Universidad Autónoma de Yucatán

Facultad de Matemáticas

Reconocimiento del Modelo de Vehículo con Ensamble de Redes Neuronales Convolucionales

Tesis

para obtener el grado de:

Maestría en Ciencias de la Computación

Presenta:

Josué David Pinzón Vivas

Directora de tesis:

Dr. Anabel Martín González

Mérida Yucatán, México Octubre 2020

Agradecimientos

Que estas líneas sirvan para expresar mi más profundo y sincero agradecimiento a todas aquellas personas que con su ayuda han colaborado en la realización del presente trabajo:

Para empezar, quiero agradecer a mi asesora de tesis la Dr. Anabel Martín González por el apoyo brindado durante el desarrollo de la misma, sin su ayuda este trabajo no sería posible. Por otra parte, quiero agradecer a la Facultad de Matemáticas de la Universidad Autónoma de Yucatán, por brindarme una segunda casa, en la cual, junto con todos sus docentes, me permitieron seguir creciendo como persona y profesionista.

Por último, pero no menos importante, agredezco todo el apoyo brindado por mis familiares. De manera especial, a mi compañera de vida, mejor amiga y esposa Brenely Herrera, al igual que a mi hija Roesly Pinzón, quienes fueron mi motivación constante y, que sin ellas y su apoyo, no hubiese sido posible la conclusión de mi tesis.

Resumen

El constante incremento en la creación e implementación de sistemas inteligentes de transporte ha dado a lugar al interés por desarrollar nuevos y mejores modelos de reconocimiento de vehículos. Usualmente existen dos enfoques hacia el reconocimiento de vehículos, el reconocimiento de grano grueso, que busca clasificar a los vehículos acorde a su tamaño o tipo, y el de grano fino, que busca clasificar los vehículos a nivel marca o modelo.

En la literatura sobre reconocimiento de vehículos es común encontrar trabajos que realizan esta tarea utilizando sólo un punto de vista de los vehículos, y son escasos los trabajos que se enfocan al problema de múltiple vista. El reconocimiento de vehículos usando múltiples vistas supone ser un problema más desafiante, pero es necesario para extender los límites de los actuales sistemas inteligentes.

En este trabajo se propuso un modelo basado en redes neuronales convolucionales de alta eficiencia para resolver el problema de reconocimiento de vehículos de grano fino usando imágenes de múltiples vistas. Se creó un Ensamble de Efficient-Nets que superó los resultados del estado del arte en reconocimiento de vehículos. El modelo propuesto logró una *accuracy* Top-1 de 94.1 % en la base de datos CompCars a nivel modelo y una *accuracy* Top-1 de 96.67 % a nivel marca.

Índice general

Índice de figuras

Capítulo 1

Introducción

La clasificación de vehículos es una tarea de vital importancia debido al incremento de sistemas inteligentes de transporte, como vigilancia automática [6, 7], cobro electrónico de peajes [8, 9] y estacionamientos inteligentes [10, 11]. También es de gran utilidad en otras aplicaciones, como sistemas de asistencia de manejo [12], monitoreo del flujo de tráfico [13], rastreo de vehículos [14], entre otras [15–17].

La clasificación y reconocimiento de imágenes es un problema muy abordado desde ya varios años [18–21], sin embargo, en el último lustro se ha logrado un gran avance a través del uso de técnicas de aprendizaje profundo, en especial, con el uso de Redes Neuronales Convolucionales [22–26]. Estas últimas, han mostrado un gran desempeño en tareas de clasificación, obteniendo los más altos índices de precisión [27, 28]. En la década pasada, la mayoría de los trabajos enfocados al reconocimiento de vehículos usaban descriptores manuales de imágenes como SIFT [29–31], SURF [32, 33] y HOG [34, 35]. No es de extrañarse que en años recientes sea tendencia el uso de Redes Neuronales Convolucionales para el reconocimiento de vehículos [36–39, 2, 40].

A pesar de que existen trabajos que tratan de resolver el problema de clasificación de vehículos,

muchas propuestas siguen presentando limitaciones y retos a resolver. Es por ello, que en este trabajo se propone un sistema automático basado en Redes Neuronales Convolucionales para la clasificación de vehículos por tipo.

1.1 Reconocimiento de Vehículos

El reconocimiento de vehículos es un problema que consiste en obtener información del vehículo a través de algún método. Como ya se ha dicho, en años recientes es tendencia usar el reconocimiento automático a través de imágenes. Por lo tanto, cuando se habla de "reconocimiento" generalmente se da por sentado a que es a través de imágenes.

En años pasados el reconocimiento de vehículos se limitaba únicamente al reconocimiento de placas, con el cual, al obtener el número de placa y a través de una base de datos se obtenía la información del vehículo. Sin embargo, la tarea de reconocimiento de placas supone dos grandes limitantes, la primera es que en la imagen la placa debe estar visible y lo suficientemente legible, y la segunda, la placa pudiese ser falsificada o robada. Por ello, es que los trabajos recientes se enfocan en extraer características directamente del vehículo a través de descriptores y redes neuronales convolucionales.

1.2 Trabajos Previos

En los trabajos previos los autores suelen dividirse principalmente en dos ramas de reconocimiento de vehículos, por grano fino y grano grueso. El reconocimiento de grano fino consiste en obtener atributos detallados del vehículo, generalmente estos atributos corresponden a marca y modelo del vehículo, sin embargo, igual pueden ser punto de vista del vehículo, año de fabricación, número de puertas, etc. Por otro lado, el reconocimiento de grano grueso consiste en clasificar los

vehículos de acuerdo a su tamaño, principalmente se enfocan en obtener el tipo de vehículo, que puede ser sedan, camioneta, camión, compacto, etc.

