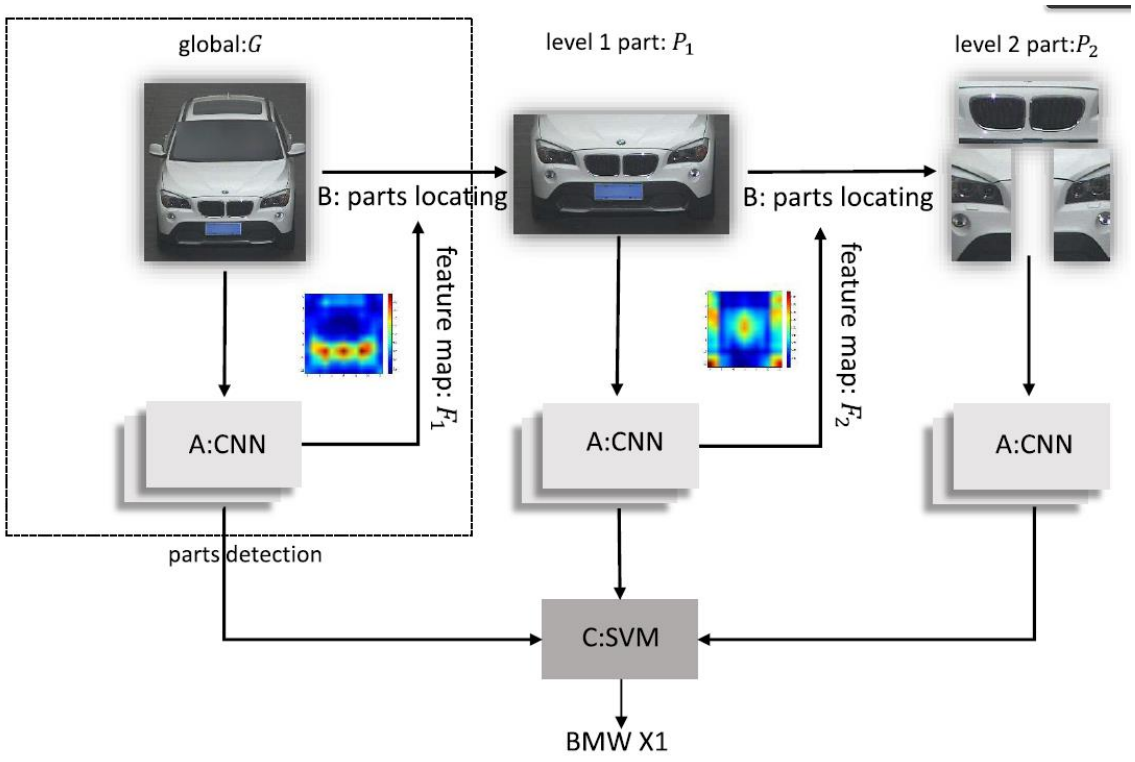

Ilustración 2.23 Framework propuesto por [47] para detección de modelo y marca de vehículo. Fuente: [47].

Otro trabajo también mencionable es el de [49]. El cual utiliza el dataset VMMRdb [50], el cual es un conjunto de datos extraído usando cámaras reales en siete zonas urbanas en Corea. Al igual que el anterior la clasificación que realiza es de marca y modelo lo que con 766 clases y con un *accuracy* de 96,33%. En cuanto al trabajo en concreto consta de dos partes diferenciadas. La primera parte consiste en un framework para semi-

automatizar el etiquetado de los vehículos, reduciendo tiempos de etiquetación manual (ver Ilustración 2.24).

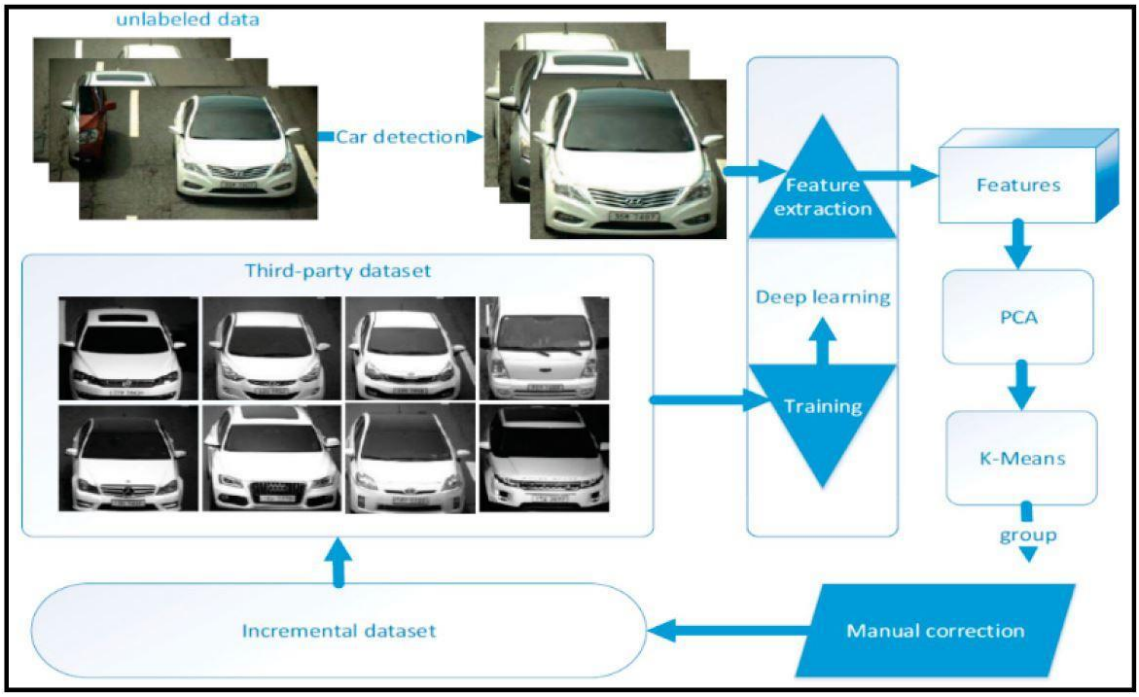

Ilustración 2.24 framework para etiquetado por clúster. Fuente: [49].

La segunda parte implementa el algoritmo usado para la clasificación, el cual consiste en el uso de una arquitectura basada en *SqueezeNet* [51] (ver Ilustración 2.25).

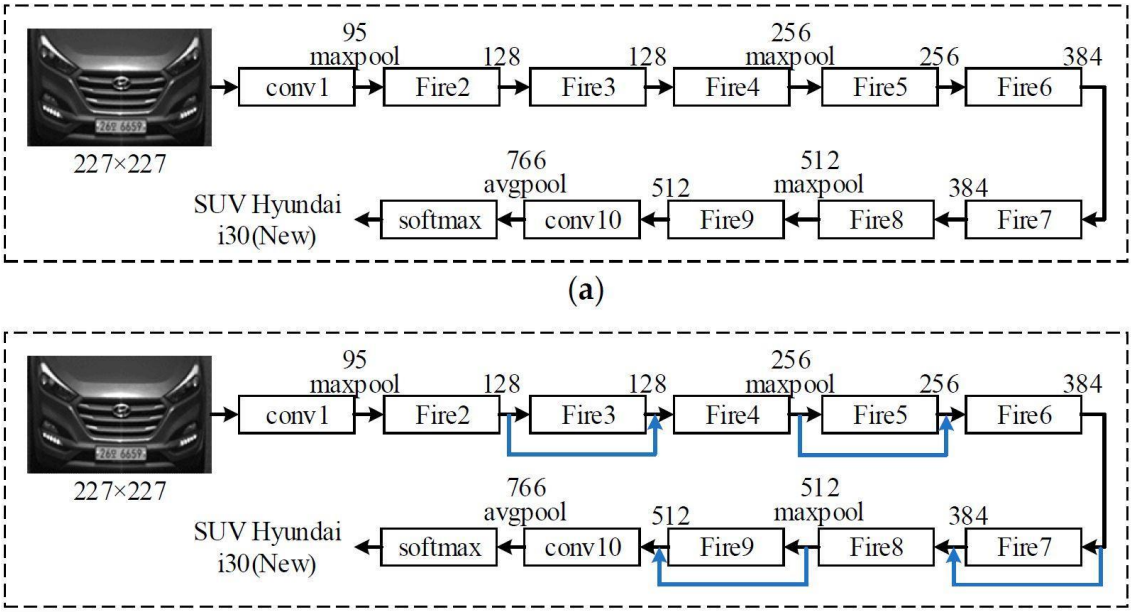

Ilustración 2.25 (a) architecture SqueezeNet. (b) arquitectura Residual SqueezeNet propuesta por [49]. Fuente: [49].

Estos dos trabajos mencionados anteriormente son trabajos donde se centran en el problema de detección de vehículos (aunque solo marca y modelo). Por otro lado, tenemos muchos trabajos los cuales están centrados en la mejora del arte del problema de clasificación de imagen (tanto como para *fine-grained* como para clasificación

estándar). Donde los resultados los aplican sobre una gran variedad de datasets, uno de ellos *Cars196* o *Stanford Cars* (el cual se usará en este trabajo y se explica en la siguiente sección con más detalle) y cuyo estado del arte se muestra en la Ilustración 2.26 donde se observa que estos últimos 5 años el estado del arte ha mejorado aproximadamente el *accuracy* en un 5%. Con mejoras en diversos puntos en el *Deep learning*, algunos de estos puntos son:

- Mejoras en las propias arquitecturas [52] [32].
- Mejoras en relación con las imágenes de entrada aplicando *Data Augmentation* [37].
- Añadiendo mecanismos de atención [52] [53].
- Introduciendo nuevas funciones de pérdidas [41] [54].

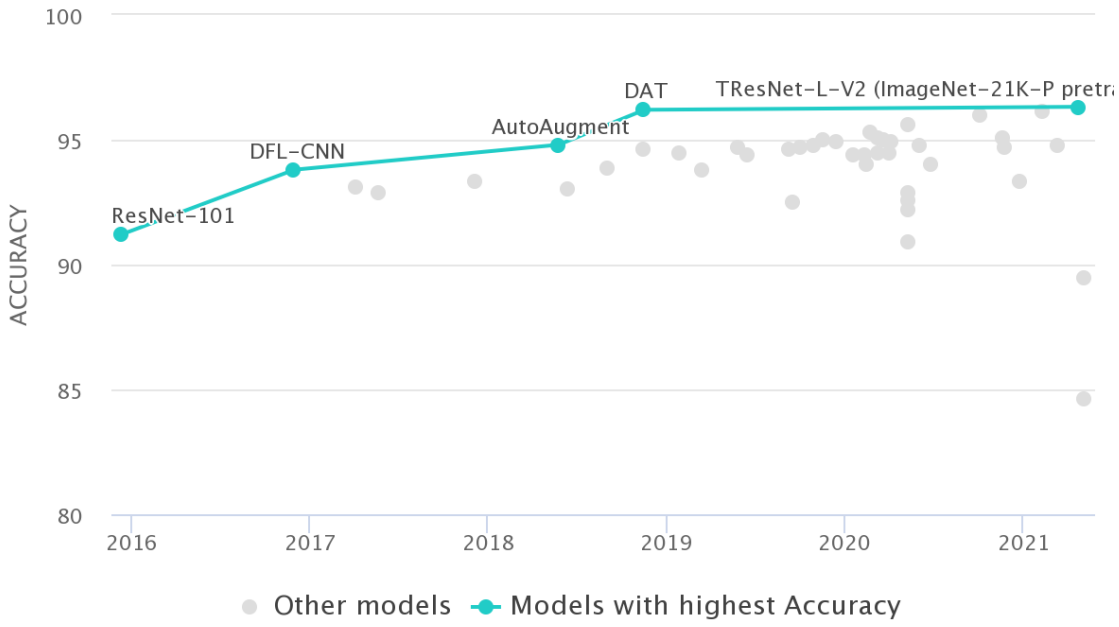

Ilustración 2.26 estado del arte en fine-grained en cars196. Fuente: [22].

En el presente trabajo se va a tratar de alcanzar el estado del arte actual aplicando las distintas técnicas que se encuentran en estos trabajos o combinación de estos. Para ello se va a proceder a explicar los experimentos que se van a realizar.

3. Entrenamiento de modelos

En esta sección se procederá a explicar los experimentos que se han realizado al igual que se explican los parámetros elegidos tanto en el setup como en los distintos experimentos.

En primer lugar, se explica el setup inicial que se utiliza y, en el cual, se apoyan los distintos experimentos y, a menos que se diga lo contrario, para realizar el experimento en cuestión los parámetros son los mismos que en el setup. En segundo lugar, se procederá a explicar el estudio de distintas arquitecturas y modelos. En el tercer experimento se demuestra la utilidad de utilizar transfer learning en los modelos. A continuación, se procede a realizar otro experimento con distintos tipos de *Data Augmentation* y sin *Data Augmentation*. Una vez finalizado esto, se han realizado otras pruebas con distintas combinaciones de learning rate para ver cómo afecta a los modelos elegidos y, de ese modo, acercarnos al mejor *learning rate* posible. También se pretenden utilizar distintas funciones de pérdida o combinación de estas para ver cual o cuales aportan mejores resultados. Finalmente, se ha realizado un experimento con los mejores parámetros obtenidos de los distintos experimentos previos, con el que se espera obtener el mayor *accuracy* posible y que consiste en calcular el porcentaje de aciertos de la clase correcta. La utilización de la métrica de *accuracy* y no otra métrica es debido a dos motivos. El primero de ellos es que es la métrica principal en visión artificial en problemas de clasificación y en concreto en todos los *papers* donde el dataset que vamos a usar se utiliza el accuracy en el conjunto de datos de validación. Y el segundo motivo es que se ha tenido en cuenta que los datos se encuentran lo suficientemente balanceados.

Setup Inicial

Para el Setup inicial se utilizó el dataset de Car de Standford [55], también conocido como Cars196. Este dataset consta de 196 clases, las cuales definen la marca, el modelo y la fecha de lanzamiento del vehículo. En relación al tamaño del dataset, este dispone de 16.185 imágenes separadas en 8.144 para entrenamiento y 8.041 para validación. Además, en PaperswithCode [22] en la tarea de clasificación de imagen de hilo fino (*Fine-Grained*), existe un *benchmark* para este dataset y cuyo estado del arte actualmente es de un accuracy del 96.32%. En este *benchmark*, para comparar los distintos artículos, se ha observado (accediendo al código disponible de algunos de ellos) que el preprocesado que realizan es el siguiente:

1. Redimensionar la imagen a 600x600 pixeles.
2. En la fase de entrenamiento aplican un *random crop* y en la fase de validación un *center crop* hasta tener, en ambos casos una imagen de 448x448 píxeles

Por tanto, esto es algo que no se ha modificado en ninguno de los experimentos planteados. Además, en cuanto a la normalización de los datos, se han aplicado los mismos parámetros que se han usado para el modelo pre-entrenado.

Para los distintos experimentos, exceptuando el experimento de diferentes modelos y arquitecturas, se ha utilizado el modelo *ResNet50* y *EfficientNetB4*. Esto es debido a que son computacionalmente más sencillos y aportan buenos resultados. El criterio que se

ha seguido es escoger los dos mejores modelos cuyo *epoch*[8] de media tarde menos de 4 minutos, es decir, que su entrenamiento es más rápido que el resto. Los 3 modelos que cumplían la característica después de haber realizado el primer experimento era *Densenet121*, *ResNet50* y *EfficientNetB4* (el tiempo necesario por minuto de cada *epoch* era de 3,5/3,4/3,6 respectivamente). Pero en relación con el *accuracy* los dos últimos eran superiores. En cuanto a la velocidad del resto de modelos se tarda más de 4 minutos por epoch siendo en algunos casos superior a 5,5 minutos por epoch. Para más información véase la Ilustración 3.1 para ver los valores en detalle.

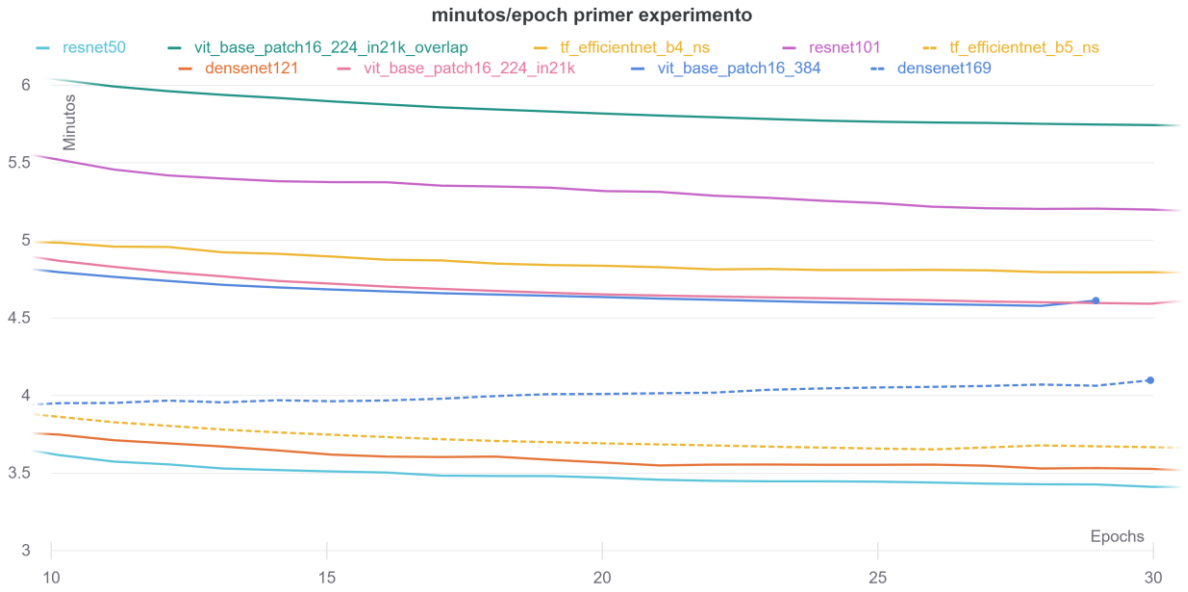

Ilustración 3.1 tiempo medio por cada epoch entre el epoch 10 y el epoch 30 del experimento 1. Fuente: elaboración propia.

También muchos de estos modelos estaban pre-entrenados y cogidos de Timm[9] [56], lo cual ha sido de gran ayuda en algunos de los experimentos realizados. Además, siempre y cuando el modelo lo permita, el tamaño del lote (de ahora en adelante *batch size*) ha sido de 50. Si esto no ha sido posible por problemas de memoria, se notifica en el experimento. En cuanto a la función de pérdida se ha usado la de *Crossentropy* por ser la más utilizada en problemas de clasificación de imagen debido a su eficacia y a su fácil implementación debido a su estandarización en cualquier librería de *Deep learning* (TensorFlow o Pytorch, por ejemplo).

En relación a la cantidad de máxima *epochs* que se realizarán por cada instancia de los experimentos, ésta ha sido de 50 *epochs* debido a que es un valor lo suficientemente grande para que la red neuronal llegue a converger, además epochs más alto supondría un tiempo de entrenamiento mayor y mayor coste de computación. Como adición, al entrenamiento se le ha añadido la posibilidad de parar antes si la métrica de accuracy en validación no mejora durante 10 *epochs* seguidos. Esto es por dos motivos: el primero para evitar *overfitting* y el segundo para ahorrar tiempo de computación.

[8] Un epoch es una iteración en todo el conjunto de imágenes.
[9] Librería de *Deep Learning* que contiene muchas arquitecturas del estado del arte y sus pesos pre-entrenados en Pytorch.

En cuanto al *learning rate* en los 5 primeros *epochs* aumentará linealmente de 0 a 0.0035[10] , para posteriormente decaer con la función del coseno, disminuyendo de esta manera lentamente hasta 0 (véase la Ilustración 3.2 más adelante para aclarar la forma que sigue el *learning rate*). El motivo de que incremente linealmente solo 5 *epochs* es debido a que es el 10% de los *epochs* del experimento y observando distintos repositorios de *papers* relacionados con este problema [52] siguen este patrón. En cuanto al optimizador que se usará, será el SGD (*Stochastic Gradient Descent*) con un *momentum*[11] de 0.9. Además, para reducir la necesidad de cómputo los tensores serán de 16 bits en lugar de 32 bits con esto se podrá aumentar el *batch size* y realizar los entrenamientos en un menor tiempo. En la Tabla 2 se puede ver un resumen del Setup inicial.

Tabla 3.1 Setup inicial. Fuente: elaboración propia.

Variable	Valor
Dataset	Cars196
Imágenes en entrenamiento	8144
Imágenes en validación	8041
Batch size	[50,16]
Epochs	50
Variable *early stopping*	Validation *accuracy*
Patience early stopping	10
Valor más alto de learning rate	0.0035
precisión	16 bits
Función de perdida	*Crossentropy*
Optimizador	Sgd -> momentun 0.9

Comparativa entre distintos modelos/arquitecturas

Las arquitecturas con sus respectivos modelos que se han utilizado en los diferentes experimentos son:

- *ResNet:*
  - Resnet50
  - Resnet101
- *DenseNet*:
  - Densenet121
  - Densenet169
- *EfficientNet*:
  - Tf_efficientnet_b4_ns
  - Tf_efficientnet_b5_ns
- *Vit*:
  - Vit_base_patch16_224_in21k
  - Vit_base_patch16_384
  - Vit_base_patch16_224_in21k_overlap

[10] Este valor se ha seleccionado porque después de realizar una serie de experimentos con distintos learning rate fue el valor con mejor accuracy.
[11] El momentum acumula el gradiente de los anteriores batch y se reutiliza este valor para realizar una pequeña corrección al gradiente en lugar de utilizar solo el batch en concreto.

Como se puede observar, no solo se han seleccionado distintas arquitecturas (*Resnet*, *DenseNet*, *EfficientNet*, *Vit*) sino que también se ha escogido distintos modelos de éstas. La principal diferencia entre los modelos de la misma arquitectura puede ser el número de capas, el tamaño del modelo o cual es el modelo pre-entrenado que se utiliza.

En el caso las arquitecturas *ResNet* y *DenseNet*, varía solo el número de capas que se utilizan, lo cual también afecta al tamaño del modelo. En el caso de *EfficientNet* entre ambos modelos existen muchas modificaciones (ver sección dos y en el anexo de descripción de arquitecturas).

Como resumen se puede decir que B5 es más grande en las 3 dimensiones que se modifican en esta arquitectura y, por ende, tiende a obtener mejores resultados a costa de mayor necesidad de cómputo. Por último, la diferencia entre los modelos de *ViT* es el tamaño de imagen de entrada para el que se entrenó (224x224 ó 384x384). Además, los marcados con in21k es que ha sido entrenado en Imagenet con 21.000 clases. Finalmente, el modelo que tiene *overlap* es que los cortes de las imágenes pueden estar superpuestas.

Otra diferencia entre los modelos es la cantidad de imágenes que se han usado en cada *batch*. En los modelos más pequeños se ha usado un *batch size* de 50 y en los más grandes de 16 (ver Tabla 3.2). Esto se debe, como ya se ha comentado, a restricciones de memoria.

Tabla 3.2 resumen de tamaño del batch de los distintos modelos. Fuente: elaboración propia.

Modelo	*Batch size*
Resnet50	50
Resnet101	50
Densenet121	50
Densenet169	50
Tf_efficientnet_b4_ns	50
Tf_efficientnet_b5_ns	16
Vit_base_patch16_224_in21k	16
Vit_base_patch16_384	16
Vit_base_patch16_224_in21k_overlap	16

El objetivo de realizar el estudio con distintas arquitecturas es analizar cuál es la arquitectura que mejores resultados da y qué requisitos tiene cada una de ellas.

Como resumen de los resultados (se recomienda ver estos resultados detenidamente más adelante) se ha decidido utilizar *EfficientNet* (B4 y B5) para el experimento final. Esto es debido a que *EfficientNet* B5 es el que mejor resultados ha dado de todos los modelos. Y en cuanto a B4 es el modelo, dentro de los más pequeños y que se ha utilizado en todos los experimentos, que mejor resultados ha dado (en la sección de resultados del learning rate se puede apreciar esto último).

Modelos pre-entrenados vs no pre-entrenados

Este experimento consistirá en demostrar la utilidad del *Transfer Learning* al problema abordado en este trabajo. Además, se desea probar distintas combinaciones de entrenamiento congelando las distintas capas de la red neuronal y entrenando solo la última capa. Solo se desea entrenar esta última capa porque el resto están pre-entrenadas con los pesos de Imagenet, pero esta capa ha sido sustituida para que esté acorde a nuestro número de clases.

Como se comenta en el setup, se utilizará solo el *Resnet50* y el *tf_efficientnet_b4_ns*. A continuación, se procede a explicar los experimentos que se han realizado:

- Sin pesos de redes pre-entrenadas
    - Esto consistió en crear el modelo y no cargarle información de la red pre-entrenada
- Con pesos pre-entrenados sin congelar capas
    - Esto consistió en crear el modelo, cargar los pesos de la red pre-entrenada y a partir de aquí reentrenar todas las capas del modelo
- Con pesos pre-entrenados y congelar todas sus capas, excepto la última
    - Esto consistió en crear el modelo, cargar los pesos de la red pre-entrenada y a partir de aquí reentrenar solo la última capa del modelo
- Con pesos pre-entrenados y congelar todas sus capas, excepto la última y en el *epoch* 25[12] descongelar todas las capas
    - Esto consistió en crear el modelo, cargar los pesos de la red pre-entrenada y, a partir de aquí, reentrenar solo la última capa del modelo hasta el *epoch* 25. En este punto, debido a tener un learning rate descendente el learning rate será menor que el máximo *learning rate* que utilizamos.

El objetivo, pues, de estos experimentos fue analizar la utilidad que tienen los modelos pre-entrenados. También han permitido analizar si se pueden reducir los tiempos de entrenamiento congelando distintas capas de la red neuronal sin perder calidad.

En base a los resultados que se obtuvieron se decidió utilizar modelos pre-entrenados. Además, en relación a la estrategia de congelación de capas se optó por no congelar estas capas.

[12]Lo ideal hubiera sido una batería de experimentos para elegir en que epoch se descongelan las capas. Pero debido a limitaciones computacionales no se ha podido realizar tal experimento y se optó por descongelar estas capas a la mitad del número de epoch para al menos obtener información sobre el comportamiento de las redes neuronales ante esta estrategia.

Experimento con *Data Augmentation*

En este experimento se desea analizar si las diferentes técnicas de *Data Augmentation* pueden mejorar los resultados obtenidos de los modelos cuando se aplican al problema a afrontar en este proyecto. Para ello se procedió a realizar los siguientes experimentos:

- Sin *Data augmentation:*
  - Este experimento consistió en no usar *Data Augmentation*. Esto nos sirvió de línea base. De este modo las transformaciones que se realizaron fueron las siguientes:
    - Redimensionamiento del tamaño de la imagen a 600x600 pixeles.
    - Realizar un corte central (a partir de ahora *center crop*) de tamaño 448x448 píxeles.
    - Transformar imagen a tensor.
    - Normalizar la imagen con media y desviación típica de Imagenet.
  - Se les ha denominado en los experimentos como cars_train_transforms_basic.
- Aplicando solo *Autoaugment:*
  - Consistió en aplicar las siguientes modificaciones a las imágenes de entrada:
    - Redimensionamiento del tamaño de la imagen a 600x600 pixeles.
    - Realizar un *random crop* de tamaño 448x448 píxeles.
    - Realizar un *random horizontal flip* con una probabilidad del 50%.
    - Aplicar *Autoaugment* (mencionado en la sección 2).
    - Transformar imagen a tensor.
    - Normalizar la imagen con media y desviación típica de Imagenet.
  - Se le denominó en los experimentos como cars_train_transforms_autoaugment.
- Aplicando solo *Mixup* y *Cutup:*
  - Consistió en aplicar las siguientes modificaciones:
    - Redimensionamiento del tamaño de la imagen a 600x600 pixeles.
    - Realizar un *center crop* de tamaño 448x448 píxeles.
    - Con una probabilidad del 75% se aplica *Mixup* o *Cutup*, de estas imágenes en el 50% se aplica *Mixup* con un alfa[13] de 0,3 y en las otras 50% se aplica *Cutup* con un alfa[13] de 0,3.
    - Transformar imagen a tensor.
    - Normalizar la imagen con media y desviación típica de Imagenet.
  - Se le denominó en los experimentos como cars_only_mixup.
- Aplicando *Autoaugment* y *Mixup* y *Cutup:*
  - Consistió en aplicar la combinación de las modificaciones de los dos experimentos anteriores:
    - Redimensionamiento del tamaño de la imagen a 600x600 pixeles.
    - Realizar un *random crop* de tamaño 448x448 píxeles.
    - Realizar un *random horizontal flip* con una probabilidad del 50%.
    - Aplicar *Autoaugment* (mencionado en la sección 2).

[13] Alfa es el porcentaje de imagen que se modifica

        - Con una probabilidad del 75% se aplica *Mixup* o *Cutup*, de estas imágenes en el 50% se aplica *Mixup* con un alfa[13] de 0,3 y en las otras 50% se aplica *Cutup* con un alfa[13] de 0,3.
        - Transformar imagen a tensor.
        - Normalizar la imagen con media y desviación típica de Imagenet.
    - Se le denominó en los experimentos como cars_autoaugment_mixup.
- Aplicando la misma transformación que se aplica en la competición de Imagenet:
    - Para ello se utilizó la implementación que viene por defecto en la librería Timm [56] (exceptuando el tamaño de la imagen) y consistió en aplicar las siguientes modificaciones:
        - Redimensionar la imagen al tamaño deseado (600).
        - *Random crop* hasta el tamaño deseado (448).
        - Realizar un *horizontal flip* con una probabilidad del 50%.
        - Realizar *color jitter*[14] con un 40% de probabilidad.
        - Transformar la imagen a tensor.
        - Normalizar la imagen con media y desviación típica de Imagenet.
    - Se le denominó en los experimentos como timm_transforms_imagenet_train.

Además, después de haberse realizado el experimento se ha observado interesante la realización de una prueba adicional combinando la transformación de Imagenet junto con la de *Autoaugment*. Por tanto, en los resultados se expondrá, además, este experimento extra.

Después de realizar el experimento correspondiente se decidió ampliar este experimento combinando las estrategias de *Autoaugment* junto con la de Imagenet debido a sus buenos resultados. Esta combinación resultó ser la mejor de todas las combinaciones por lo que se ha decidido usarla en el experimento final.

Experimento con distintos tipos de *Learning Rate*

Mediante este experimento se desea encontrar un *learning rate* con el que se obtenga buenos resultados. Este análisis es necesario debido a que si se utiliza un *learning rate* muy grande la red neuronal nunca convergerá, y si el learning rate es muy pequeño puede que alcanzase un óptimo local muy rápidamente y no sea capaz de obtener buenos resultados. Por ello, se desea probar de forma logarítmica distintos valores de *learning rate*. Además, para completar el estudio se analizaron distintos comportamientos del *learning rate*:

- Estrategias de *learning rate* que se usarán:
    - Aplicar un *learning rate* constante.
        - Se denominará en los gráficos constant.
        - La utilización de esta estrategia se justifica porque es la más fácil de implementar y la más usada inicialmente en los proyectos debido a esto.
    - Aplicar el *learning rate* del coseno desde el principio.
        - Se denominará en los gráficos cosine_descent.

[14] Cambios aleatorios de brillo, contraste, saturación y el tono de una imagen.

        - La utilización de esta estrategia se justifica debido a que inicialmente la red neuronal tiene mucho que aprender por lo que nos interesa un valor alto, pero a medida que va aprendiendo si el *learning rate* sigue siendo alto no permite a la red neuronal converger, por lo que para solucionar esto lo ideal es ir disminuyendo este valor poco a poco, en este caso en concreto con la función del coseno.
    - Aplicar el *learning rate* lineal creciente hasta el *epoch* 5 y luego un descenso del coseno. Véase la Ilustración 3.2 para visualizar mejor la evolución del *learning rate*.
        - Se denominará en los gráficos lineal_ascent_cosine_descent.
        - La utilización de esta estrategia se justifica debido a la misma justificación que en la estrategia anterior, con la diferencia de que en este caso los primeros epoch tenemos un *learning rate* ascendente hasta llegar al máximo que se defina, para posteriormente disminuir con la función del coseno, esta estrategia es utilizada en [57].
- Valores de *learning rate* a utilizar:
    - 0.1.
    - 0.01.
    - 0.001.

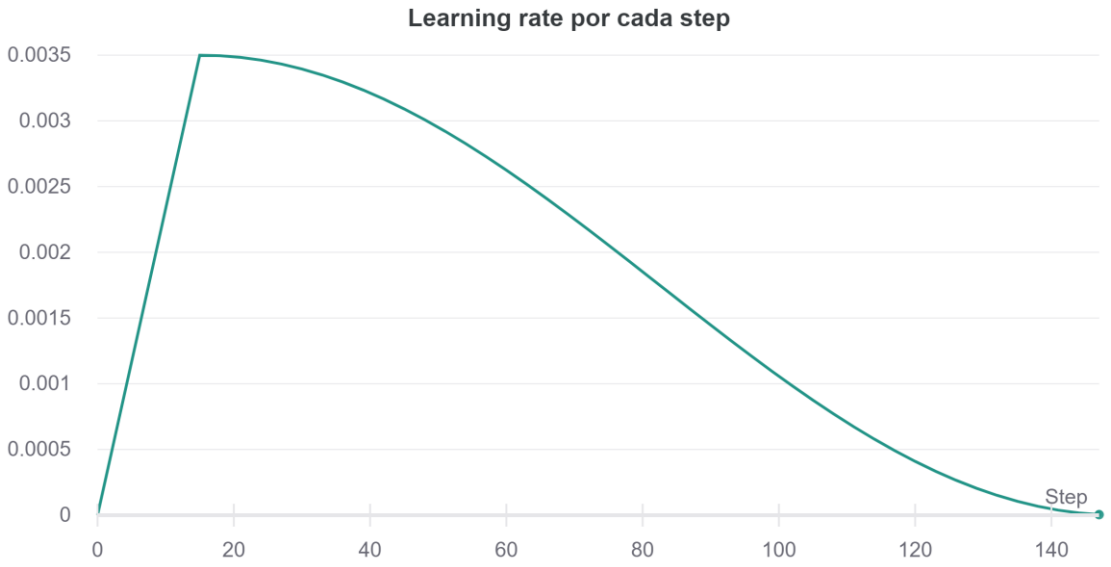

Ilustración 3.2 forma del learning rate aplicando un crecimiento lineal hasta un learning rate máximo (0,035 en este caso) y luego un descenso del coseno. Fuente: elaboración propia.

Se ha decidido utilizar estos valores de *learning rate* para utilizar una escala logarítmica y de esta manera poder descartar rápidamente valores demasiado alto o demasiado bajos de *learning rate*. Lo ideal sería aplicar una búsqueda más exhaustiva pero como se apreciará más adelante el valor de *learning rate* óptimo para cada modelo varía por lo que por limitaciones técnicas solo se hará una búsqueda un poco más exhaustiva en el último experimento, seleccionando el valor medio de los *learning rate* escogidos.

Con los experimentos realizados se demuestra que un valor de learning rate constante dificulta y retrasa el aprendizaje de las redes neuronales. Ya que como se ha dicho

anteriormente llega un punto en el entrenamiento que este valor no permite la convergencia, es por ello la inclusión de las dos estrategias que incluyen un descenso del gradiente. Esto último mencionado se observará en los resultados de este experimento.

Después de realizar este experimento se decidió utilizar como estrategia de learning rate el crecimiento lineal junto con el descenso del coseno. Y como valor de *learning rate* se decidió utilizar los siguientes 3 valores debido a que cada modelo tiene un valor óptimo distinto (ver resultados para más información):

- 0,1.
- 0,05.
- 0,01.

Experimento con distintas funciones de pérdidas

El objetivo de este experimento fue comprobar qué función de pérdida o combinación de funciones de pérdida eran con las que se obtenían los mejores resultados. La hipótesis de partida es que si se usa cualquiera de las dos primeras funciones de pérdida puede ayudar a ordenar el espacio latente producido antes de la clasificación y esto podría facilitar la clasificación.

En este caso se ha utilizado como *Data Augmentation* la misma transformación que se aplica para la competición de Imagenet (descrita anteriormente).

También hay que destacar que para aplicar *Contrastive Loss* y *Triplet Loss* la imagen inicial pasa por dos y tres transformaciones respectivamente para poder luego aplicar la función de pérdida correspondiente. Esto conlleva a mayor uso de memoria y, por eso, los experimentos en los que sólo se usa *Crossentropy* se usó un *batch size* de 50 pero, en el resto, este tamaño se redujo a 16.

En cuanto a los experimentos que se realizaron fueron los siguientes:

- Usar solo la función de pérdida de *Crossentropy*.
- Utilizar *Crossentropy* junto con similitud (o *Contrastive Loss*).
- Utilizar *Crossentropy* junto con *Triplet loss*.

Por último, se ha visualizado el espacio latente en las distintas estrategias con la ayuda de una reducción de dimensionalidad (TSNE en este caso). La idea principal de esta visualización fue confirmar como ordenan el espacio latente las distintas funciones de pérdida y mejorar su comprensión de estas.

Como resumen de los resultados se ha decidido utilizar la función de pérdida de *Crossentropy* junto con *Contrastive Loss* debido a que es la que mejores resultados ha dado.

Experimento con mejores resultados

En este experimento se pretende utilizar las estrategias que mejores resultados de los experimentos anteriores, estas estrategias se encuentran tanto comentadas al final de cada apartado en la siguiente sección como definidas a continuación en formato resumen:

- Modelos que se utilizarán:
  - *EfficientNetB4* con un *batch size* de 24.
  - *EfficientNetB5* con un *batch size* de 16.
- Para ambos modelos se utilizarán redes pre-entrenadas y no se congelará ninguna capa en el proceso de entrenamiento.
- En relación con la estrategia de *Data Augmentation* se utilizará la transformación de Imagenet combinada con *Autoaugment*.
- La estrategia de entrenamiento del *learning rate* será lineal (creciente) hasta el *epoch* 5 y luego se utilizará un descenso del coseno.
- En cuanto a los valores elegidos de *learning rate* se utilizarán los siguientes:
  - 0,1.
  - 0,05.
  - 0,01.
- En cuanto a la función de pérdida elegida será la combinación de *Crossentropy* junto con *Contrastive Loss*.

4. Resultados

En esta sección, se describen los resultados de las diferentes aproximaciones que se han realizado para intentar conseguir una estrategia óptima y entender que variables son las que influyen para poder obtener mejores resultados. Hay que recalcar que al menos que se mencione lo contrario los resultados mostrados serán con el conjunto de datos de validación.

Experimento 1. Análisis de arquitecturas

A continuación, se procede a mostrar los resultados obtenidos en el primer experimento. Donde la idea principal era ver cuál podría ser la mejor arquitectura/modelo a utilizar en el experimento final. En primer lugar, se visualizará un gráfico de barras con los resultados obtenidos en *accuracy* en el dataset de validación (véase Ilustración 4.1).