El estado del arte en reconocimiento de vehículos se presenta en las dos categorías antes descritas, reconocimiento de grano fino y reconocimiento de grano grueso.

1.2.1 Reconocimiento de Grano Fino

Los trabajos de Hsieh et al.[41] y Chen et al. [42], implementan una versión enriquecida y mejorada de SURF como descriptor para detectar vehículos en conjunto con un nuevo esquema de clasificación basada en una representación esparcida para reconocer la marca y modelo del vehículo. SURF fue desarrollada en 2006 para resolver problemas de visión computacional, sobre todo en el reconocimiento de objetos [43]. Una versión mejorada de esta herramienta, BoSURF, es utilizada por Siddiqui et al. [44]. Sus resultados en precisión mejoran en comparación a los otros dos trabajos, pero su velocidad de procesamiento por imagen disminuye considerablemente.

He et al. [45], presentan un método para el reconocimiento de marca y modelo de vehículos usando imágenes de cámaras de tráfico. El vehículo es localizado usando un detector basado en partes a la vez que localiza la placa y lámparas del vehículo para rectificar la distorsión de proyección. Las características extraídas son normalizadas para luego ser clasificadas por una ENN (siglas en inglés - *ensemble neural network*). Su método es comparado con el trabajo de Hsieh [41], y se aprecia una leve mejora en precisión, pero con un aumento considerable en el tiempo de procesamiento.

Podemos ver en el trabajo de Huang et al. [46] el uso de una CNN para el reconocimiento de logos de vehículos, como método para obtener la marca. Los autores implementa una red neuronal convolucional previamente entrenada usando una estrategia basada en PCA (siglas en inglés - *principal component analysis*) para reducir el costo computacional del entrenamiento. Obtiene

muy buenos resultados de precisión, sin embargo, está limitado al reconocimiento de los vehículos en imágenes en las que esté presente el logo y que sea lo suficientemente claro.

Gao et al. [47], presentan un modelo de red neuronal denominado *Local Tiled Convolutional Neural Network* (LTCNN) para el reconocimiento de marca y modelo usando imágenes frontales de vehículos. Antes de que la imagen sea procesada por la red neuronal, se le aplica un histograma de gradientes orientados (HOG, por sus siglas en inglés - *histogram oriented gradient*). El modelo propuesto mejora la precisión en comparación a otras técnicas de extracción de características.

Biglari et al. [48], proponen un método para el reconocimiento de marca y modelo de vehículo basado en máquinas de soporte vectorial latentes (SVM, por sus siglas en inglés - *Support vector machine*). Utiliza un algoritmo de localización para encontrar partes del vehículo y extraer sus características. El sistema aprende de las características extraídas de cada parte al mismo tiempo que de la relación espacial entre ellas. Para entrenar su modelo utilizan una base de datos creados por ellos mismos, denominada BVMMR. También utilizan la base de datos *CompCars* [49], de la cual usan las imágenes de cámaras de vigilancia. *CompCars* contiene imágenes de naturaleza web y de cámaras de vigilancia. Los datos de naturaleza web contienen 163 marcas de vehículos y 1,716 modelos diferentes. Consta de un total de 136,726 imágenes que muestran el auto completo y 27,618 que muestran partes del vehículo. Los datos de cámaras de vigilancia están compuestos por 50,000 imágenes frontales de vehículos. Su método lo compara con el de Fang et al. [50], el cual usa una CNN. Dicho trabajo, en conjunto con el sistema basado en partes, obtiene mejores resultados en precisión que el de Biglari et al. basado en SVM, quienes, sin embargo, justifican que su sistema requiere mucho menos recursos computacionales.

En el trabajo de Fang et al. [50], se demuestra que la localización de partes discriminantes del vehículo, donde las diferencias entre modelos son más notables, ayuda a resolver de mejor manera el problema de reconocimiento de marca y modelo de vehículo. Este método utiliza una CNN

para extraer la información del vehículo y, al igual que el de Biglari et al., discrimina partes de interés, pero usando una CNN. Los autores usan una SVM como método de clasificación. En este trabajo se logra obtener mejores resultados de precisión comparados con otros trabajos previos, incluyendo los antes citados.

Yu et al. [51], proponen un método de clasificación fina de vehículo que consta de dos módulos principales, detección del vehículo y clasificación. En el primer módulo, adopta la arquitectura Faster R-CNN en conjunto con un clasificador SVM para detectar vehículos en una imagen, que puede contener uno o más vehículos con fondos confusos. La salida de este módulo representa sub-imágenes que contienen un sólo vehículo, las cuales, son pasadas al módulo de clasificación. Este módulo utiliza una CNN para extraer características, junto con una red bayesiana para obtener la clasificación fina de vehículos. Los autores omiten la comparación de sus resultados de precisión en su artículo.

En el trabajo de Lu et al. [52], se propone un esquema novedoso en el reconocimiento de marca y modelo de imágenes frontales de vehículos. El esquema consiste en jerarquizar la clasificación de vehículo, dividiendo el proceso de clasificación en dos etapas. Primero, obtiene información modular del frente del vehículo para realizar una clasificación de marca, y luego realiza una sub-clasificación más fina para obtener el modelo. El autor implementa la CNN usada en el trabajo de Fang et al. [50] como método de clasificación y obtiene una precisión muy parecida al mismo.

Biglari et al. [53], implementan una mejora a su trabajo del 2017[48], agregan una característica adicional denominada Esquema en Cascada para el reconocimiento de marca y modelo de vehículo. Usando una estructura HOG-SVM, permite que su sistema sea capaz de obtener clasificaciones en las etapas previas del procesamiento. Sus resultados presentan una leve mejoría en precisión en comparación a su trabajo previo.