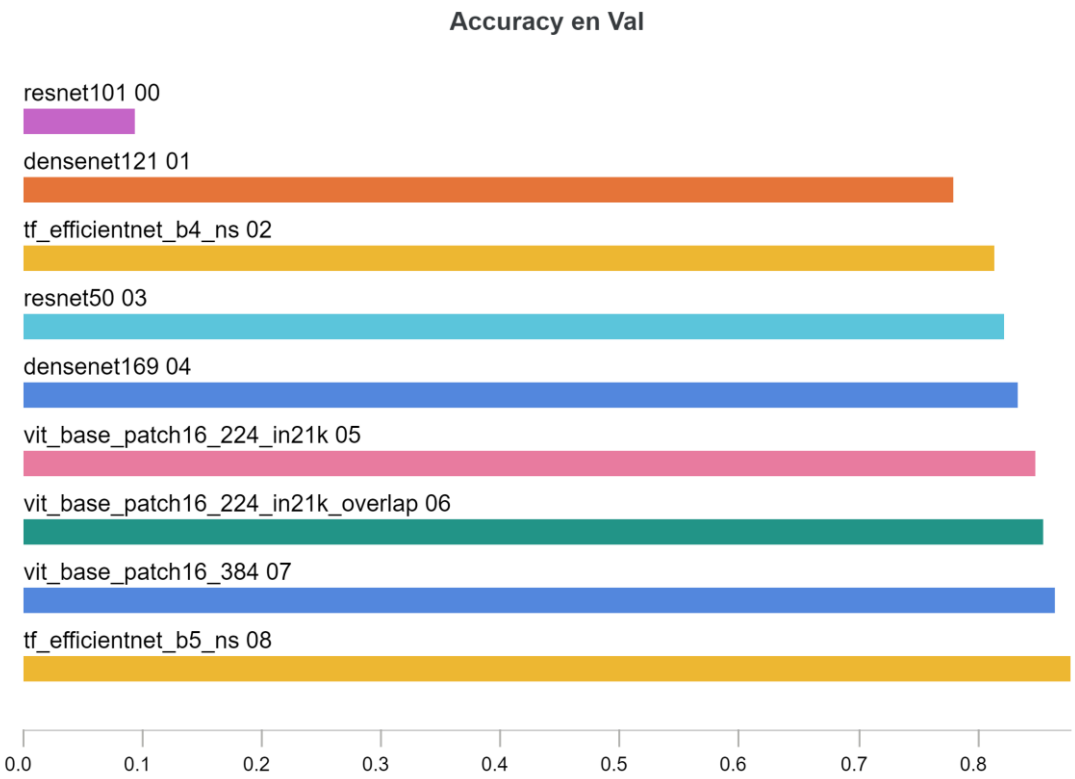

Ilustración 4.1 accuracy en el dataset de validación, cada barra representa a un modelo y el id de la barra. Fuente: elaboración propia.

En los gráficos siguientes se ha filtrado el modelo de *Resnet101* debido a que su resultado es muy dispar respecto al resto de los modelos (un 9% de precisión solo).

Para analizar el posible *overfitting* que se puede haber producido durante el entrenamiento se ha optado por realizar una gráfica temporal donde el eje "x" es el epoch y el eje "y" es la diferencia del *accuracy* en la fase de entrenamiento respecto a la de validación, la gráfica se muestra en la Ilustración 4.2. En esta gráfica se aprecia que todos los modelos tienen bastante *overfitting* ya que hay más de un 10% de diferencia entre ambos *accuracy* destacando el *overfitting* de *DenseNet*, probablemente debido a todas

las *skip-connections* que tiene su arquitectura. Esto último provoca que el modelo memorice las particularidades de los datos de training.

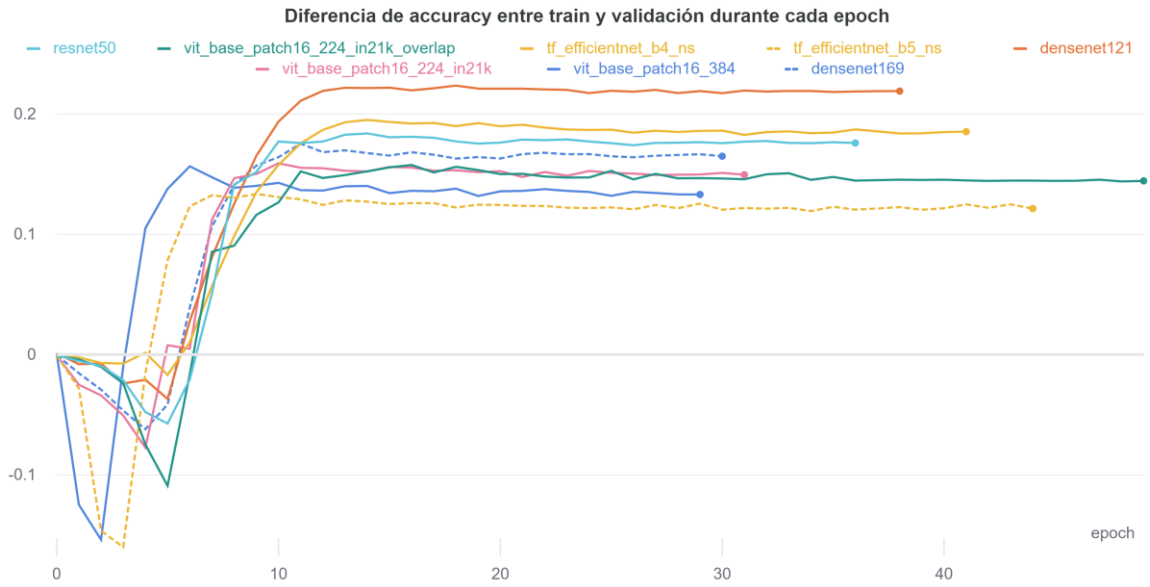

Ilustración 4.2 Diferencia de accuracy entre entrenamiento y validación durante cada epoch. Fuente: elaboración propia.

En la Ilustración 4.3 se muestra un gráfico de coordenadas paralelas (el color de la línea en el gráfico representa el valor del accuracy, amarillo es más alto y azul más bajo) donde se puede observar que los modelos con un *batch size* de 16 obtienen mejores resultados. Esto no es debido al *batch size* como uno podría pensar inicialmente sino a que son modelos de mayor tamaño (con mayor número de parámetros) los que tienden a obtener mejores resultados. Además, para ver los resultados en forma más detallada se dispone de la Tabla 4.1.

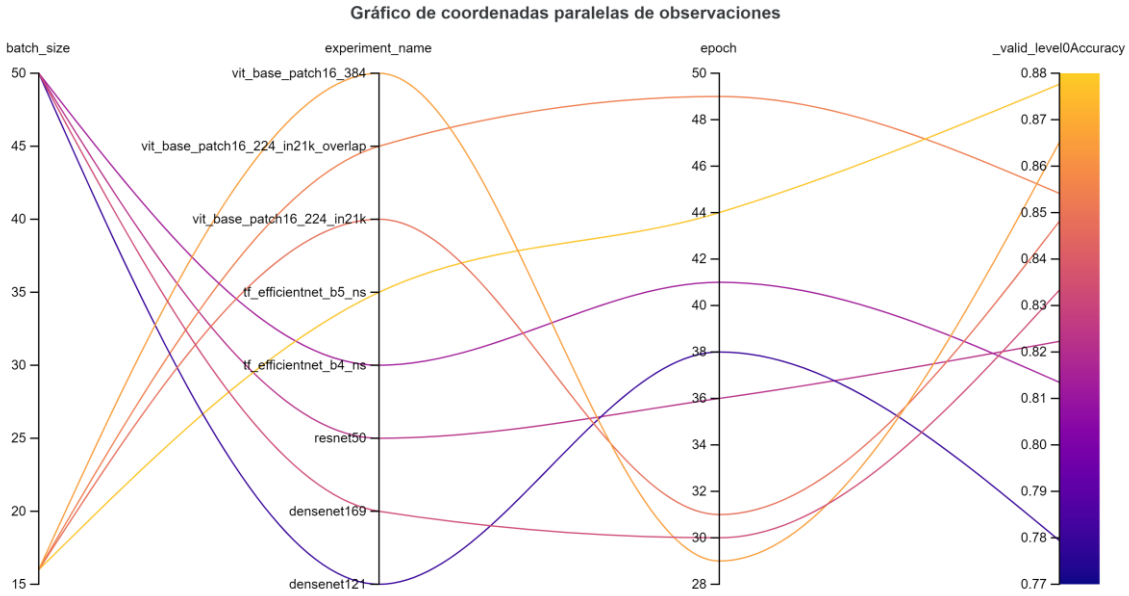

Ilustración 4.3 Gráfico de coordenadas paralelas de observaciones del experimento 1. Fuente: elaboración propia.

Tabla 4.1 Resultados del experimento 1. Fuente: elaboración propia.

Batch size	Modelo	Epoch	Accuracy (%)
16	tf_efficientnet_b5_ns	44	88
16	vit_base_patch16_384	29	87
16	vit_base_patch16_224_in21k_overlap	49	85
16	vit_base_patch16_224_in21k	31	85
50	densenet169	30	83
50	resnet50	36	82
50	tf_efficientnet_b4_ns	41	81
50	densenet121	38	78

Con todos estos resultados se optó por utilizar en el último experimento la arquitectura de *EfficientNet* con ambos modelos: el modelo b5 debido a que es el que mejor resultados ha dado (un 87,76% de *accuracy*) y el modelo b4 porque hemos realizado todos los experimentos con este modelo y se sabe cuáles son los parámetros para obtener los mejores resultados (debido a que se han estudiado en los distintos experimentos).

Experimento 2. Redes pre-entrenadas vs entrenadas

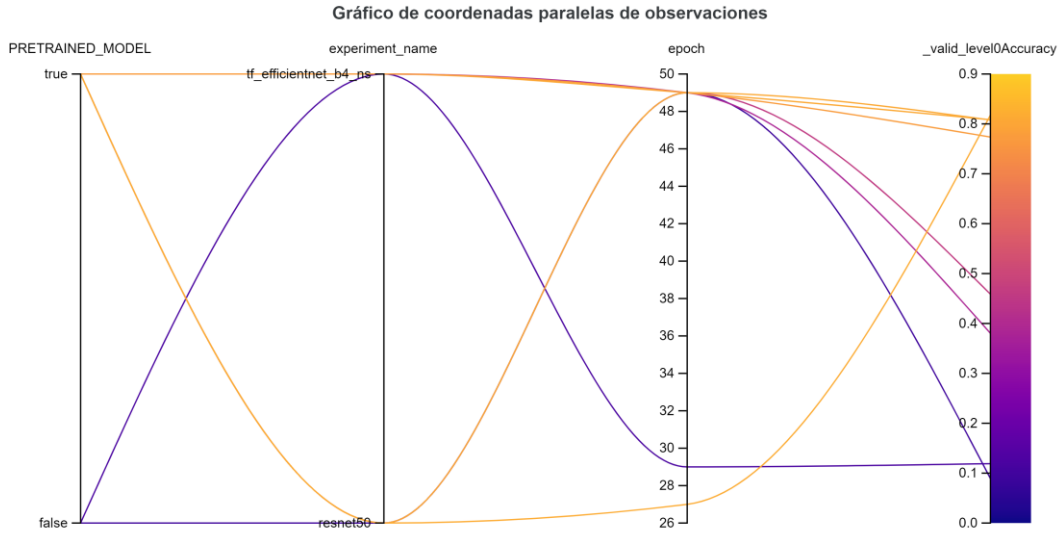

Ilustración 4.4 Gráfico de coordenadas paralelas de observaciones experimento 2, comparando modelos pre-entrenados contra no pre-entrenados. Fuente: elaboración propia.

Como se puede observar en la Ilustración 4.4 los modelos pre-entrenados obtienen mejores resultados. También cabe destacar que la arquitectura de *EfficientNet*, aunque no se usará un modelo pre-entrenado aprendió rápidamente. El problema es que tuvo un alto nivel de *overfitting*, (véase la Ilustración 4.5), lo que se podría haber solucionado mediante *Data Augmentation*, lo cual no era parte de este experimento. Además, hay que destacar que todos los modelos pre-entrenados tienen un valor de *overfitting* similar el cual es aproximadamente de 0,2 mayor su accuracy en entrenamiento respecto al dataset de validación.

Una vez visto que las redes no pre-entrenadas no dan buenos resultados, se decide eliminar de las siguientes visualizaciones de este experimento para poder apreciar mejor los resultados.

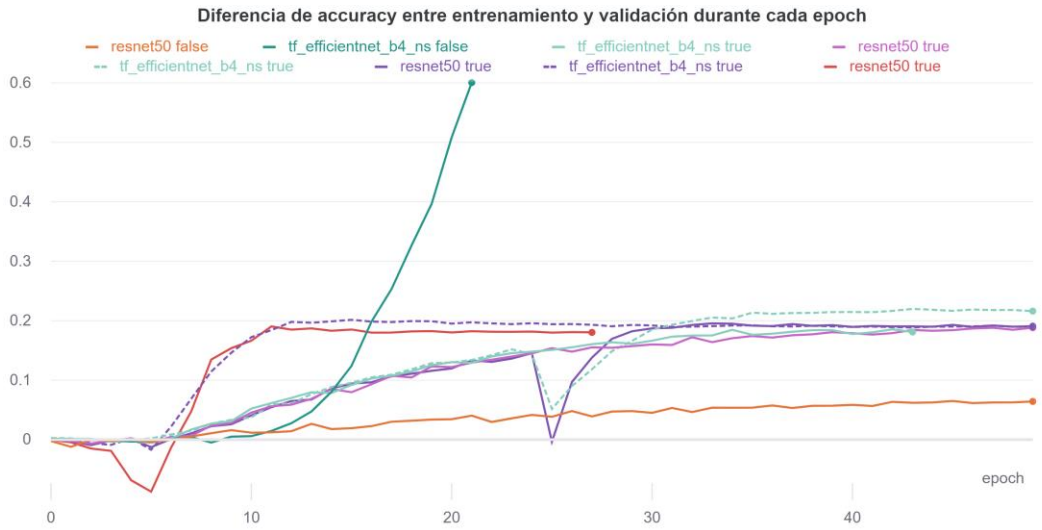

Ilustración 4.5 Diferencia de accuracy entre entrenamiento y validación durante cada epoch, el primer nombre es el modelo que se usó y si es false o true está relacionado con si se ha usado el modelo pre-entrenado o no. Fuente: elaboración propia.

En cuanto a congelar capas, en la Ilustración 4.6 se dispone de un gráfico de coordenadas paralelas, donde se observa principalmente que congelar todas las capas salvo la última no obtiene buenos resultados. Esto puede ser debido a que las capas intermedias no son capaces de adaptarse a los datos del problema. También se observa que se obtienen mejores resultados sin congelar las capas, pero no es una diferencia tan significativa como en el caso de modelos pre-entrenados vs no pre-entrenados. La Ilustración 4.7 muestra el *accuracy* final de las distintas estrategias de congelamiento de capas cada barra representa un modelo y la estrategia de congelamiento de capas seguida.

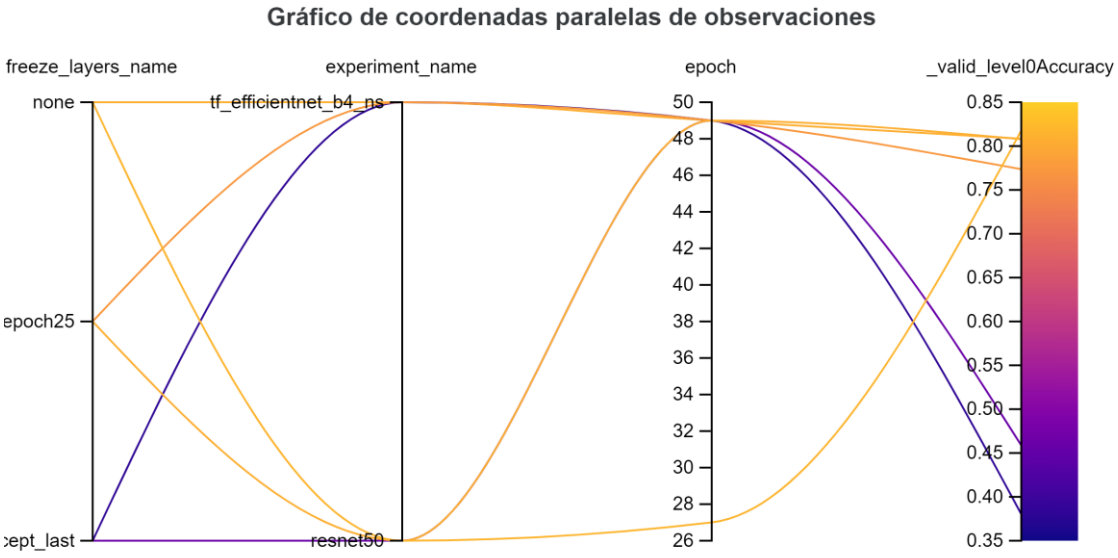

Ilustración 4.6 Gráfico de coordenadas paralelas de observaciones experimento 2, comparando las capas congeladas. En freeze layers, que no se visualiza correctamente se tienen las siguientes clases de arriba abajo, none, congelamiento de todas las capas excepto la última hasta el epoch 25 y congelar todas las capas excepto la última. Fuente: elaboración propia.

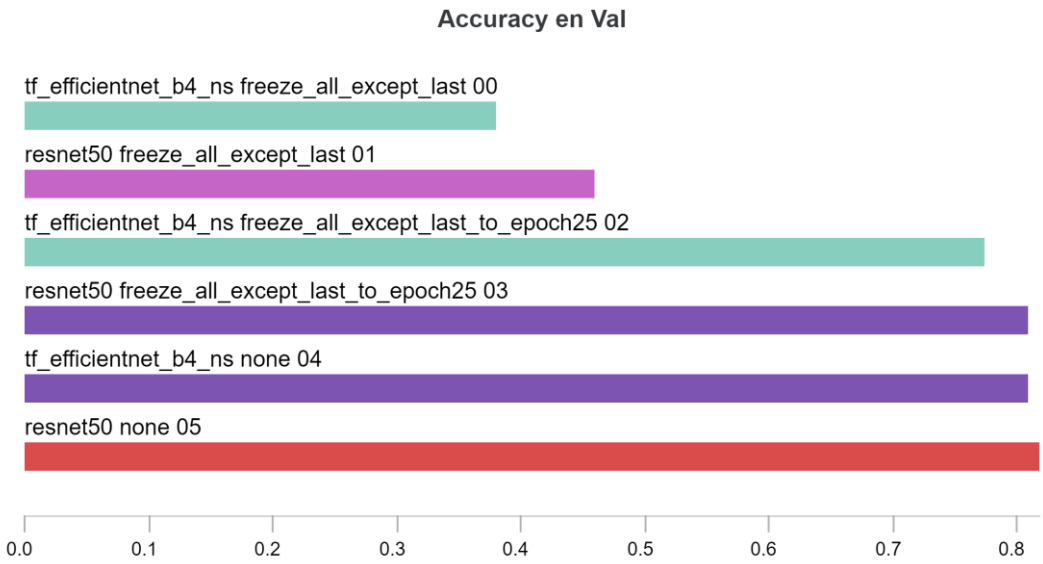

Ilustración 4.7 Gráfico de barras donde cada barra es el modelo, las capas que se congelaron y el id del experimento. Fuente: elaboración propia.

A continuación, se va a analizar si estas diversas estrategias suponen alguna mejora en tiempos de entrenamiento (ver Ilustración 4.8). Se puede observar que congelar todas las capas excepto la última, hasta el epoch 25, es demasiado restrictivo ya que las redes empiezan a mejorar notablemente cuando las redes donde no se han congelado capas ya han convergido y obtienen el mejor resultado.