Soon et al. [54], proponen el uso de una red de análisis de componentes principales (PCANet)

basada en redes neuronales convolucionales (PCNN) para obtener la marca y modelo de vehículo. Este modelo se enfoca en obtener características discriminantes únicamente del faro del vehículo, de esta manera se elimina la necesidad de extraer y segmentar diversas características locales del vehículo. Las características discriminantes son obtenidas automáticamente de manera jerárquica por la PCNN, además se reduce de manera considerable el tiempo de entrenamiento. Sus resultados

Tabla 1.1: Métodos de reconocimiento de grano fino de vehículos.

Autores	Año	Método
Hsieh et al.[41]	2014	SURF+SVM
Chen et al.[42]	2015	SURF+SVM
He et al.[45]	2015	Object detection with discriminatively trained part-based models + KNN, SVM, ENN, AdaBoost
Huang et al.[46]	2015	CNN y CNN pre-entrenada basada en PCA
Siddiqui et al.[44]	2016	BoSURF + SVM
Gao et al.[47]	2016	HOG + LTCNN
Biglari et al. [48]	2017	Part Based Model + SVM
Fang et al.[50]	2017	CNN Part Based Model + SVM
Yu et al.[51]	2017	Faster R-CNN + SVM
Lu et al.[52]	2018	Esquema jerárquico con uso de CNN Part Based + SVM
Biglari et al.[53]	2018	Part Based Model + Esquema en cascada HOG-SVM
Soon et al.[54]	2018	PCNN
Zhang et al.[55]	2018	CNN

son comparados con los trabajos de Hsieh et al. [41] y Fang et al. [50]. El trabajo de Fang et al. [50] presenta mejores resultados de precisión, sin embargo, Soon et al. lograron una mejor precisión que en el trabajo de Hsieh et al. [41] y con un mayor número de clases.

Zhang et al. [55], proponen un modelo de CNN basado en aprendizaje por refuerzo de error y muestras propensas a error para la tarea de reconocimiento de marca y modelo de vehículo. La principal ventaja de este modelo radica en la reducción de un 30 % de tiempo de entrenamiento. El autor compara sus resultados con CNN sin el modelo de refuerzo de aprendizaje por error, y con el sistema HOG-SVM. Muestra una mejora en precisión, pero su muestra de ejemplos no es lo suficientemente grande para considerar del todo relevantes sus resultados.

Se puede apreciar en la tabla 1.1 el listado de los distintos métodos usados por los trabajos analizados. En ella se puede apreciar que la mayoría de los trabajos utilizan CNN en sus sistemas, ya sea como discriminador, identificador o clasificador, o incluso *end-to-end* CNN.

1.2.2 Reconocimiento de Grano Grueso

Dong et al. [56], proponen un método para clasificar vehículos por tipo usando una red neuronal convolucional semi-supervisada con imágenes frontales de vehículos. A diferencia de otros métodos, su propuesta es capaz de aprender automáticamente las características necesarias para realizar su tarea de clasificación, utilizando tanto imágenes etiquetadas como no etiquetadas para su entrenamiento. Sus resultados son comparados con diferentes métodos de clasificación evaluados con su propia base de datos (*BIT-Vehicle dataset*), obteniendo una precisión del 88.11 %. Adicionalmente, comparan sus resultados con otros trabajos que usan la base de datos de [57], obteniendo una precisión del 96.1 % con imágenes diurnas, y 89.4 % con imágenes nocturnas. Sus resultados muestran ser superiores al de los trabajos comparados.

En el trabajo de Zhuo et al. [40], se utiliza una red neuronal basada en GoogLeNet [58], para

la clasificación del tipo de vehículo en imágenes de tránsito. Utilizando la técnica de transferencia de aprendizaje, su CNN propuesta se entrena primeramente con la base de datos ILSVRC [59] y posteriormente realizan un entrenamiento fino con su propia base de datos *VehicleDataset*. Muestra obtener muy buenos resultados de precisión y un buen tiempo computacional de procesamiento por imagen.

En el trabajo de Hu et al. [1], se propone un enfoque para el reconocimiento de vehículos, tanto fino como grueso, usando múltiples señales de manera conjunta en una red neuronal convolucional, la cual, localiza el vehículo en la imagen como primera fase y luego lo clasifica. La localización permite usar imágenes con fondos variados, o en donde el vehículo no ocupa la mayor parte de la imagen. El desempeño de su modelo es comparado usando la base de datos *CompCars* [49] y una base de datos desarrollada por ellos mismos llamada *SYSU-Vehicle*. Con la base de datos *CompCars* Hu et al. obtienen una precisión máxima del 94.3 %, superando otros métodos como el AlexNet [28], OverFeat y GoogLeNet [58]. Para *SYSU-Vehicle*, dividen sus resultados de precisión según el tipo de vehículo, obteniendo la mejor precisión para los vehículos de tipo *Van* y *Truck*.