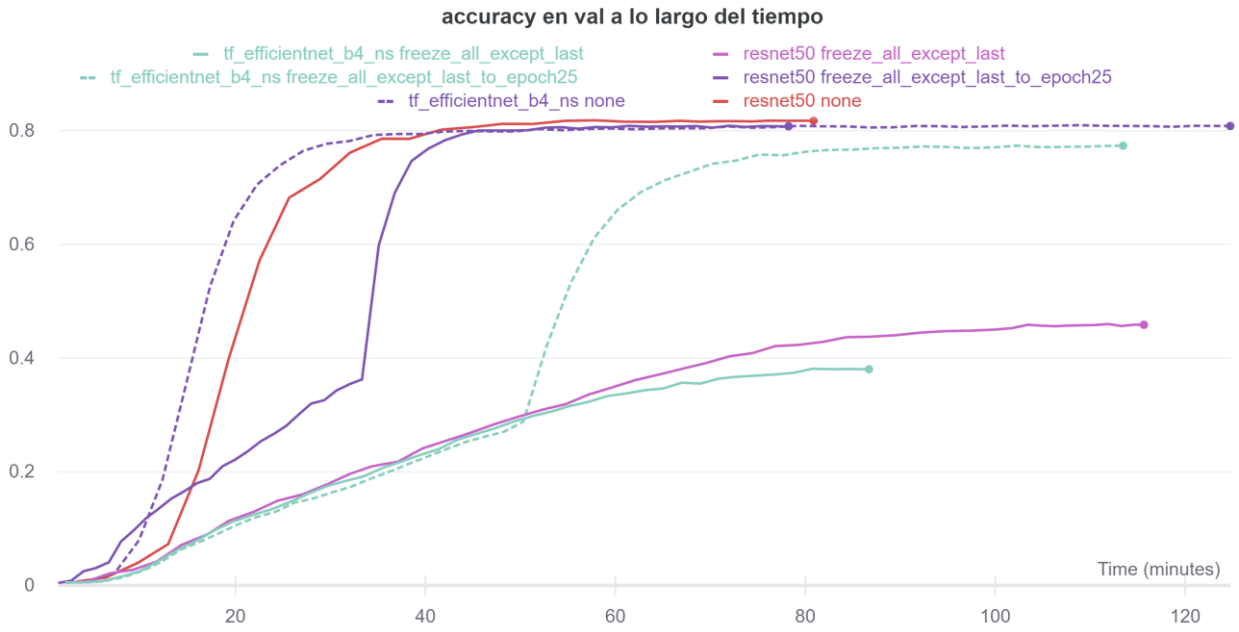

Ilustración 4.8 Gráfico de líneas donde se observa el accuracy a lo largo del tiempo, donde "x" es el tiempo en minutos del entrenamiento. Fuente: elaboración propia.

En base a los resultados obtenidos, en el experimento final se optará por no congelar capas y afinar el modelo con pesos pre-entrenados en el proceso de *transfer learning*.

Experimento 3. Pruebas con distintas estrategias de *Data Augmentation*

Tanto en la Ilustración 4.9 como en la Ilustración 4.10 (cada barra es el modelo junto con la estrategia de transformación seguida) se puede apreciar que las estrategias que contenían *Autoaugment* o la transformación que se utiliza en Imagenet tienen mejores resultados que las que no. Además, en relación con la combinación de *Autoaugment* y *Mixup*, no se observa una mejora significativa si lo comparamos con la aproximación que sólo usa *Autoaugment*. Esto se puede observar fácilmente en la Ilustración 4.10 donde la diferencia entre una estrategia y la otra es difícilmente observable.

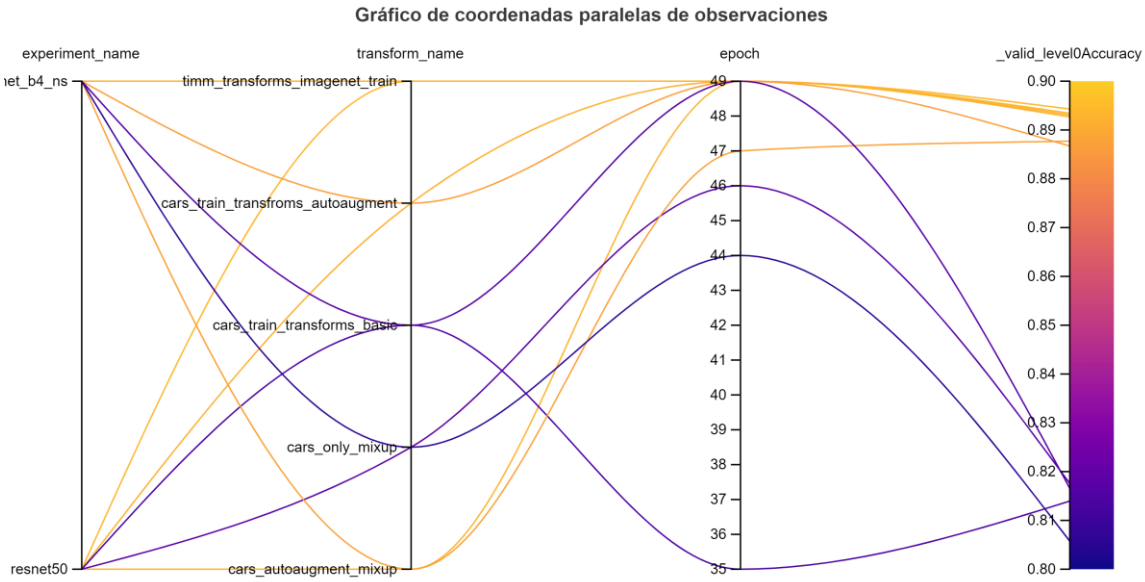

Ilustración 4.9 Gráfico de coordenadas paralelas de observaciones experimento 3, comparando modelos con distintos tipos de data augmentation, en la primera columna el valor superior es EfficientNet. Fuente: elaboración propia.

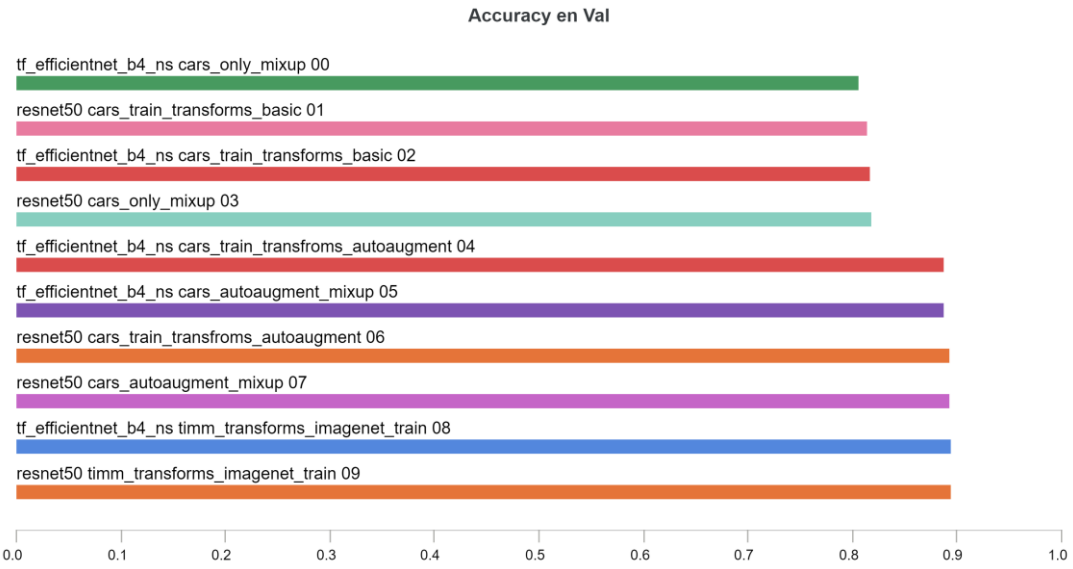

Ilustración 4.10 Gráfico de barras donde cada barra es el modelo, el data augmentation utilizado y el id del experimento. Fuente: elaboración propia.

Respecto al análisis del *overfitting* en cada aproximación analizada, se ha optado por la diferencia de *accuracy* entre entrenamiento y validación durante cada *epoch* (véase la Ilustración 4.11) observándose que en las estrategias donde se ha utilizado *Autoaugment* la diferencia de *accuracy* comparando entrenamiento y validación es de 0,1. También se observa que en las estrategias que no se aplica ni *Autoaugment* ni la transformación de Imagenet este valor ronda el 0,2.

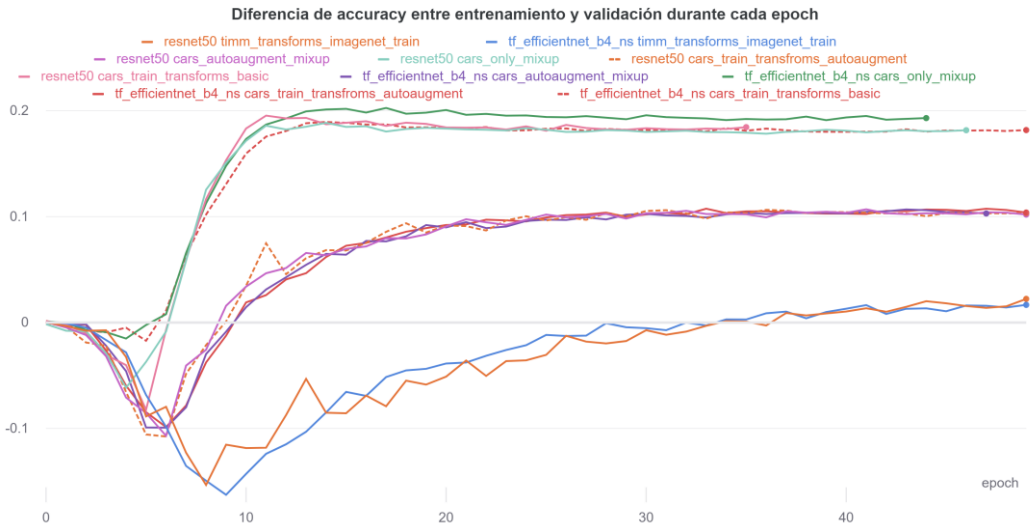

Ilustración 4.11 Diferencia de accuracy entre entrenamiento y validación durante cada epoch. Fuente: elaboración propia.

Por último, observando la utilización de sólo la transformación de Imagenet se observa que apenas hay *overfitting*, es decir, el valor del *accuracy* tanto en entrenamiento como en validación son prácticamente el mismo. Debido a esto, se ha realizado un experimento extra, donde se combina la estrategia de transformación de Imagenet con la usada en *Autoaugment* con el objetivo de analizar si esta combinación ofrecía alguna mejora.

Los resultados de este experimento extra se pueden visualizar en la Ilustración 4.12 para ver el estudio del *overfitting* y en la Ilustración 4.13 para visualizar el *accuracy* de las últimas estrategias estudiadas.

En el estudio del *overfitting* (Ilustración 4.12) se observa como la estrategia combinada funciona mejor, dando resultados que se mantienen por debajo incluso de 0, es decir que se obtienen mejores resultados en el dataset de validación que en el de entrenamiento. Y en la Ilustración 4.13 (cada barra es el modelo junto con la estrategia de transformación seguida) se puede observar que para la arquitectura *EfficientNet* se obtienen mejores resultados con la estrategia combinada. Por otro lado, resulta curioso que en el caso de la arquitectura *ResNet* no suceda lo mismo (ver sección 5, Discusión).

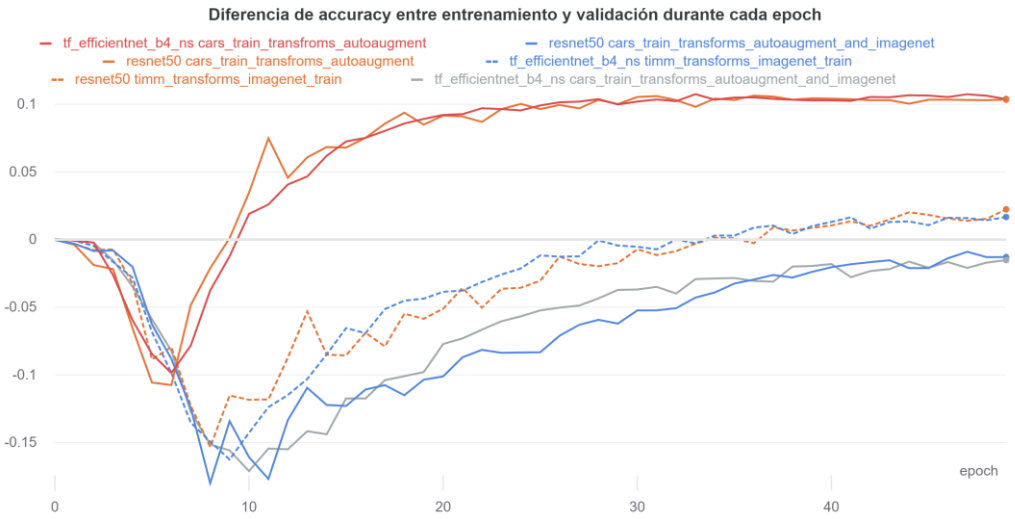

Ilustración 4.12 Diferencia de accuracy entre entrenamiento y validación durante cada epoch, mejores estrategias combinadas. Fuente: elaboración propia.

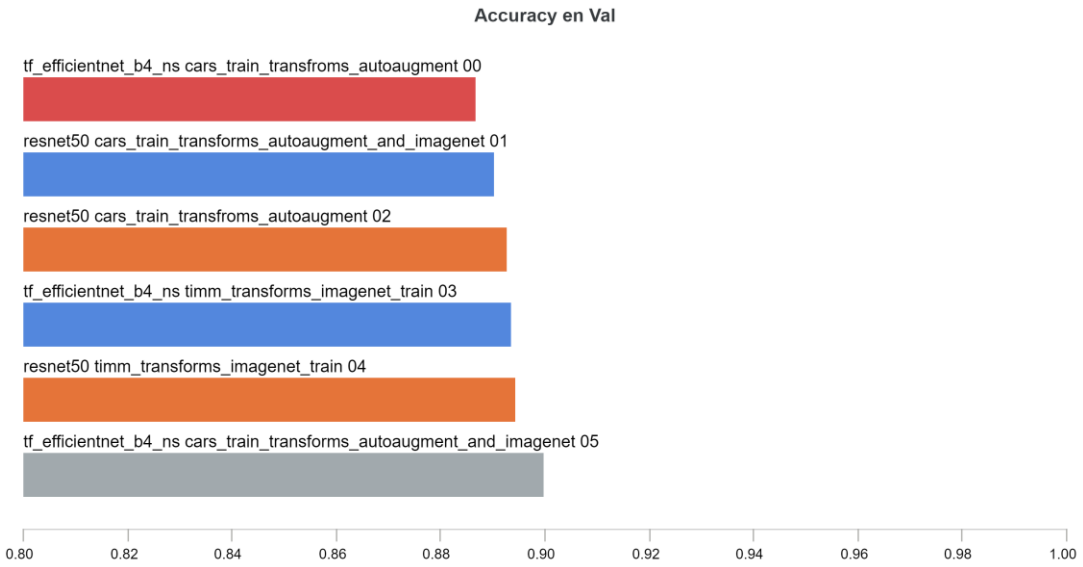

Ilustración 4.13 Gráfico de barras de las mejores estrategias de transformación combinadas donde cada barra es el modelo, el data augmentation utilizado y el id del experimento. Fuente: elaboración propia.

Finalmente, cabe destacar que los modelos que se están usando sin *Data Augmentation* no superan el accuracy del 83% (se visualiza fácilmente en Ilustración 4.10) y que usando *Data Augmentation* se acercan a un *accuracy* del 90% significando una mejoría de alrededor un 7% en *accuracy* y estando solo a un 6% del estado del arte.

Por tanto, para el experimento final se utilizará la estrategia combinada de aplicar la transformación de Imagenet junto con la de *Autoaugment*.

Experimento 4. Distintas estrategias y valores de learning rate

El objetivo de este experimento es analizar cuál es el mejor valor de *learning rate* y cuál es la mejor estrategia que se podría usar. En la Ilustración 4.14 se puede visualizar cómo evoluciona el *learning rate* con las distintas estrategias.

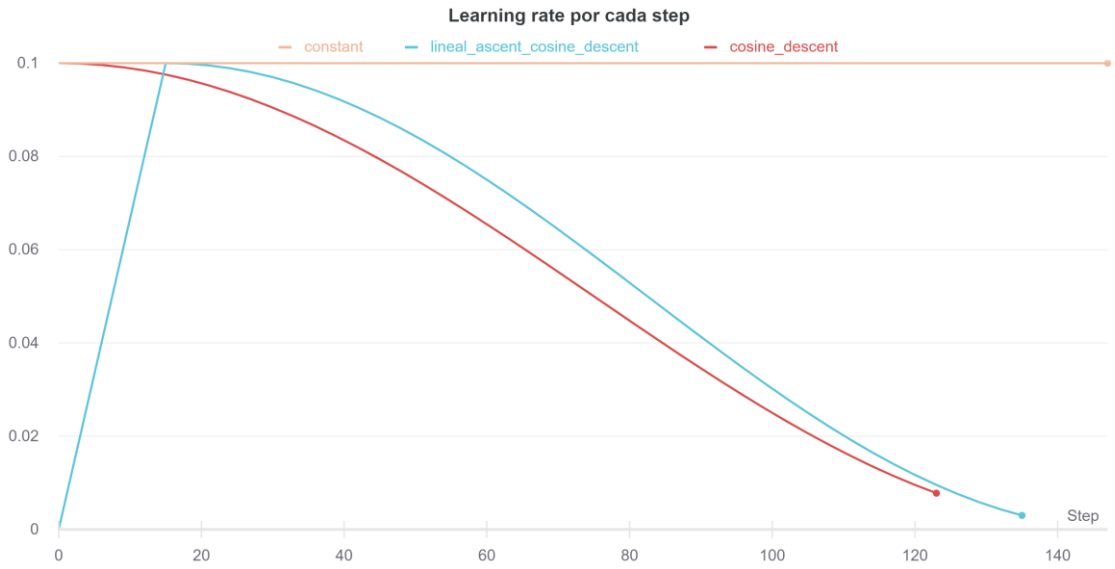

Ilustración 4.14 Visualización de las distintas estrategias de learning rate. Fuente: elaboración propia.

En cuanto a los resultados obtenidos (véase la Ilustración 4.15) se destaca que un *learning rate* de 0.001 (el más bajo que hemos seleccionado) afecta negativamente a los resultados. El motivo más probable por el que esto puede ocurrir es que la red alcance óptimos locales de los cuales no pueda salir debido a su bajo *learning rate*. También, si se observa la Ilustración 4.16, la velocidad a la que el *accuracy* mejora es muy bajo respecto a los otros valores.

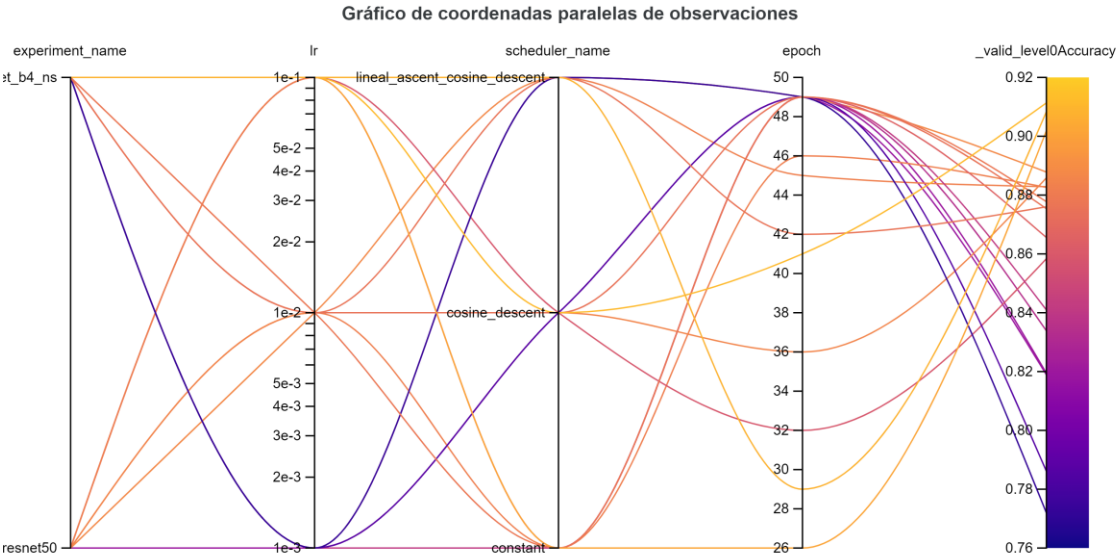

Ilustración 4.15 Gráfico de coordenadas paralelas de observaciones experimento 4, comparando modelos con distintos learning rate y distintas estrategias de learning rate. Por último, para aclarar la poca visibilidad en la primera columna el valor superior es EfficientNet. Fuente: elaboración propia.