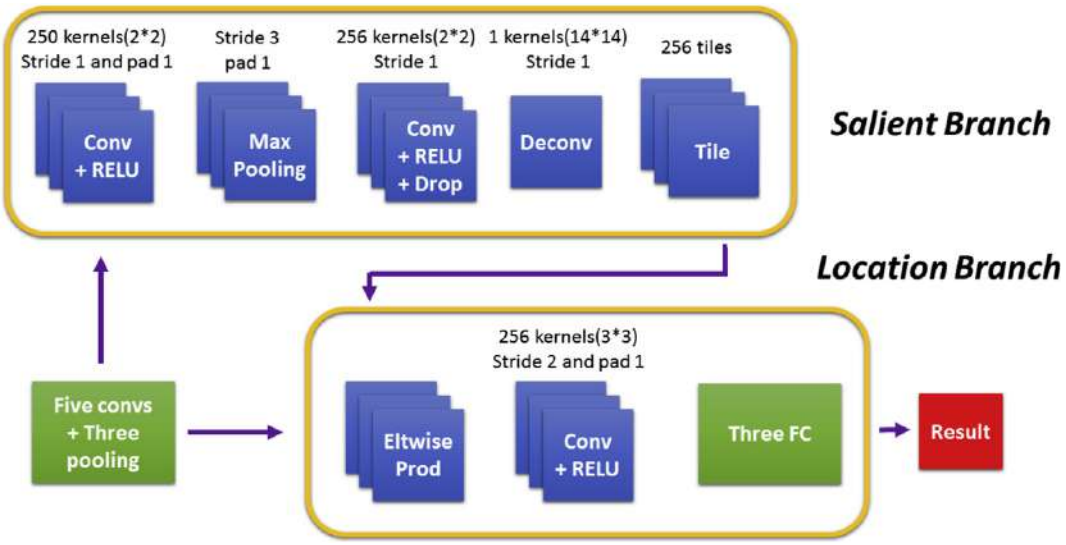


Figura 1.1: Arquitectura de la red de localización de Hu et al. tomado de [1].

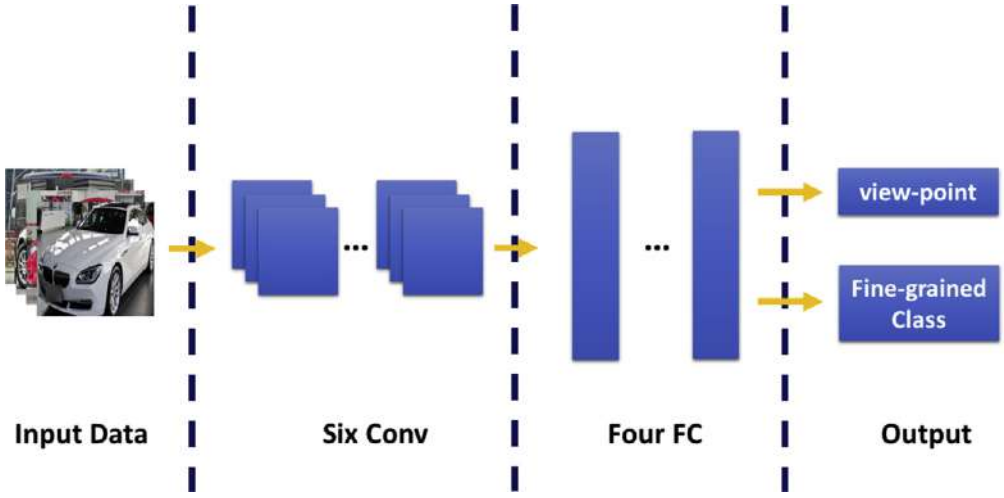


Figura 1.2: Arquitectura de la red de reconocimiento de Hu et al. tomado de [1].

Dewen et al. [36], utilizan una CNN con configuración VGG-16 [60] para la clasificación de vehículos por tipo. La red es entrenada en su propia base de datos y sus resultados son comparados con la AlexNet [28] y *K-Nearest Neighbors*.

Xue et al. [61], proponen un modelo basado en GoogLeNet [58] y una técnica denominada *feature fusion* para el reconocimiento del tipo de vehículo en imágenes de baja calidad. Sus resultados son comparados con otras técnicas de visión computacional y aprendizaje máquina, así como con otras arquitecturas de redes neuronales, como la R-NN, VGG-16, VGG-19 [60] y la GoogLeNet [58] sin aplicar *feature fusion*. Utiliza las bases de datos [62] y [63]. Los autores presentan muy buenos resultados de precisión, pero no hay una comparación directa con otros trabajos de la misma problemática.

Wang et al. [37] proponen un modelo de entrenamiento para el reconocimiento de vehículos en imágenes de cámaras de vigilancia usando imágenes de naturaleza web con transferencia de aprendizaje. La arquitectura usada por Wang et al. es similar a la de AlexNet [28]. Para el entrenamiento de su CNN usan un método de regularización en las últimas dos capas completamente conectadas a través del método de Máxima Discrepancia Media (MDM). A través de la regulari-

zación con MDM disminuyen la diferencia de las características extraídas entre las imágenes de naturaleza web y de cámaras de vigilancia. Wang et al. utilizan la base de datos CompCars [49] ya que ésta contiene tanto imágenes de naturaleza web como de cámaras de vigilancia. El método propuesto de transferencia de aprendizaje muestra mejores resultados que los anteriores métodos de transferencia existentes.

En el trabajo de Chen et al. [39], se propone un nuevo modelo para clasificar vehículos de imágenes tomadas de la vida real, basado en el algoritmo de AdaBoost y CNN. El modelo propuesto supera significativamente a los algoritmos tradicionales como SIFT-SVM [64], HOG-SVM [65] y SURF-SVM [66]. Adicionalmente, el modelo de CNN usado para extraer características contiene menos parámetros y consume menos recursos en comparación a otros modelos del estado del arte. Su arquitectura está inspirada en las arquitecturas de VGG [60] y AlexNet [28], y fue diseñada para extraer directamente las características de los vehículos. La salida de la CNN es tomada como base para el aprendizaje del algoritmo de AdaBoost, el cual utiliza SVM's como clasificadores débiles con la técnica "uno vs uno", para obtener un total de 10 clasificadores SVM. La experimentación de este trabajo se realiza con imágenes de partes traseras de vehículos, 23,510 tomadas de *CompCars* [49] y 21,720 recolectadas del mundo real. Y se utiliza un total de cinco clases de vehículos. Los resultados obtenidos demuestran una precisión del 99.5 %, además de que sólo le toma 28 ms al sistema identificar al vehículo, haciéndolo ideal para aplicaciones de tiempo-real.

Chen et al. [2], proponen un modelo de CNN con mejora de retroalimentación de múltiples ramas (FM-CNN, del inglés *Feedback-enhancement Multi-branch CNN*) para resolver la problemática de reconocimiento del tipo de vehículo usando imágenes con distintos ángulos. La red neuronal está basada en AlexNet [28], y las múltiples ramas consisten en tres derivaciones de la imagen de entrada. Un sólo valor global de error es insuficiente para entrenar la red, por lo que se emplea la mejora de retroalimentación que consiste en obtener un error por cada rama, véase Figura 1.3.

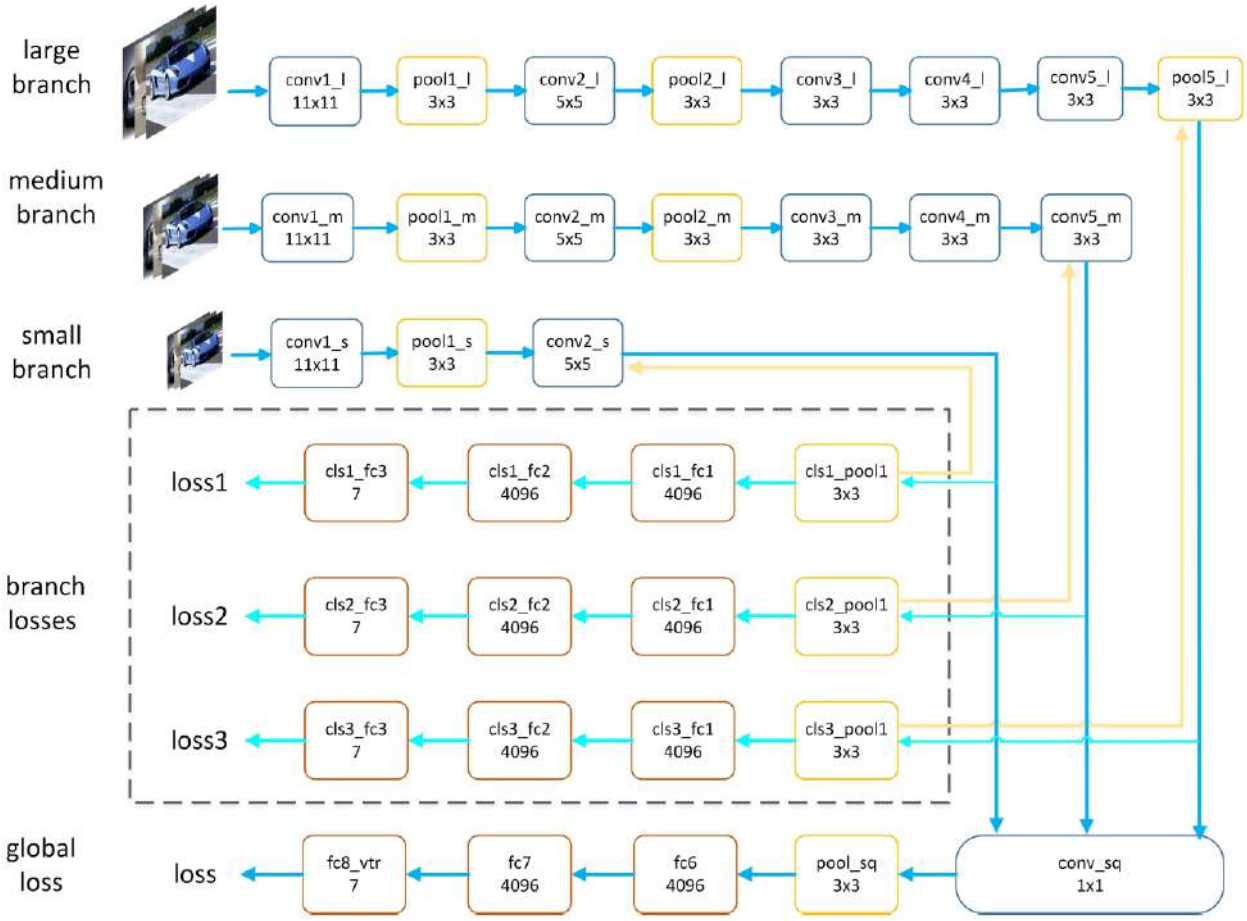


Figura 1.3: Modelo CNN con retroalimentación de múltiples ramas basado en AlexNet propuesto por Chen et al. tomado de [2].

La experimentación se realiza sobre su propia base de datos llamada *MVVTR dataset*. Los resultados obtenidos son comparados con los de otras arquitecturas de CNN, obteniendo un 94.9 % de precisión.

En la tabla 1.2 puede observarse el listado de los distintos métodos de reconocimiento de grano grueso usados por los trabajos analizados. Adicionalmente, puede notarse que todos ellos utilizan CNN en sus métodos.

De la misma manera que en los trabajos de reconocimiento de grano fino, no existe una norma o medida estándar para obtener la precisión, por lo que no se puede presentar una comparación

directa de los resultados de precisión de todos los trabajos analizados. Adicionalmente, surge la misma limitante en donde la mayoría de los trabajos únicamente usan imágenes frontales o traseras de vehículos y los que usan imágenes con múltiples ángulos aún presentan un gran margen de error. Por lo tanto, los trabajos a futuro deben enfocarse en resolver el problema usando imágenes de diversos ángulos del vehículo para mejorar las tasas de reconocimiento de los trabajos del estado del arte.

Tabla 1.2: Métodos de reconocimiento de grano grueso de vehículos.