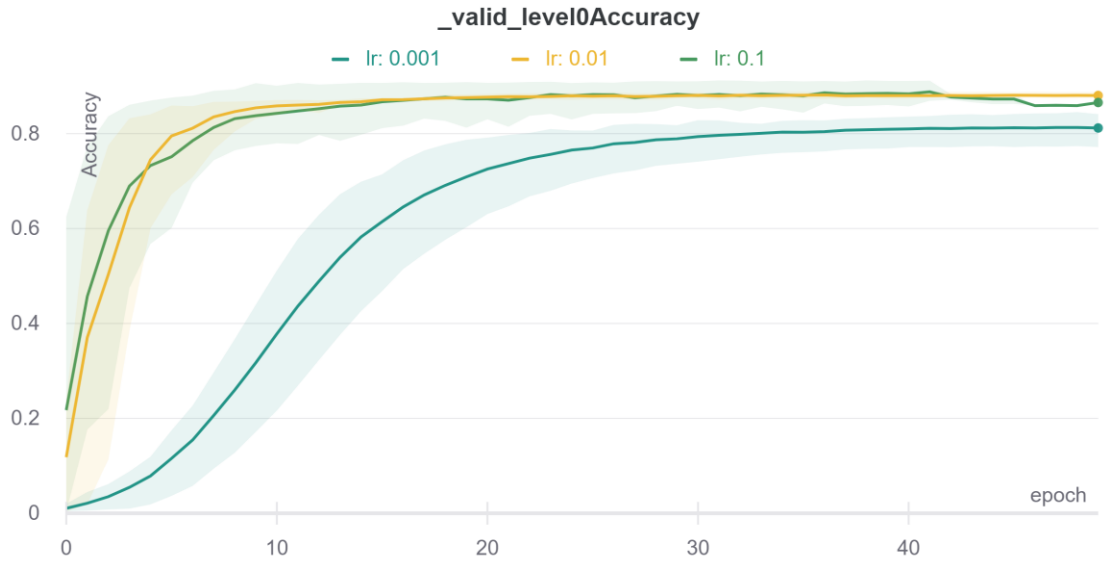

Ilustración 4.16 Gráfico con la evolución del accuracy (agrupado por los experimentos con el mismo learning rate) en el dataset de validación por cada epoch, la línea de color fuerte es la media de los experimentos con ese learning rate y la parte sombreada la desviación típica. Fuente: elaboración propia.

Para facilitar una visualización más sencilla se ha optado por eliminar los experimentos con valor de *learning rate* de 0.001 a partir de la Ilustración 4.17 (inclusive). En este gráfico de coordenadas paralelas se observa que el *learning rate* influye distinto para cada arquitectura/modelo utilizado. Apoyándonos en la Ilustración 4.18 y en la Ilustración 4.19 se observa como el modelo de *EfficientNet* que se utiliza consigue mejores valores con un *learning rate* de 0.1 y que para la *ResNet* se obtienen mejores resultados con un *learning rate* de 0.01. Por tanto, para el experimento final se debe tener en cuenta que un valor de *learning rate* dependerá del modelo y no solo del dataset utilizado.

Ilustración 4.17 Gráfico de coordenadas paralelas de observaciones experimento 4, comparando modelos con distintos learning rate y distintas estrategias de learning rate, la diferencia con el gráfico anterior es la eliminación de las pruebas realizadas con un learning rate de 0.001. Por último, para aclarar la poca visibilidad en la primera columna el valor superior es EfficientNet. Fuente: elaboración propia.

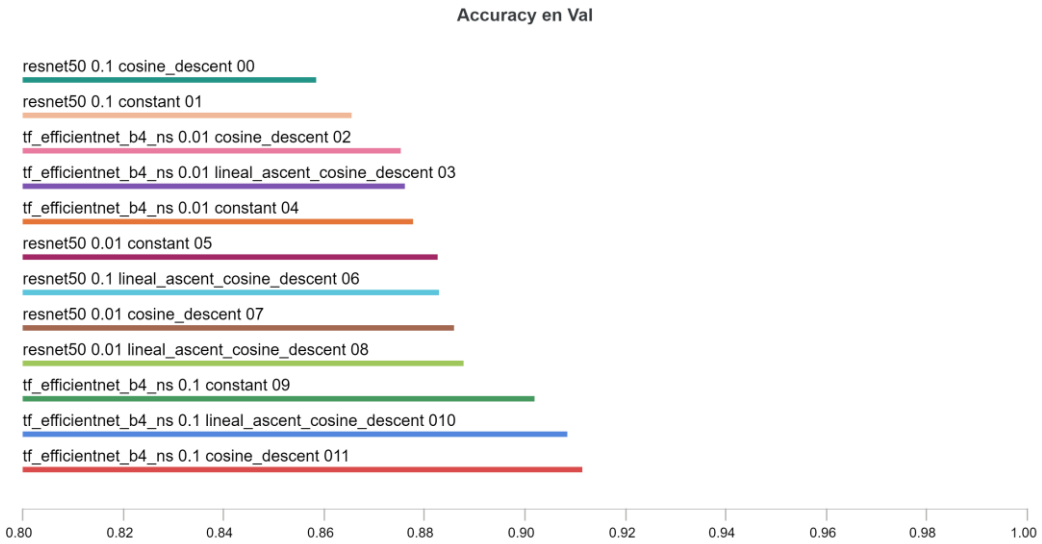

Ilustración 4.18 Gráfico de barras donde cada barra es el modelo, el learning rate utilizado, la estrategia de learning rate seguida y el número de barra. Fuente: elaboración propia.

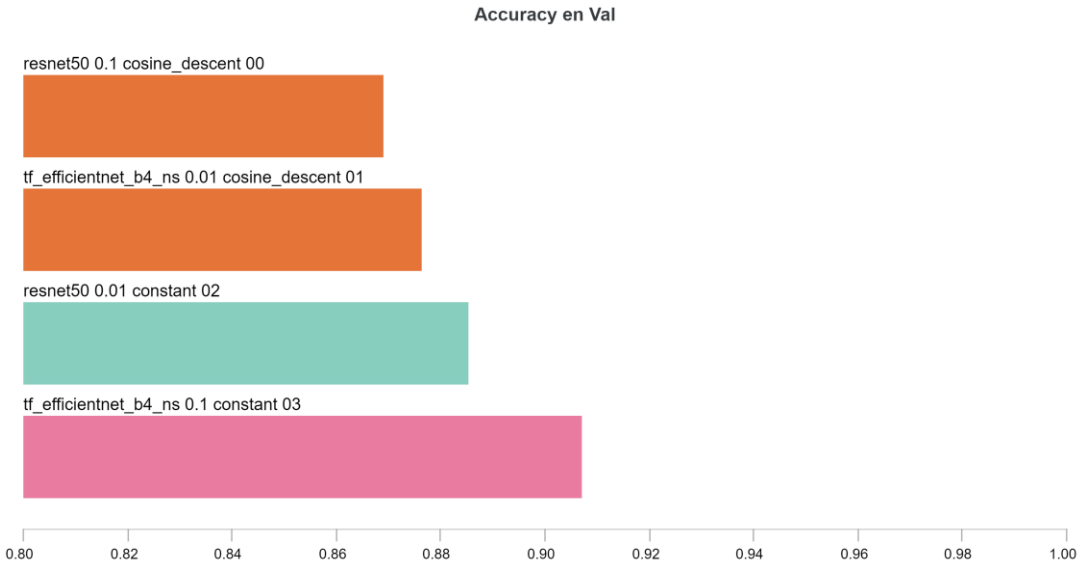

Ilustración 4.19 Gráfico de barras donde cada barra es la media de accuracy agrupadas por modelo, el learning rate utilizado y el número de barra. Fuente: elaboración propia.

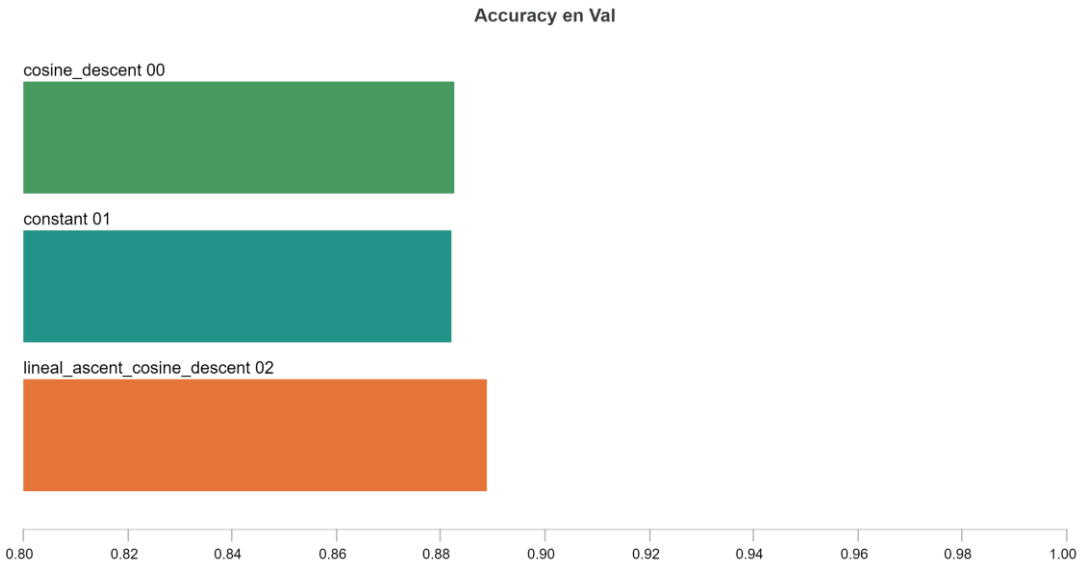

Ilustración 4.20 Gráfico de barras donde cada barra es la media de accuracy de la estrategia de learning rate utilizada y el número de barra. Fuente: elaboración propia.

En cuanto a la discusión de qué estrategia de *learning rate* es mejor, en la Ilustración 4.20 se dispone de una gráfica de barras con el *accuracy* en validación y con el eje de las abscisas entre 0,8 y 1 para poder visualizar correctamente la diferencia. Cómo se puede ver la estrategia de lineal_ascent_cosine_descent ligeramente mejor, pero con diferencias poco significativas respecto del resto de aproximaciones.

Para concluir estos resultados se ha decidido utilizar la estrategia de un crecimiento lineal creciente hasta el *epoch* 5 y luego un descenso del coseno debido a que es la estrategia cuya media da mejores resultados, como se observa en la Ilustración 4.20.

En cuanto al valor del *learning rate*, se ha decidido usar distintos valores para el experimento final debido a que se ha observado que el valor óptimo depende del modelo que se va a utilizar. Anteriormente se decidió que se iba a utilizar *EfficientNet* (B4 y B5) y ya que B4 se ha demostrado que funciona muy bien con un *learning rate* de 0,1 este será uno de los valores a utilizar. Además, también se utilizará el valor de 0,01 ya que ha proporcionado buenos resultados (en *ResNet50* en concreto). Pero debido a que no se sabe cuál es el óptimo de *EfficientNetB5*, también se usará el valor intermedio de 0,05 (de esta manera es más posible encontrar o saber cuál es el *learning rate* óptimo para estos modelos). Por último, tener en cuenta que no se escogen valores de *learning rate* superiores a 0,1 debido a que valores muy alto puede producir infinitos y que no permita converger al modelo.

Experimento 5. Funciones de pérdida

Se observa que salvo la combinación del modelo *ResNet* junto con la función de pérdida de *Crossentropy* y *Triplet Loss*, en el resto de los casos, utilizar una combinación de estás funciones o la *Crossentropy* con *Contrastive Loss* mejora los resultados. La mejora que se consigue al utilizar la combinación de funciones de pérdida representa aproximadamente 2 puntos porcentuales respecto del *accuracy* (ver la Ilustración 4.21 y la Ilustración 4.22).

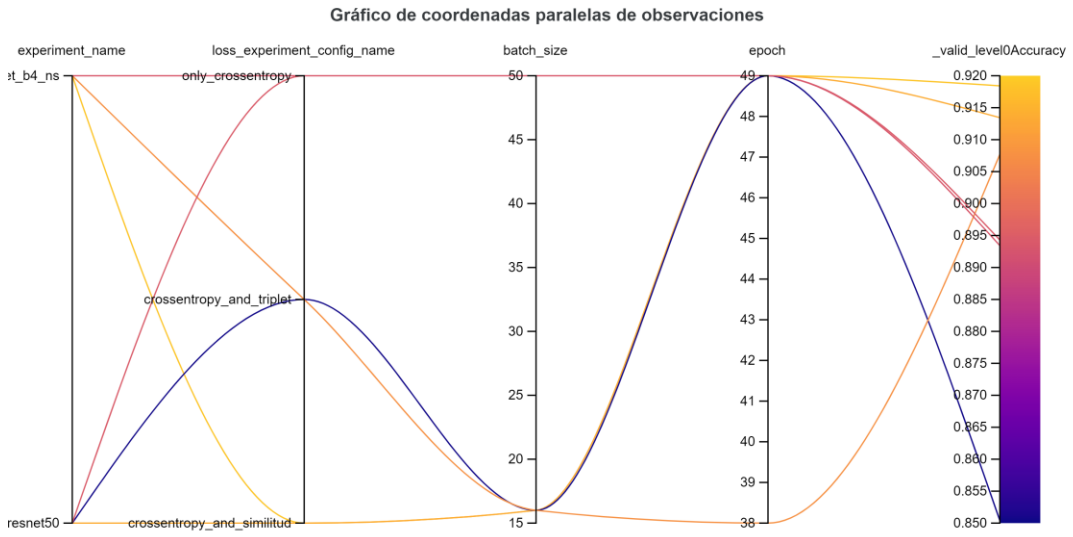

Ilustración 4.21 Gráfico de coordenadas paralelas de observaciones experimento 3, comparando modelos con distintas estrategias en la función de pérdida, en la primera columna el valor superior es EfficientNet. Fuente: elaboración propia.

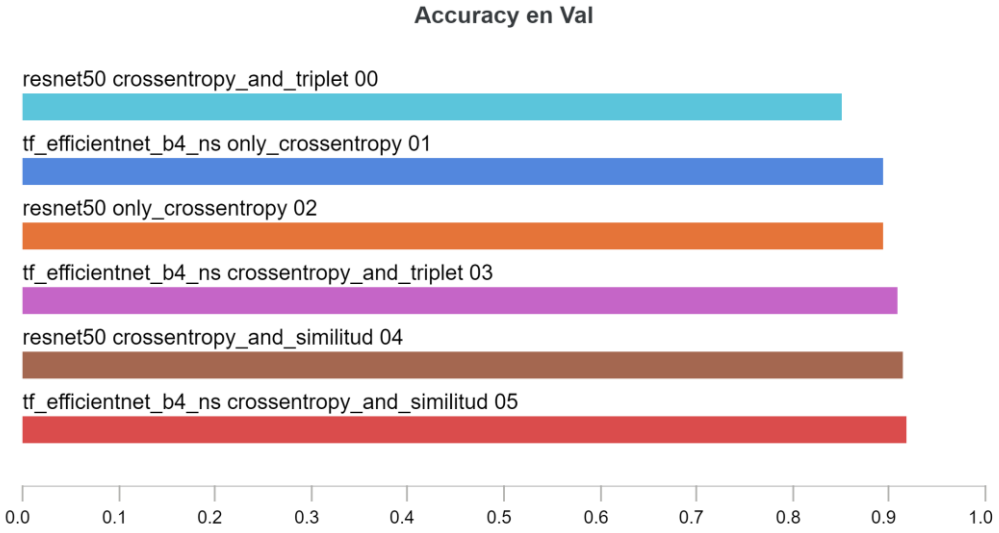

Ilustración 4.22 Gráfico de barras donde cada barra es el modelo, el data augmentation utilizado y el id de la barra. Fuente: elaboración propia.

En relación al análisis de *overfitting* que se ha realizado en este experimento, es curioso ver (como se muestra en la Ilustración 4.23) como se ha reducido notoriamente debido también a que en este experimento las redes no han logrado un 100% de *accuracy* en la fase de entrenamiento. Esto es debido, a la utilización del *Data Augmentation* usado (el de Imagenet) y debido a esto los resultados obtenidos les cuesta tener un alto *overfitting*.

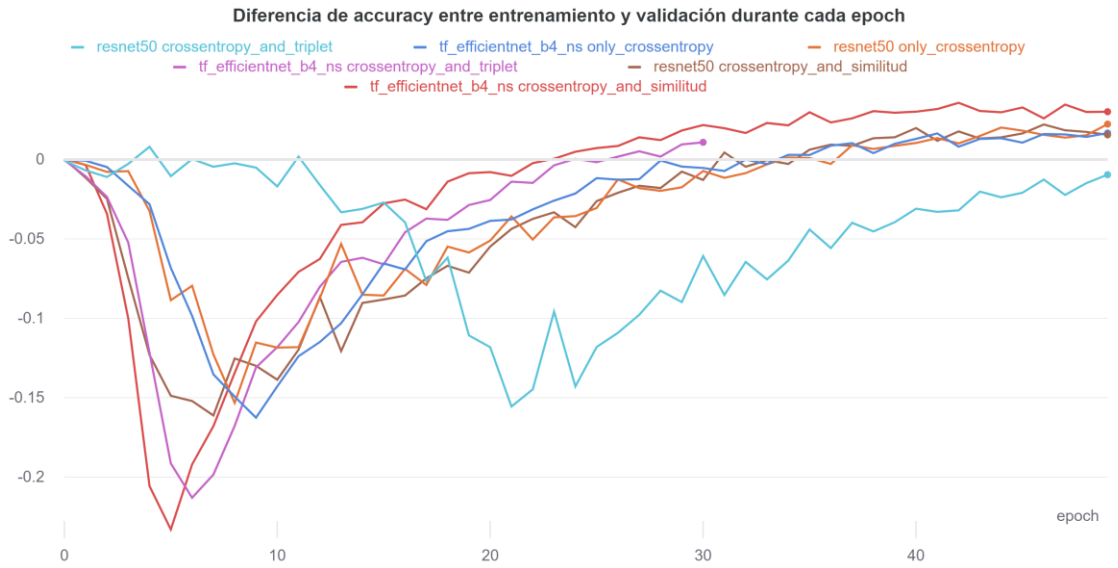

Ilustración 4.23 Diferencia de accuracy entre entrenamiento y validación durante cada epoch. Fuente: elaboración propia.

En relación con el espacio latente, se han creado la Ilustración 4.24 y la Ilustración 4.25. En la primera de ellas se puede observar cómo este espacio está desordenado, este es el estado inicial en el que se encuentra el espacio latente.

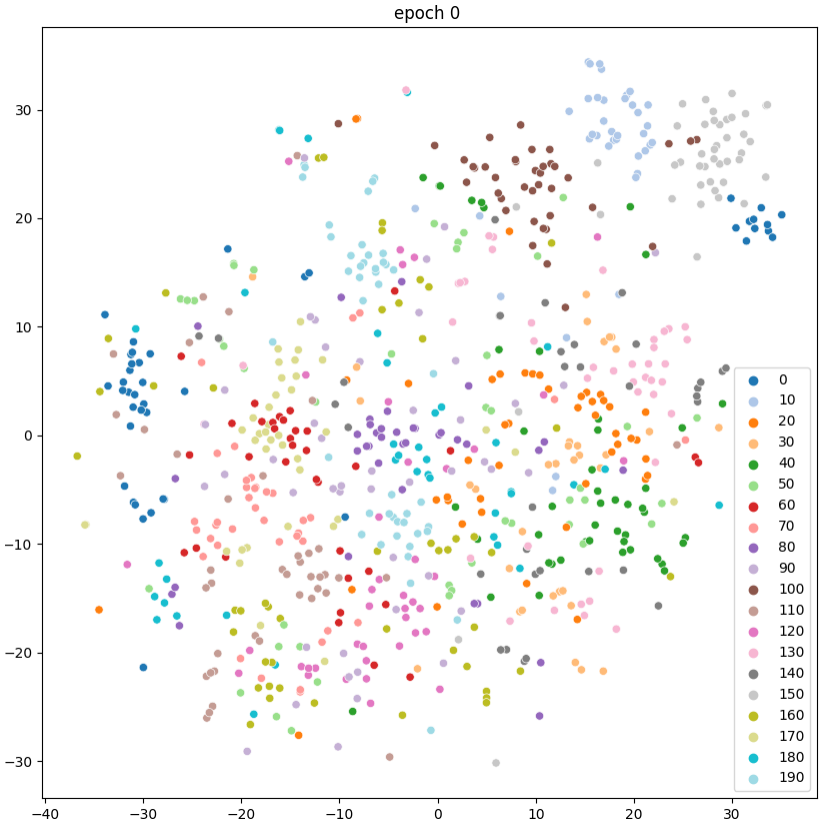

Ilustración 4.24 espacio latente antes del epoch 1 en el modelo EfficientNetB4 *con la función de pérdida Crossentropy. Fuente: elaboración propia.*

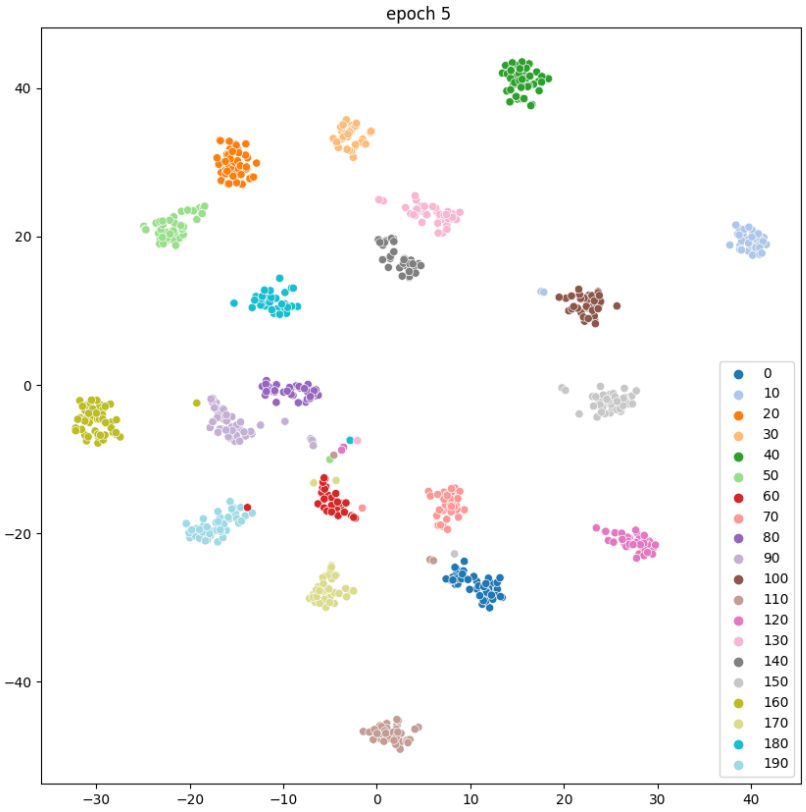

Ilustración 4.25 espacio latente después de 5 epoch en el modelo EfficientNetB4 con la función de pérdida Crossentropy. Fuente: elaboración propia.

Por otro lado, en la segunda gráfica se observa que este espacio latente empieza a ordenarse e incluso se podría hacer unas muy buenas predicciones con algún algoritmo de clusterización. Esto es debido a que la red neuronal ha aprendido a extraer las características importantes de cada clase. Y las clases similares tienden a tener un vector similar a la salida de las capas convoluciones. Para poder visualizar este vector de n dimensiones se procede a utilizar algún algoritmo de reducción de la dimensionalidad como PCA o T-SNE de esta manera se observa como todas las clases tienen sus propios clústeres y a medida que la red se entrena más y alcanza mayor accuracy se podrá observar como el espacio latente queda más ordenado.

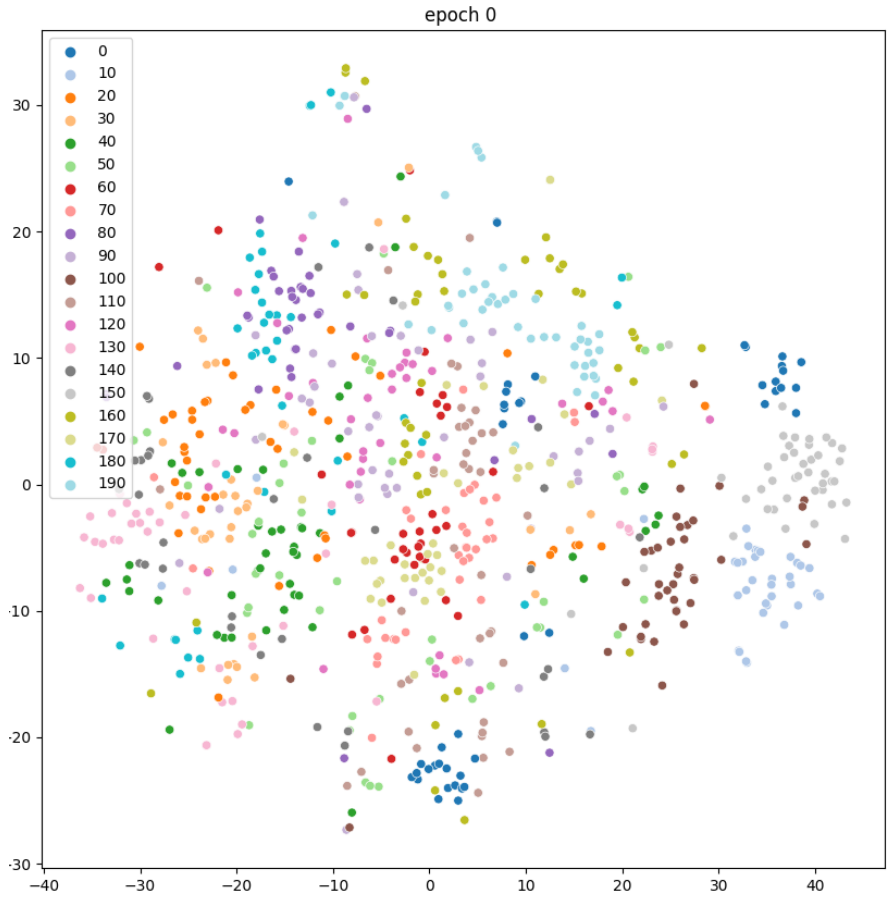

Ilustración 4.26 espacio latente antes del epoch 1 en el modelo EfficientNetB4 con la función de *pérdida* combinada de Crossentropy y Contrastive Loss. Fuente: elaboración propia.

Para poder comparar las distintas funciones de pérdidas (en este caso en concreto *Contrastive Loss* y *Crossentropy*) y visualizar correctamente sus espacios latentes se muestran la Ilustración 4.26, la cual es el espacio desordenado y la Ilustración 4.27 el cual es el espacio latente justo antes de empezar el *epoch* 5. Se puede observar que este espacio latente tiene los clústeres mucho más juntos que el anterior debido al efecto de *Contrastive Loss* ya que tiene como objetivo reducir la distancia en el espacio n dimensional entre las imágenes que se parecen entre ellas.

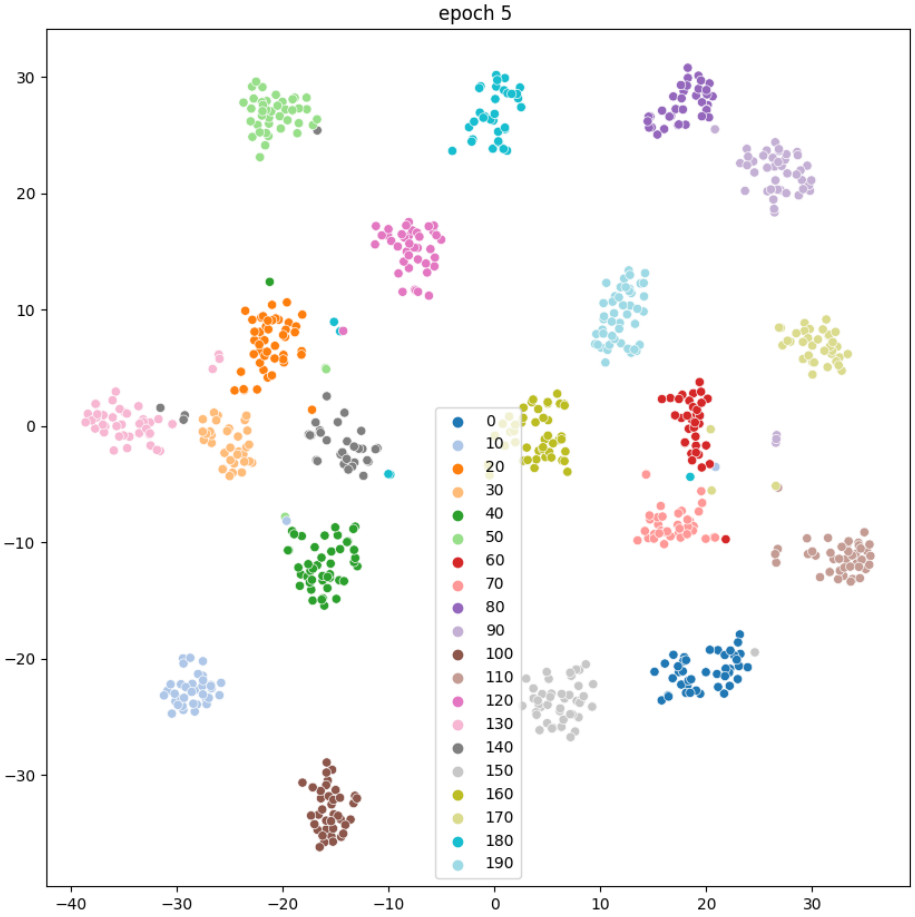

Ilustración 4.27 espacio latente antes del epoch 5 en el modelo EfficientNetB4 con la función de pérdida combinada de Crossentropy y Contrastive Loss. Fuente: elaboración propia.

En el caso de la combinación de la función *Crossentropy* con la *Triplet Loss* se ha obviado la ilustración en el *epoch* 0, debido a que es exactamente igual a la Ilustración 4.26 o Ilustración 4.24 (es un espacio latente desordenado ya que es antes de empezar el entrenamiento). En la Ilustración 4.28, antes del *epoch* 5, se visualiza un espacio más desordenado que en las anteriores aproximaciones para el mismo *epoch*, debido a que la convergencia de esta aproximación es más lenta. Por eso se ha seleccionado la Ilustración 4.29 la cual es antes del *epoch* 15 y se ha seleccionado debido a que tiene un *accuracy* similar al que tienen las otras aproximaciones en el *epoch* 5. Volviendo a la Ilustración 4.28 se puede observar el funcionamiento de la *Triplet Loss* ya que no se tienen los puntos perfectamente distribuidos en el espacio bidimensional. En este caso hay clústeres en el espacio que se alejan/repelen entre ellos dejando espacios bastante separados entre los mismos (Entre la parte superior y la parte de la derecha se observa este vacío). En cuanto a la Ilustración 4.29 lo que se puede apreciar es que tiene un comportamiento similar a la estrategia que se aplica *Contrastive Loss*.

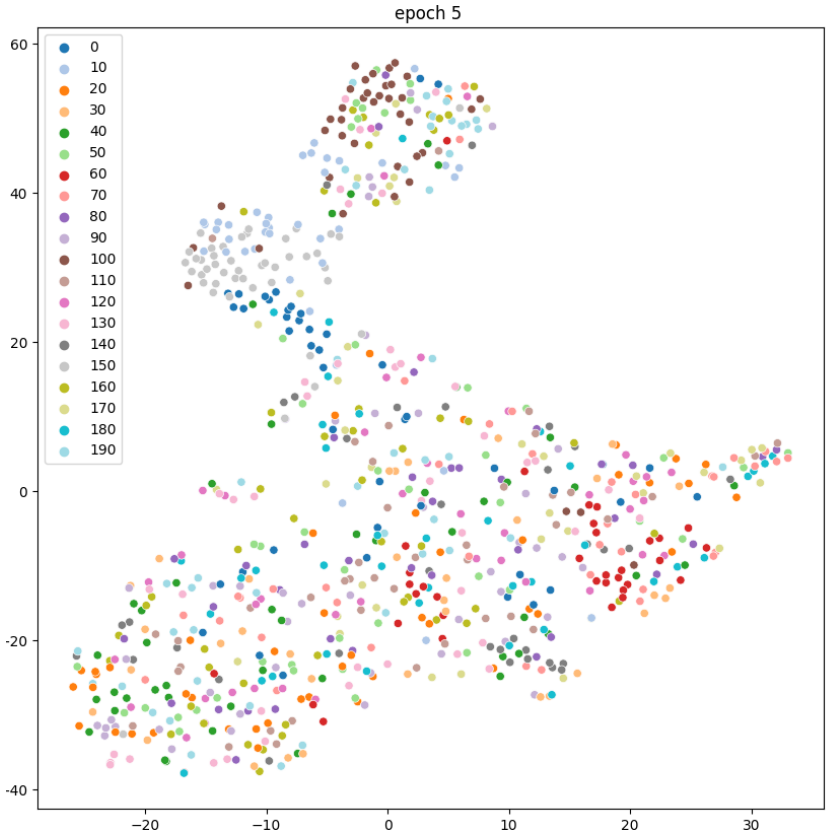

Ilustración 4.28 espacio latente antes del epoch 5 en el modelo EfficientNetB4 con la función de pérdida combinada de Crossentropy y Triplet Loss. Fuente: elaboración propia.

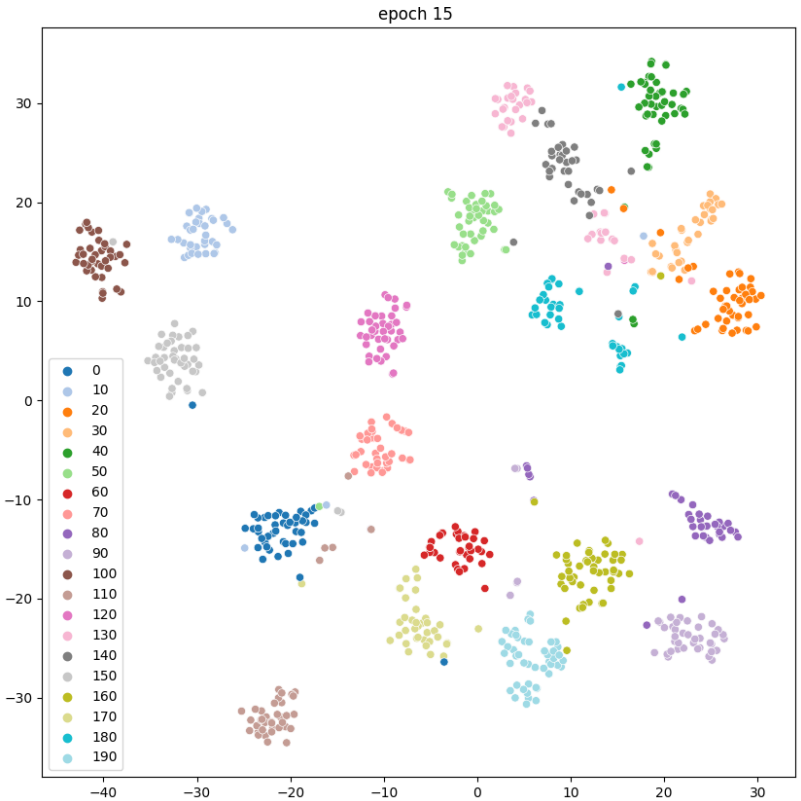

Ilustración 4.29 espacio latente antes del epoch 15 en el modelo EfficientNetB4 con la función de pérdida combinada de Crossentropy y Triplet Loss. Fuente: elaboración propia.

Por último, en este apartado, se ha decidido que en el experimento final se optará por utilizar la estrategia combinada de *Crossentropy* y *Contrastive Loss* debido a que es la que mejores resultados ha obtenido.

Experimento 6. Combinación de estrategias óptimas

En esta situación ya disponemos de todos los experimentos básicos y con estrategias individuales realizadas. Por lo que en este último experimento se mezclan las mejores estrategias obtenidas para cumplir con el objetivo de este trabajo, identificar la marca, el modelo y la fecha de lanzamiento de un vehículo con el mejor *accuracy* posible.

En la Ilustración 4.30 se observa que los resultados de *accuracy* están comprendidos entre el 92% y el 93,1% siendo los mejores resultados obtenidos de cualquier experimento realizado. También apoyándonos en la Ilustración 4.31, se observa que el mejor resultado (93,1%) es la combinación del modelo b5 de *EfficientNet* con un *learning rate* de 0.05. Además, también se aprecia que si se utiliza el *learning rate* correcto un modelo más pequeño puedo tener mejores resultados que un modelo mayor si en este no se dispone de un *learning rate* óptimo.

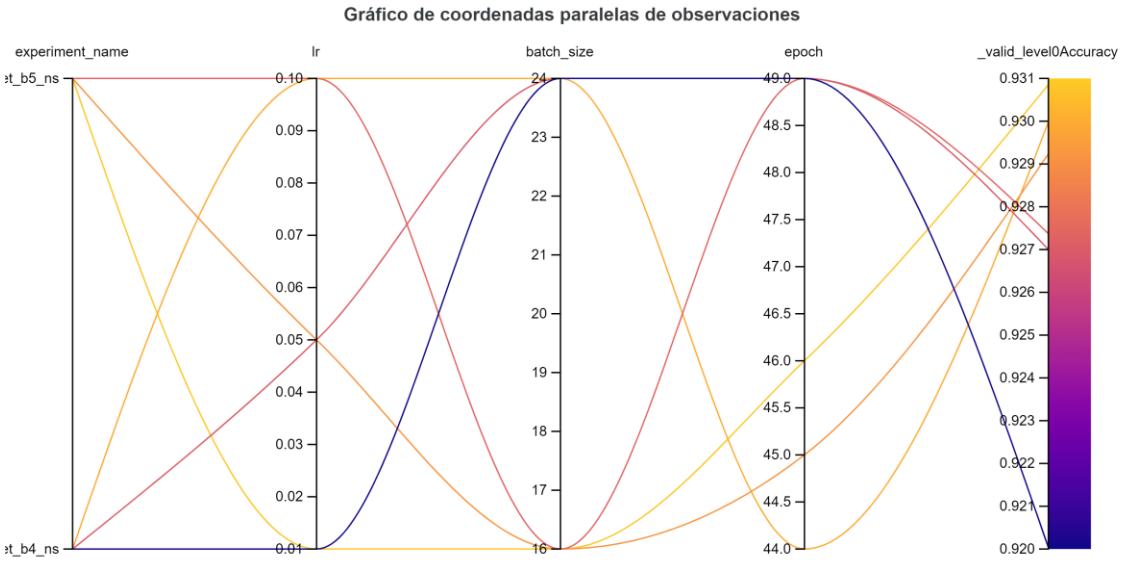

Ilustración 4.30 Gráfico de coordenadas paralelas de observaciones del experimento final comparando EfficientNet b5 en la primera columna y valor superior y EfficientNet b4 valor inferior. Fuente: elaboración propia.