Autores	Año	Método
Dong et al.[56]	2015	CNN + Filtro Laplaciano
Zhuo et al.[40]	2017	CNN (GoogLeNet) pre-entrenada en ILSVRC-2012
Hu et al. [1]	2017	CNN de múltiples señales para localizacion + CNN para reonocimiento
Dewen et al. [36]	2018	CNN basada en la VGG-16
Xue et al. [61]	2018	CNN con *feature fusion* basada en GoogLeNet
Wang et al. [37]	2018	CNN con nuevo método de transferencia de aprendizaje
Chen et al. [39]	2018	CNN + AdaBoost con SVM
Chen et al. [2]	2019	FM-CNN basada en AlexNet

1.3 Objetivos

1.3.1 Objetivo General

El objetivo principal de este trabajo es implementar un modelo basado en Redes Neuronales Convolucionales para el reconocimiento de tipo de vehículos usando imágenes de naturaleza web con distintos ángulos de vista del vehículo.

1.3.2 Objetivos Específicos

Los objetivos específicos de esta tesis son los siguientes:

- Implementar un modelos basado en redes neuronales convolucionales para el reconocimiento de modelo del vehículo.
- Publicar en revista científica la revisión del estado del arte sobre reconocimiento de vehículos.
- Entrenar varios modelos basados en redes neuronales convolucionales con la misma base de datos.
- Crear un ensamble de redes neuronales con los modelos previamente entrenados.
- Obtener el mayor índice de precisión en la tarea de reconocimiento del modelo del vehículo usando imágenes de múltiples vista.

1.4 Organización

El presente trabajo de tesis se encuentra estructurado en 5 capítulos incluyendo la introducción y conclusiones. A continuación, se presenta una breve síntesis del contenido de cada uno de los capítulos:

Capítulo 1.

El primer capítulo es la introducción de la tesis, la cual contiene la explicación del problema, trabajos previos, así como los objetivos y la organización del trabajo realizado.

Capítulo 2.

En este capítulo se presenta todo el marco teórico del trabajo. Se inicia con una introducción a las redes neuronales artificiales a través de la términos y conceptos base y se explican los procesos y técnicas que existen para el entrenamiento de las mismas. Luego se procede con las Redes Neuronales Convolucionales, su composición y funcionamiento. Por último, se explica y da a conocer la arquitectura base empleada en este trabajo, la EfficientNet [3].

Capítulo 3.

Este capítulo corresponde a la metodología. Se inicia con la presentación del modelo propuesto y su arquitectura. Damos paso a la explicación de los métodos y técnicas de entrenamiento que fueron empleadas, junto con la presentación de la base de datos usada. Se finaliza este capítulo describiendo la técnica utilizada para validar el modelo propuesto.

Capítulo 4.

En este capítulo se presentan los resultados obtenidos, haciendo una comparación con otros trabajos y modelos del estado del arte, así como un análisis de algunos ejemplos que el modelo no fue capaz de clasificar correctamente.

Capítulo 5.

Finalmente, se describen las conclusiones de la tesis, que corresponde a la síntesis de los logros de este estudio, el alcance de los objetivos y se proponen algunas futuras líneas de investigación y mejora.

1.5 Publicaciones

- Revisión de Métodos de Reconocimiento de Vehículos usando Aprendizaje Automático. *Abstraction & Application*, vol. 23, pp. 62-70, 2019.
- EfficientNets for a Real-Time Vehicle Recognition System. *Poster, 3rd International Symposium on Intelligent Computing Systems - ISICS 2020*

Capítulo 2

Redes Neuronales Artificiales

Las Redes Neuronales Artificiales (ANN, por sus sigles en inglés - *Artificial Neural Networks*) son un intento de imitar el funcionamiento del cerebro. Es un modelo computacional que cuenta con un conjunto de unidades, llamadas neuronas artificiales, que transmiten señales a través de ellas, aplicándoles operaciones, para emitir una o varias salidas. El modelo lógico-matemático en el que están basadas las ANN se remonta desde la década de 1950, pero no fue hasta la década de 1980 donde se crearon fuertemente los conceptos y definiciones de las ANN. Sin embargo, las ANN retomaron popularidad cerca de la década de los 2010, ya que fue el avance tecnológico y el de poder de cómputo los que permitieron poder implementar ANN en computadoras de uso personal, y con ello, volverse más atractivas a la comunidad científica.

El deseo de hacer un modelo basado en el cerebro viene inspirado en la capacidad que tiene el cerebro para realizar tareas de reconocimiento de patrones, control motriz, inferencia flexible, entre otras. Sin embargo, así como el cerebro tiene características positivas, igual presenta problemas de generalización e imprecisión, los cuales también están presentes en los modelos de redes neuronales.

2.1 Neurona Artificial

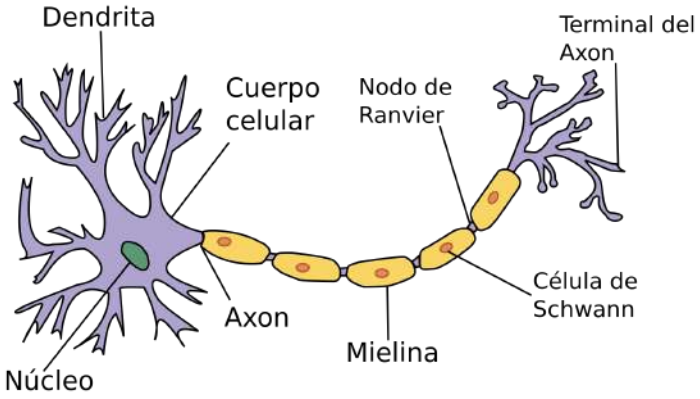


Figura 2.1: Representación de una neurona y sus partes principales.