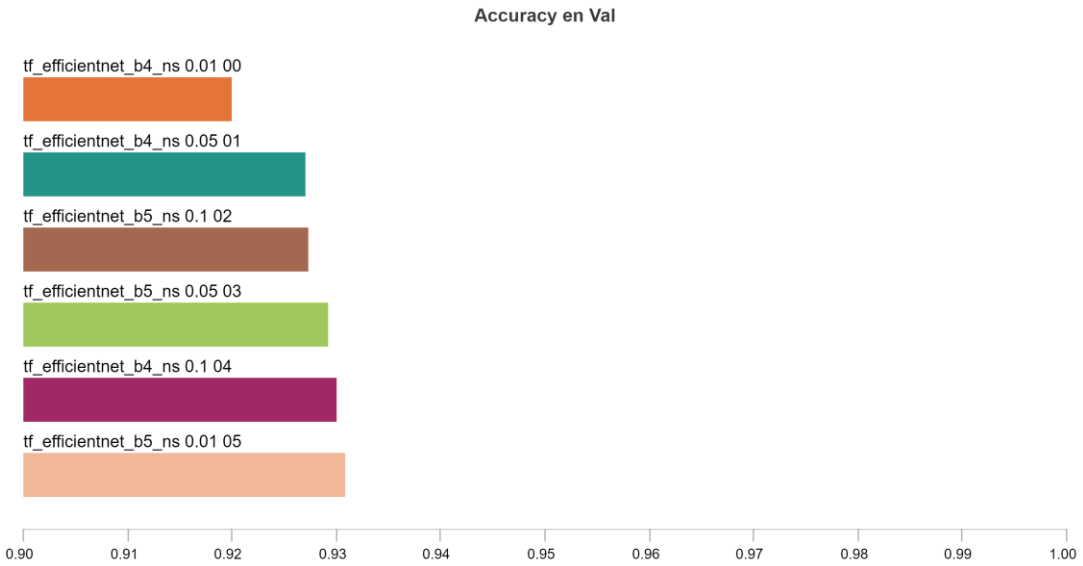

Ilustración 4.31 Gráfico de barras donde cada barra es un modelo empleado y el valor del learning rate utilizado. Fuente: elaboración propia.

En relación con el *overfitting* se puede visualizar en la Ilustración 4.32 que todas las instancias del experimento siguen el mismo patrón debido a que todas utilizan el mismo *Data Augmentation* y, en este caso, se observa que es un valor muy cercano a 0 al igual que sucedía en el experimento 3.

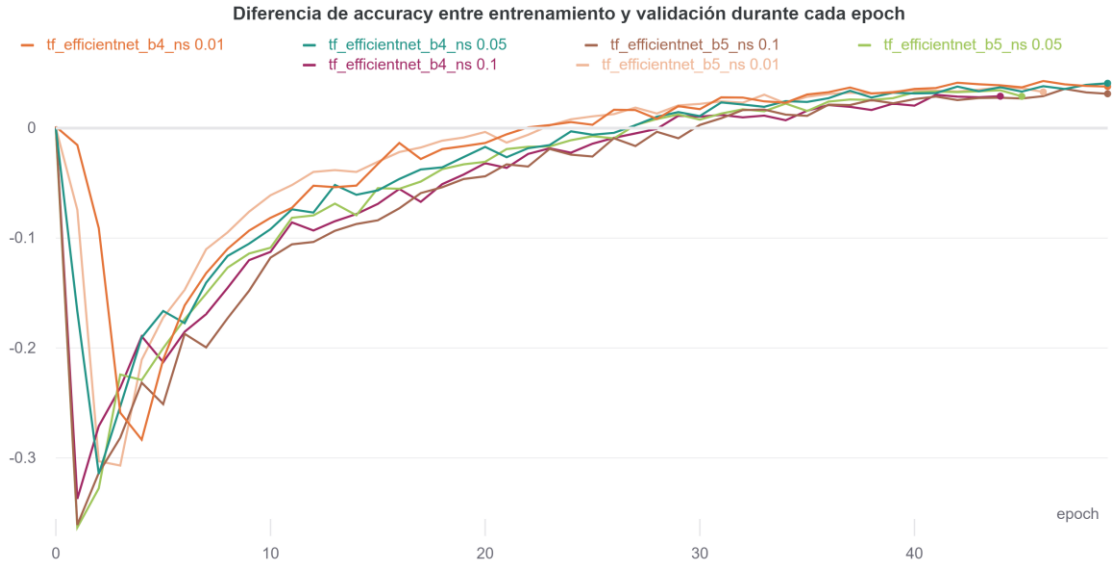

Ilustración 4.32 Diferencia de accuracy entre entrenamiento y validación durante cada epoch. Fuente: elaboración propia.

5. Discusión y limitaciones

El principal problema que se ha encontrado es la aparición de valores *NaN* producidos por valores que no convergían y tendían al infinito, los cuales provocaban error en el entrenamiento. Esto ocurría cuando el learning rate era muy alto para el modelo y dependía también de la función de pérdida que se usará. Además, una de las causas es utilizar un entrenamiento de 16 bits en lugar de 32 bits. Este entrenamiento suponía menor precisión y eso significa que es posible que algunos valores se dividieran por 0 en lugar de por un número muy pequeño y eso diera *NaN*. En [58], se enfrentan a un problema similar, pero con Transformers, planteando dos soluciones:

- La primera consiste en la relajación de la capa de atención del *Transformer*, por lo que solo se podría aplicar en *ViT*.
- La segunda consiste en un "sándwich" de capas de normalización, es decir, normalizar los datos en la entrada y en la salida de cada capa.

Otra limitación/problema ha sido que el número de *epochs* realizados siempre ha sido de 50. Esto ha sido debido a que hemos realizado un gran número de experimentos (más de 100 lanzados) con una duración entre 1 y 12 horas (*ViT*, por ejemplo). Es muy posible que si se hubiera aumentado el número de *epochs* en los modelos más grandes (incluido el *EfficientNet* b5) la estrategia de aumentar linealmente el *learning rate* unos pocos *epochs* y luego aplicar un descenso del mismo con forma del coseno, hubiera dado mejor *accuracy* en validación. Esto hubiera permitido más epochs en los distintos rangos de valores del *learning rate* por lo que conllevaría a mejor aprendizaje de la red neuronal.

En relación a los experimentos, se ha demostrado que las redes neuronales pre-entrenadas siempre funcionan mejor que una red entrenada desde 0. Además, se ha demostrado la importancia de aplicar estrategias de *Data Augmentatio*n las cuales producen mejores resultados al aumentar de forma artificial las imágenes de las que se dispone permitiendo a la red realizar mejores generalizaciones. También se ha demostrado que esa misma estrategia no tiene que ser efectiva para otros modelos, como se observa en la Ilustración 4.13 donde la *ResNet* no obtiene mejoras significativas con la mejor estrategia para la *EfficientNet*. Esto probablemente es debido a que la arquitectura de la red neuronal ya se encuentra saturada y no tenga posibilidad de obtener mejores resultados.

Por otro lado, con lo aprendido en el experimento relacionado con el *learning rate* se observa que el primer experimento, relacionado con los distintos modelos utilizados, no es concluyente. Esto es debido a que se ha descubierto que cada modelo tiene un *learning rate* óptimo y, aunque esto sería solucionable utilizando la estrategia del learning rate de ascenso lineal y descenso del coseno comentada anteriormente, se necesitaría un gran número de *epochs* para que todos los modelos pasarán un número de epochs suficiente con cada valor de *learning rate* (lo cual era computacionalmente prohibitivo para este Trabajo Fin de Máster).

En cuanto a los resultados, se ha pasado de un *accuracy* inicial (en el caso del *EfficientNet* b4) cercano al 80% a un accuracy del 93%, estando solo a 3 puntos porcentuales del estado del arte en este problema en concreto. Éste es un problema de

fine-grained y la mayoría de las arquitecturas/modelos están centrados en resolver este tipo de problema mediante aproximaciones que permitan distinguir los pequeños detalles de las imágenes. Pero en el caso de este trabajo hemos utilizado distintas técnicas para acercarnos a un resultado similar, sin enfocarnos en que es un problema donde las imágenes, en algunos casos, son difícilmente distinguibles entre clases.

6. Conclusiones y trabajos futuros

En el presente trabajo se ha intentado resolver un problema de clasificación de hilo fino (cuyo *accuracy* en el estado del arte es del 96%) mediante una aproximación general (llegando a obtener un accuracy del 93%). Esto se consigue mediante el estudio de distintas arquitecturas y modelos de redes neuronales. Asimismo, se han estudiado distintas estrategias que pueden afectar a la mejora de estas redes neuronales. También se ha analizado la importancia de utilizar redes pre-entrenadas para mejorar el *accuracy* y los tiempos de entrenamiento y la falta de datos. Otro de los estudios ha sido el análisis de distintas estrategias de *Data Augmentation* para la disminución del *overfitting* en la red neuronal y cuyo resultado ha sido la mezcla de utilizar las transformaciones aplicadas en Imagenet junto con Autoaugment. También se ha estudiado cómo puede llegar afectar el *learning rate* y se ha comprendido que el valor óptimo varía según el problema y el modelo que se esté usando. Además, se ha visto que la aplicación de combinación de funciones de pérdida mejora los resultados que si se utilizara solo la función de pérdida de *Crossentropy* que es la que se usa comúnmente en estos problemas.

En relación con los trabajos futuros posibles sería ideal replicar este estudio para diferentes datasets de hilo fino, los principales usados para investigación se pueden encontrar en [59]. Además, se podría hacer experimentos modificando el tamaño de las imágenes de entrada, ya que esto, puede afectar enormemente al *accuracy* a costa de más capacidad de computación.

Otro punto para tener en cuenta es que este problema se encuentra limitado al número de clases existentes siendo necesario reentrenar la red cada vez que se le añade una nueva clase, por lo que se propone como trabajo futuro un estudio de aprendizaje auto supervisado para resolver este problema. Es decir, una red neuronal la cual no necesite de imágenes etiquetadas para aprender, ejemplos de este tipo de arquitectura podría ser *SimCLR* [60], *MoCo* [61] o *Barlow Twins* [62]entre otros. Además, para realizar este trabajo se podría apoyar en uno de estos dos frameworks *Vissl* [63] o *Lightly* [64].

7. Referencias

[1] C. S. Vega, «Canal DotCSV,» 6 Diciembre 2020. [En línea]. Available: https://www.youtube.com/watch?v=Jlbxj182bhg&ab_channel=DotCSV.

[2] Wikipedia, «wikipedia,» [En línea]. Available: https://es.wikipedia.org/wiki/Ley_de_Moore#:~:text=vez%20m%C3%A1s%20baj o.-,%C2%BFQu%C3%A9%20es%20la%20ley%20de%20Moore%3F,una%20reducci%C3%B3n%20de%20costo%20conmensurable.. [Último acceso: 12 Diciembre 2020].

[3] «The wall street journal,» [En línea]. Available: https://www.wsj.com/articles/huangs-law-is-the-new-moores-law-and-explains-why-nvidia-wants-arm-11600488001.

[4] K. H. a. X. Z. a. S. R. a. J. Sun, «Deep Residual Learning for Image Recognition,» *arXiv,* 2015.

[5] S. M. a. R. K. a. M. S. a. S. Y. a. G. J. Soufi, «Deep-COVID: Predicting COVID-19 From Chest X-Ray Images Using Deep Transfer Learning,» *arXiv,* 2020.

[6] J. Elcacho, «lavanguardia,» [En línea]. Available: https://www.lavanguardia.com/economia/20210219/6246063/algoritmo-mascarilla-deteccion-ub-logmask-api-aigecko.html. [Último acceso: 17 Junio 2021].

[7] C.-H. H. a. H.-Y. W. a. Y.-L. Lin, «HarDNet-MSEG: A Simple Encoder-Decoder Polyp Segmentation Neural Network that Achieves over 0.9 Mean Dice and 86 FPS,» *arXiv,* 2021.

[8] Z. Net, «Youtube,» [En línea]. Available: https://www.youtube.com/watch?v=RLdvDfCrwj8. [Último acceso: 17 Junio 2021].

[9] H. Ford.

[10] istobal, «istobal,» [En línea]. Available: https://istobal.com/. [Último acceso: 24 06 2021].

[11] «seekpng,» [En línea]. Available: https://www.seekpng.com/ima/u2w7o0r5y3a9y3e6/. [Último acceso: 25 Junio 2021].

[12] C. S. Vega, «Canal DotCSV,» DotCSV, 2020. [En línea]. Available: https://www.youtube.com/watch?v=V8j10ENVz00&ab_channel=DotCSV. [Último acceso: 2020].

[13] M.-C. a. B. V. E. a. P.-P. L. a. M. N. Popescu, «Multilayer perceptron and neural networks,» *WSEAS Transactions on Circuits and Systems},* vol. 8, nº 7, pp. 579--588, 2009.

[14] Y. a. B. B. a. D. J. S. a. H. D. a. H. R. E. a. H. W. a. J. L. D. LeCun, «Backpropagation applied to handwritten zip code recognition,» *Neural computation,* vol. 1, nº 4, pp. 541--551, 1989.

[15] A. a. S. N. a. P. N. a. U. J. a. J. L. a. G. A. N. a. K. L. a. P. I. Vaswani, «Attention is all you need,» *arXiv ,* 2017.

[16] I. a. P.-A. J. a. M. M. a. X. B. a. W.-F. D. a. O. S. a. C. A. a. B. Y. Goodfellow, «Generative adversarial nets,» *Advances in neural information processing systems,* vol. 27, 2014.

[17] «Pytorch,» [En línea]. Available: https://pytorch.org/vision/stable/auto_examples/plot_transforms.html#sphx-glr-auto-examples-plot-transforms-py. [Último acceso: 18 Junio 2021].

[18] Lightly, «Github,» [En línea]. Available: https://github.com/lightly-ai/lightly. [Último acceso: 30 Junio 2021].

[19] «wikipedia,» [En línea]. Available: https://es.wikipedia.org/wiki/Extremos_de_una_funci%C3%B3n. [Último acceso: 30 Junio 2021].

[20] A. F. Gad, «paperspace,» [En línea]. Available: https://blog.paperspace.com/deep-learning-metrics-precision-recall-accuracy/#:~:text=Accuracy%20is%20a%20metric%20that,the%20total%20number%20of%20predictions.. [Último acceso: 30 Junio 2021].

[21] S. a. W. P. a. A. K. Dong, «A survey on deep learning and its applications,» *Computer Science Review,* vol. 40, p. 100379, 2021.

[22] «paperswithcode,» [En línea]. Available: https://paperswithcode.com/.

[23] J. a. D. S. a. G. R. a. F. A. Redmon, «You only look once: Unified, real-time object detection,» *IEEE,* pp. 779--788, 2016.

[24] W. a. A. D. a. E. D. a. S. C. a. R. S. a. F. C.-Y. a. B. A. C. Liu, «Ssd: Single shot multibox detector,» *Springer,* pp. 21--37, 2016.

[25] L. a. L. L. a. L. X. a. H. K. Zhang, «Is faster R-CNN doing well for pedestrian detection?,» *Springer,* pp. 443--457, 2016.

[26] «lawtomated,» [En línea]. Available: https://lawtomated.com/a-i-technical-machine-vs-deep-learning/. [Último acceso: 25 Junio 2021].

[27] J. D. H. S. J. K. S. S. S. M. Z. H. A. K. A. K. M. B. A. C. B. &. L. F.-F. Olga Russakovsky, «ImageNet Large Scale Visual Recognition Challenge».

[28] I. S. G. E. H. Alex Krizhevsky, «ImageNet Classification with Deep Convolutional Neural Networks,» 2012.

[29] Y. a. B. L. a. O. G. B. a. M. K.-R. a. o. LeCun, «Neural networks: Tricks of the trade,» *Springer Lecture Notes in Computer Sciences,* vol. 1524, nº 5-50, p. 6, 1998.

[30] N. Adaloglou, «theaisummer,» 23 Marzo 2020. [En línea]. Available: https://theaisummer.com/skip-connections/. [Último acceso: 6 Junio 2020].

[31] G. H. a. Z. L. a. L. v. d. M. a. K. Q. Weinberger, «Densely Connected Convolutional Networks,» *arXiv,* 2018.

[32] M. T. a. Q. V. Le, «EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks,» *arXiv,* 2020.

[33] A. a. B. L. a. K. A. a. W. D. a. Z. X. a. U. T. a. D. M. a. M. M. a. H. G. a. G. S. a. o. Dosovitskiy, «An image is worth 16x16 words: Transformers for image recognition at scale,» *arXiv,* 2020.

[34] Y. a. M. V. I. a. D. L. S. Wang, «Learning a discriminative filter bank within a cnn for fine-grained recognition,» *IEEE,* pp. 4148--4157, 2018.

[35] A.-X. a. Z. K.-X. a. W. L.-W. Li, «Zero-shot fine-grained classification by deep feature learning with semantics,» *Springer,* vol. 16, nº 5, pp. 563--574, 2019.

[36] «albumentations,» [En línea]. Available: https://albumentations.ai/. [Último acceso: 25 Mayo 2021].

[37] E. D. a. Z. B. a. M. D. a. V. V. a. L. Q. V. Cubuk, «Autoaugment: Learning augmentation policies from data,» *arXiv preprint arXiv:1805.09501,* 2018.

[38] S. a. H. D. a. O. S. J. a. C. S. a. C. J. a. Y. Y. Yun, «Cutmix: Regularization strategy to train strong classifiers with localizable features,» *Proceedings of the IEEE/CVF International Conference on Computer Vision,* pp. 6023--6032, 2019.

[39] H. a. C. M. a. D. Y. N. a. L.-P. D. Zhang, «mixup: Beyond empirical risk minimization,» *arXiv preprint arXiv:1710.09412,* 2017.

[40] P. a. T. P. a. W. C. a. S. A. a. T. Y. a. I. P. a. M. A. a. L. C. a. K. D. Khosla, «Supervised contrastive learning,» *arXiv preprint arXiv:2004.11362,* 2020.

[41] D. K. P. Florian Schroff, «FaceNet: {A} Unified Embedding for Face Recognition and Clustering,» 2015.

[42] D. K. Koidl. [En línea]. Available: https://www.scss.tcd.ie/~koidlk/cs4062/Loss-Functions.pdf. [Último acceso: 26 Mayo 2021].

[43] «hackernoon,» [En línea]. Available: https://hackernoon.com/latent-space-visualization-deep-learning-bits-2-bd09a46920df. [Último acceso: 31 Mayo 2021].

[44] X. a. H. K. Chen, «Exploring Simple Siamese Representation Learning,» *arXiv preprint arXiv:2011.10566,* 2020.

[45] A. a. B. A. R. a. Z. M. Z. a. B. D. a. M. F. Jaiswal, «A survey on contrastive self-supervised learning,» *Technologies,* vol. 9, nº 1, p. 2, 2021.

[46] «Wikipedia,» [En línea]. Available: https://commons.wikimedia.org/wiki/File:Triplet_loss.png. [Último acceso: 31 Mayo 2021].

[47] J. a. Z. Y. a. Y. Y. a. D. S. Fang, «Fine-grained vehicle model recognition using a coarse-to-fine convolutional neural network architecture,» *IEEE Transactions on Intelligent Transportation Systems,* vol. 18, nº 7, pp. 1782-1792, 2016.

[48] L. a. L. P. a. C. L. C. a. T. X. Yang, «A large-scale car dataset for fine-grained categorization and verification,» de *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 3973--3981.

[49] H. J. a. U. I. a. W. W. a. G. Y. a. F. Z. Lee, «Real-time vehicle make and model recognition with the residual SqueezeNet architecture,» *Sensors,* vol. 19, nº 5, p. 982, 2019.

[50] F. a. F. H. a. N. K. Tafazzoli, «A large and diverse dataset for improved vehicle make and model recognition,» *Tafazzoli, Faezeh and Frigui, Hichem and Nishiyama, Keishin,* pp. 1--8, 2017.

[51] F. N. a. H. S. a. M. M. W. a. A. K. a. D. W. J. a. K. K. Iandola, «SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and< 0.5 MB model size,» *arXiv ,* 2016.

[52] J. a. C. J. a. L. S. a. K. A. a. Y. C. a. B. Y. a. W. C. a. Y. A. He, «TransFG: A Transformer Architecture for Fine-grained Recognition,» *arXiv ,* 2021.

[53] H. a. N. H. Hanselmann, «Fine-Grained Visual Classification with Efficient End-to-end Localization,» *arXiv,* 2020.