La unidad básica de las redes neuronales artificiales está basada igualmente en la célula básica del cerebro, la neurona, véase Figura 2.1 como referencia. De manera general, las neuronas funcionan con la acumulación de potencial eléctrico a través de sus dendritas, cuando reciben cierto potencial causa que se descarguen a través de su axon, el cual está conectado con las dendritas de otras neuronas. El cerebro humano tiene billones de neuronas interconectadas entre sí y forman patrones muy complejos.

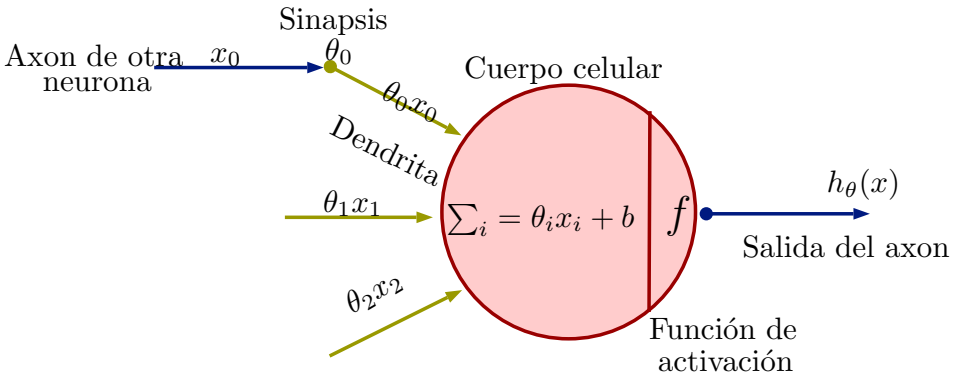


Figura 2.2: Modelo matemático de una neurona.

De manera similar a las neuronas cerebrales, las neuronas artificiales cuentan con un conjunto de pesos, una función de activación y una salida, homólogas a las dendritas, potencial eléctrico que cada neurona usa para descargarse y axones, Figura 2.2.

En notación matemática, una neurona puede verse como una función h que consta de un vector de entradas x, con su consiguiente vector de pesos θ, cuyo producto es la entrada de la función de activación f, y la salida de la función f es la salida de la neurona. En la Figura 2.3 se tiene la representación de una neurona artificial en donde $h_\theta(x)$ es la salida de la neurona dados el vector de entrada x y pesos θ:

$$x = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \qquad \theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \vdots \\ \theta_n \end{bmatrix}.$$

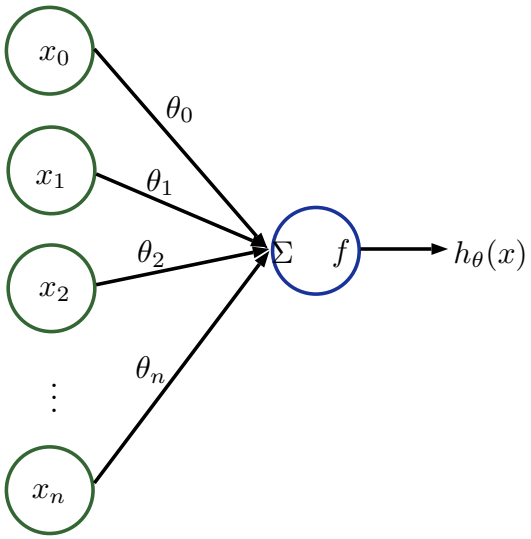


Figura 2.3: Representación de una neurona artificial.

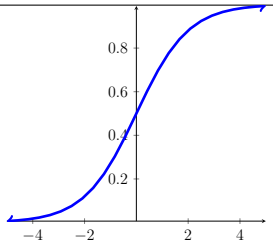
Por consiguiente la salida de la neurona puede expresarse como:

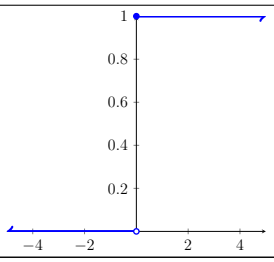
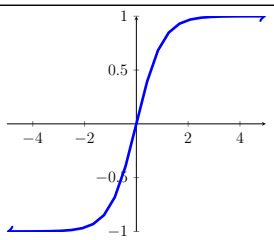
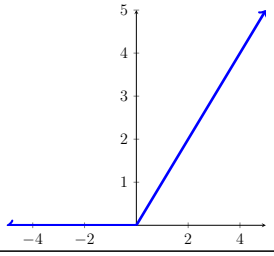
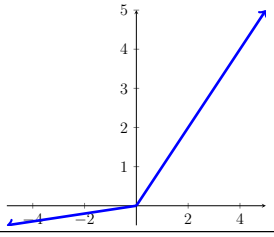
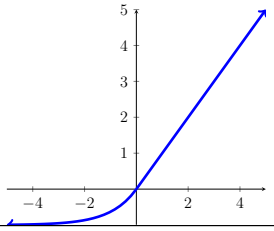
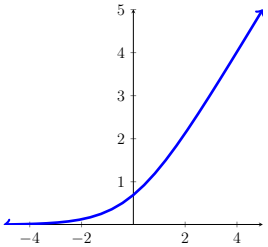
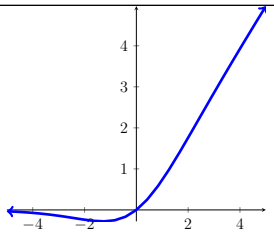
$$\begin{aligned} h_\theta(x) &= f(x_0\theta_0 + x_1\theta_1 + x_2\theta_2 + \cdots + x_n\theta_n) \\ &= f(\theta^T x)\ . \end{aligned}$$