[54] P. a. W. Y. a. Q. Y. Zhuang, «Learning attentive pairwise interaction for fine-grained classification,» *Proceedings of the AAAI Conference on Artificial Intelligence,* vol. 34, nº 07, pp. 13130--13137, 2020.

[55] J. K. a. M. S. a. J. D. a. L. Fei-Fei, «3D Object Representations for Fine-Grained Categorization,» *4th International IEEE Workshop on 3D Representation and Recognition (3dRR-13,* 2013.

[56] R. Wightman, «PyTorch Image Models,» *GitHub,* 2019.

[57] J. H. a. J.-N. C. a. S. L. a. A. K. a. C. Y. a. Y. B. a. C. W. a. A. Yuille, «TransFG: A Transformer Architecture for Fine-grained Recognition,» *arXiv,* 2021.

[58] M. a. Y. Z. a. H. W. a. Z. W. a. Z. C. a. Y. D. a. L. J. a. Z. X. a. S. Z. a. Y. H. a. o. Ding, «CogView: Mastering Text-to-Image Generation via Transformers,» *arXiv preprint arXiv:2105.13290,* 2021.

[59] «PaperWithCode,» [En línea]. Available: https://paperswithcode.com/task/fine-grained-image-classification. [Último acceso: 17 Junio 2021].

[60] T. C. a. S. K. a. M. N. a. G. Hinton, «A Simple Framework for Contrastive Learning of Visual Representations,» *arXiv,* 2020.

[61] K. H. a. H. F. a. Y. W. a. S. X. a. R. Girshick, «Momentum Contrast for Unsupervised Visual Representation Learning,» *arXiv,* 2020.

[62] B. T. S.-S. L. v. R. Reduction, «Jure Zbontar and Li Jing and Ishan Misra and Yann LeCun and Stéphane Deny,» *arXiv,* 2021.

[63] P. G. a. Q. D. a. J. R. a. M. L. a. M. X. and, «VISSL,» 2021. [En línea]. Available: https://github.com/facebookresearch/vissl. [Último acceso: 17 Junio 2021].

[64] I. Susmelj, «Lightly,» [En línea]. Available: https://github.com/lightly-ai/lightly.

[65] M. a. S. L. Buzzelli, «Revisiting the CompCars Dataset for Hierarchical Car Classification: New Annotations, Experiments, and Results,» *sensor,* vol. 21, nº 2, p. 596, 2021.

[66] i. d. s. y. Q. V. L. c. p. G. A. L. Mingxing Tan, «googleblog,» 29 Mayo 2019. [En línea]. Available: https://ai.googleblog.com/2019/05/efficientnet-improving-accuracy-and.html. [Último acceso: 19 Junio 2021].

[67] «googleblog,» [En línea]. Available: https://ai.googleblog.com/2020/12/transformers-for-image-recognition-at.html. [Último acceso: 26 Junio 2021].

[68] R. chandaliya, «Tele Stroke System for Stroke Detection,» 2020.

[69] P. A. C.-Y. F. W. L. J. K. A. C. B. Patrick Poirson1, «Fast Single Shot Detection and Pose Estimation,» p. 2016.

[70] C. a. N. X. a. C. F. a. A. E. Aytekin, «Clustering and unsupervised anomaly detection with l 2 normalized deep auto-encoder representations,» *2018 International Joint Conference on Neural Networks (IJCNN),* pp. 1-6, 2018.

[71] G. a. V. T. a. H. R. a. S. G. a. Y. Y. a. G. D. Loukas, «Cloud-based cyber-physical intrusion detection for vehicles using deep learning,» *Ieee,* vol. 6, pp. 3491--3508, 2017.

8. Anexos

Descripción jerárquica del dataset

En la Ilustración 8.1 se puede observar un esquema de la jerarquía del dataset que se utiliza donde la capa inferior solo puede estar unida a un elemento de la capa superior y la capa superior puede tener varios elementos de la capa inferior (aclarar que en este trabajo no se utilizará la información del tipo de vehículo) .Para conseguir este objetivo se realizaron distintos experimentos mediante los cuales se han analizado distintas modificaciones que se le pueden aplicar al entrenamiento de la red neuronal para que obtenga mejores resultados.

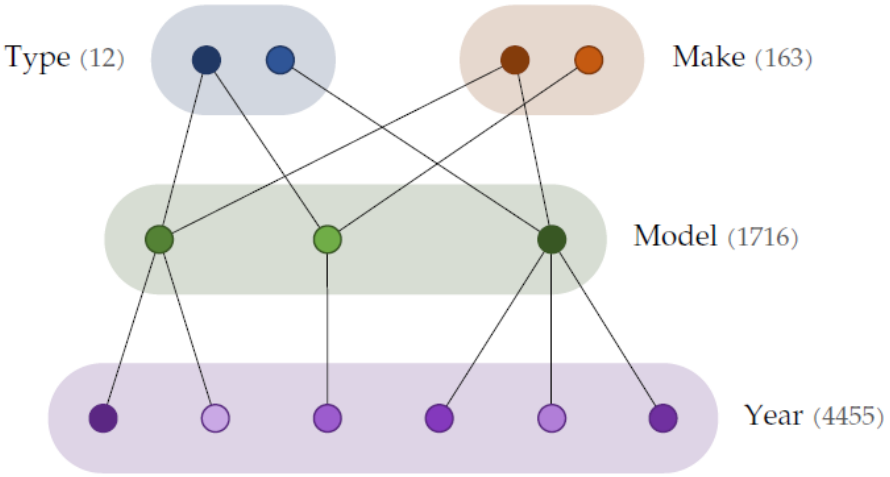

Ilustración 8.1 Jerarquía de problema de clasificación de vehículos donde se observa la relación existente entre las distintas jerarquías. Un modelo (Model) solo puede tener un fabricante/marca (Make) pero un fabricante/marca puede tener varios modelos. Fuente [65].

Descripción de arquitecturas

EfficientNet

Explicación extraída de [22]

EfficientNet es una arquitectura de red neuronal convolucional y un método de escalado que escala uniformemente todas las dimensiones de profundidad / ancho / resolución utilizando un coeficiente compuesto. A diferencia de la práctica convencional que escala arbitrariamente estos factores, el método de escala EfficientNet escala uniformemente el ancho, la profundidad y la resolución de la red con un conjunto de coeficientes de escala fijos. Por ejemplo, si queremos usar 2^n veces más recursos computacionales, entonces simplemente podemos aumentar la profundidad de la red por α^n, ancho por β^n y el tamaño de la imagen por γ^n, dónde α, β, γ son coeficientes constantes determinados por una búsqueda de cuadrícula pequeña en el modelo pequeño original. EfficientNet utiliza un coeficiente compuesto para escalar uniformemente el ancho, la profundidad y la resolución de la red de una manera basada en principios.

El método de escalado compuesto se justifica por la intuición de que, si la imagen de entrada es más grande, entonces la red necesita más capas para aumentar el campo receptivo y más canales para capturar patrones más finos de una imagen más grande.

Además, como información extra se añade la comparación de diferentes métodos de escalado y la arquitectura de la red básica EfficientNet-B0.

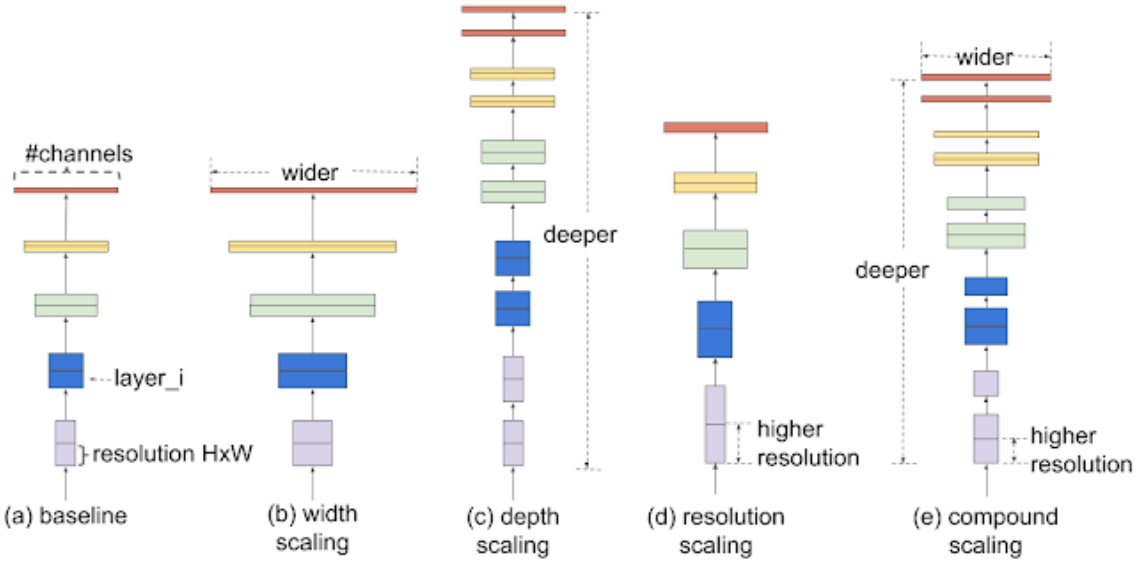

Ilustración 8.2 comparación de diferentes métodos de escalado. A diferencia de los métodos de escalado convencionales (b) - (d) que escalan arbitrariamente una sola dimensión de la red, nuestro método de escalado compuesto escala uniformemente todas las dimensiones de una manera basada en principios. Fuente: *[66]*.

EfficientNet Architecture

The effectiveness of model scaling also relies heavily on the baseline network. So, to further improve performance, we have also developed a new baseline network by performing a neural architecture search using the AutoML MNAS framework, which optimizes both accuracy and efficiency (FLOPS). The resulting architecture uses mobile inverted bottleneck convolution (MBConv), similar to MobileNetV2 and MnasNet, but is slightly larger due to an increased FLOP budget. We then scale up the baseline network to obtain a family of models, called EfficientNets.

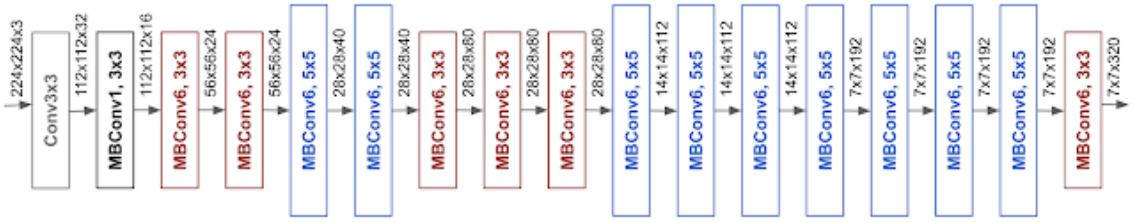

Ilustración 8.3 La arquitectura de nuestra red básica EfficientNet-B0 es simple y limpia, lo que facilita la escala y generalización. Fuente: [66].

ResNet

La arquitectura ResNet se compone de las capas de la siguiente ilustración.

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		[3×3, 64; 3×3, 64] ×2	[3×3, 64; 3×3, 64] ×3	[1×1, 64; 3×3, 64; 1×1, 256] ×3	[1×1, 64; 3×3, 64; 1×1, 256] ×3	[1×1, 64; 3×3, 64; 1×1, 256] ×3
conv3_x	28×28	[3×3, 128; 3×3, 128] ×2	[3×3, 128; 3×3, 128] ×4	[1×1, 128; 3×3, 128; 1×1, 512] ×4	[1×1, 128; 3×3, 128; 1×1, 512] ×4	[1×1, 128; 3×3, 128; 1×1, 512] ×8
conv4_x	14×14	[3×3, 256; 3×3, 256] ×2	[3×3, 256; 3×3, 256] ×6	[1×1, 256; 3×3, 256; 1×1, 1024] ×6	[1×1, 256; 3×3, 256; 1×1, 1024] ×23	[1×1, 256; 3×3, 256; 1×1, 1024] ×36
conv5_x	7×7	[3×3, 512; 3×3, 512] ×2	[3×3, 512; 3×3, 512] ×3	[1×1, 512; 3×3, 512; 1×1, 2048] ×3	[1×1, 512; 3×3, 512; 1×1, 2048] ×3	[1×1, 512; 3×3, 512; 1×1, 2048] ×3
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

Ilustración 8.4 Arquitectura de ResNet. Fuente: [4].

DenseNet

La arquitectura DenseNet se compone de las capas de la siguiente ilustración.

Layers	Output Size	DenseNet-121($k = 32$)	DenseNet-169($k = 32$)	DenseNet-201($k = 32$)	DenseNet-161($k = 48$)
Convolution	112×112	7×7 conv, stride 2			
Pooling	56×56	3×3 max pool, stride 2			
Dense Block (1)	56×56	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 6$
Transition Layer (1)	56×56	1×1 conv			
	28×28	2×2 average pool, stride 2			
Dense Block (2)	28×28	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 12$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 12$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 12$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 12$
Transition Layer (2)	28×28	1×1 conv			
	14×14	2×2 average pool, stride 2			
Dense Block (3)	14×14	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 24$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 32$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 48$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 36$
Transition Layer (3)	14×14	1×1 conv			
	7×7	2×2 average pool, stride 2			
Dense Block (4)	7×7	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 16$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 32$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 32$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 24$
Classification Layer	1×1	7×7 global average pool			
		1000D fully-connected, softmax			

Ilustración 8.5 Arquitectura de DenseNet. Fuente: [31].

Vision Transformer

Información extraída de [67]

Transformers for Image Recognition at Scale

Thursday, December 3, 2020

Posted by Neil Houlsby and Dirk Weissenborn, Research Scientists, Google Research

While convolutional neural networks (CNNs) have been used in computer vision since the 1980s, they were not at the forefront until 2012 when AlexNet surpassed the performance of contemporary state-of-the-art image recognition methods by a large margin. Two factors helped enable this breakthrough: (i) the availability of training sets like ImageNet, and (ii) the use of commoditized GPU hardware, which provided significantly more compute for training. As such, since 2012, CNNs have become the go-to model for vision tasks.

The benefit of using CNNs was that they avoided the need for hand-designed visual features, instead learning to perform tasks directly from data “end to end”. However, while CNNs avoid hand-crafted feature-extraction, the architecture itself is designed specifically for images and can be computationally demanding. Looking forward to the next generation of scalable vision models, one might ask whether this domain-specific design is necessary, or if one could successfully leverage more domain agnostic and computationally efficient architectures to achieve state-of-the-art results.

As a first step in this direction, we present the Vision Transformer (ViT), a vision model based as closely as possible on the Transformer architecture originally designed for text-based tasks. ViT represents an input image as a sequence of image patches, similar to the sequence of word embeddings used when applying Transformers to text, and directly predicts class labels for the image. ViT demonstrates excellent performance when trained on sufficient data, outperforming a comparable state-of-the-art CNN with four times

fewer computational resources. To foster additional research in this area, we have open-sourced both the code and models.

The Vision Transformer treats an input image as a sequence of patches, akin to a series of word embeddings generated by a natural language processing (NLP) Transformer.

The Vision Transformer

The original text Transformer takes as input a sequence of words, which it then uses for classification, translation, or other NLP tasks. For ViT, we make the fewest possible modifications to the Transformer design to make it operate directly on images instead of words, and observe how much about image structure the model can learn on its own.

ViT divides an image into a grid of square patches. Each patch is flattened into a single vector by concatenating the channels of all pixels in a patch and then linearly projecting it to the desired input dimension. Because Transformers are agnostic to the structure of the input elements we add learnable position embeddings to each patch, which allow the model to learn about the structure of the images. A priori, ViT does not know about the relative location of patches in the image, or even that the image has a 2D structure — it must learn such relevant information from the training data and encode structural information in the position embeddings.

Scaling Up

We first train ViT on ImageNet, where it achieves a best score of 77.9% top-1 accuracy. While this is decent for a first attempt, it falls far short of the state of the art — the current best CNN trained on ImageNet with no extra data reaches 85.8%. Despite mitigation strategies (e.g., regularization), ViT overfits the ImageNet task due to its lack of inbuilt knowledge about images.

To investigate the impact of dataset size on model performance, we train ViT on ImageNet-21k (14M images, 21k classes) and JFT (300M images, 18k classes), and compare the results to a state-of-the-art CNN, Big Transfer (BiT), trained on the same datasets. As previously observed, ViT performs significantly worse than the CNN equivalent (BiT) when trained on ImageNet (1M images). However, on ImageNet-21k (14M images) performance is comparable, and on JFT (300M images), ViT now outperforms BiT.

Finally, we investigate the impact of the amount of computation involved in training the models. For this, we train several different ViT models and CNNs on JFT. These models span a range of model sizes and training durations. As a result, they require varying

amounts of compute for training. We observe that, for a given amount of compute, ViT yields better performance than the equivalent CNNs.

Left: Performance of ViT when pre-trained on different datasets. Right: ViT yields a good performance/compute trade-off.

High-Performing Large-Scale Image Recognition

Our data suggest that (1) with sufficient training ViT can perform very well, and (2) ViT yields an excellent performance/compute trade-off at both smaller and larger compute scales. Therefore, to see if performance improvements carried over to even larger scales, we trained a 600M-parameter ViT model.

This large ViT model attains state-of-the-art performance on multiple popular benchmarks, including 88.55% top-1 accuracy on ImageNet and 99.50% on CIFAR-10. ViT also performs well on the cleaned-up version of the ImageNet evaluations set "ImageNet-Real", attaining 90.72% top-1 accuracy. Finally, ViT works well on diverse tasks, even with few training data points. For example, on the VTAB-1k suite (19 tasks with 1,000 data points each), ViT attains 77.63%, significantly ahead of the single-model state of the art (SOTA) (76.3%), and even matching SOTA attained by an ensemble of multiple models (77.6%). Most importantly, these results are obtained using fewer compute resources compared to previous SOTA CNNs, e.g., 4x fewer than the pre-trained BiT models.

Vision Transformer matches or outperforms state-of-the-art CNNs on popular benchmarks. Left: Popular image classification tasks (ImageNet, including new validation labels ReaL, and CIFAR, Pets, and Flowers). Right: Average across 19 tasks in the VTAB classification suite.

Visualizations

To gain some intuition into what the model learns, we visualize some of its internal workings. First, we look at the position embeddings — parameters that the model learns to encode the relative location of patches — and find that ViT is able to reproduce an intuitive image structure. Each position embedding is most similar to others in the same row and column, indicating that the model has recovered the grid structure of the original images. Second, we examine the average spatial distance between one element attending to another for each transformer block. At higher layers (depths of 10-20) only global features are used (i.e., large attention distances), but the lower layers (depths 0-5) capture both global and local features, as indicated by a large range in the mean attention distance. By contrast, only local features are present in the lower layers of a CNN. These experiments indicate that ViT can learn features hard-coded into CNNs

(such as awareness of grid structure), but is also free to learn more generic patterns, such as a mix of local and global features at lower layers, that can aid generalization.

Left: ViT learns the grid like structure of the image patches via its position embeddings. Right: The lower layers of ViT contain both global and local features, the higher layers contain only global features.

Summary

While CNNs have revolutionized computer vision, our results indicate that models tailor-made for imaging tasks may be unnecessary, or even sub-optimal. With ever-increasing dataset sizes, and the continued development of unsupervised and semi-supervised methods, the development of new vision architectures that train more efficiently on these datasets becomes increasingly important. We believe ViT is a preliminary step towards generic, scalable architectures that can solve many vision tasks, or even tasks from many domains, and are excited for future developments.

A preprint of our work as well as code and models are publically available.

Acknowledgements

We would like to thank our co-authors in Berlin, Zürich, and Amsterdam: Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, and Jakob Uszkoreit. We would like to thank Andreas Steiner for crucial help with infrastructure and open-sourcing, Joan Puigcerver and Maxim Neumann for work on large-scale training infrastructure, and Dmitry Lepikhin, Aravindh Mahendran, Daniel Keysers, Mario Lučić, Noam Shazeer, and Colin Raffel for useful discussions. Finally, we thank Tom Small for creating the Visual Transformer animation in this post.