2.2 Funciones de Activación

Las funciones de activación empleadas en las redes neuronales generalmente mapean el resultado entre en los rangos 0 a 1 o -1 a 1. Su intención es la de decidir, valores debajo de cierto umbral son considerados como un "Off" y por arriba del umbral son considerados como un "On", de ahí el nombre de "activación". Entre las funciones de activación más comunes están la Sigmoide (también llamada logística), función escalón y tangente hiperbólica, las primeras dos están mapeadas de 0 a 1 y la última de -1 a 1. Hay otras funciones de activación que no están mapeadas en dichos rangos comunes, sino que están abiertas hasta el infinito. Estas funciones de activación aún cuentan con un punto de inflexión o un umbral, que sirve como punto de decisión. Entre este tipo de funciones de activación está la ReLU (*Rectified Linear Unit*) y sus variantes: PReLU (*Parametric Rectified Linear Unit*), ELU (*Exponential Linear Unit*) y Softplus (también llamada SmoothReLU).

Tabla 2.1: Funciones de activación.

Nombre	Gráfica	Ecuación
Sigmoide		$f(x) = \dfrac{1}{1+e^{-x}}$

Escalón		$f(x) = \begin{cases} 0 & \text{para} \quad x < 0 \\ 1 & \text{para} \quad x \geq 0 \end{cases}$
Tangente hiperbólica		$f(x) = \dfrac{e^x - e^{-x}}{e^x + e^{-x}}$
ReLU		$f(x) = \begin{cases} 0 & \text{para} \quad x < 0 \\ x & \text{para} \quad x \geq 0 \end{cases}$
PReLU		$f(x) = \begin{cases} \alpha x & \text{para} \quad x < 0 \\ x & \text{para} \quad x \geq 0 \end{cases}$
ELU		$f(x) = \begin{cases} \alpha(e^x - 1) & \text{para} \quad x < 0 \\ x & \text{para} \quad x \geq 0 \end{cases}$
SoftPlus		$f(x) = \ln(1 + e^x)$
Swish		$f(x) = x \cdot \text{sigmoide}(x) = \dfrac{x}{1 + e^{-x}}$

Todas las funciones de activación cuentan con ventajas y desventajas, pero las más usadas actualmente en las redes neuronales son la sigmoide y la ReLU, esta última es altamente usada en las redes neuronales convolucionales, que serán explicadas más adelante. Como dato a observar, una neurona con activación sigmoide funge como una regresión logística, muy usada en estadística y aprendizaje máquina. Existe otra función de activación llamada *Swish* [67], la cual ha demostrado en años recientes ser más eficiente que la ReLU en redes muy profundas, por lo que su uso ha empezado a popularizarse.

2.3 Arquitectura Básica de las Redes Neuronales Artificiales

La forma más sencilla de red neuronal consta de una capa de neuronas conectadas a una neurona de salida. Cada neurona también es llamada **unidad**, y la salida de cada una es llamada **activación**. Una capa es un conjunto de unidades que reciben, generalmente, las mismas entradas. Por conveniencia, el vector de entrada es considerado como la primera capa de la red neuronal y la neurona

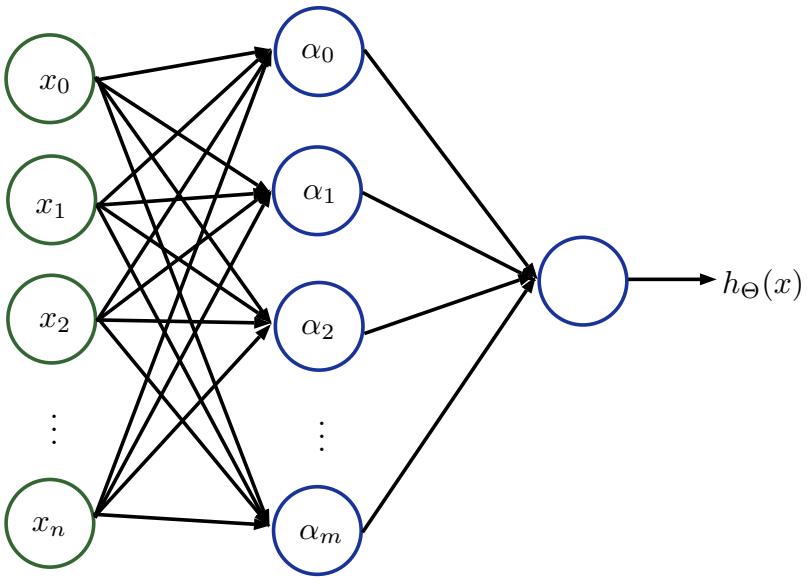


Figura 2.4: Representación de una red neuronal artificial.

de salida es considerada como la capa final. En notación matemática, una red neuronal puede verse como una función $h_\Theta(x)$, donde x es el vector de entradas y Θ es el conjunto de pesos de todas las neuronas de la red. En la Figura 2.4 se observa la representación de una ANN con una sola capa de neuronas y una salida.

Tomando de referencia la Figura 2.4, se define $\alpha_i^{(l)}$ como la activación de la unidad i de la capa l y a $\Theta^{(l)}$ como el conjunto de pesos que mapean las conexiones de la capa l a la $l+1$, de manera singular $\Theta_{i,j}^{(l)}$ representa el peso de la capa l que conecta la unidad i de la capa l a la unidad j de la capa $l+1$, se pueden expresar las activaciones de la capa intermedia como

$$\begin{aligned}
\alpha_0^{(2)} &= f(\Theta_{0,0}^{(1)}x_0 + \Theta_{1,0}^{(1)}x_1 + \cdots + \Theta_{n,0}^{(1)}x_n) \\
\alpha_1^{(2)} &= f(\Theta_{0,1}^{(1)}x_0 + \Theta_{1,1}^{(1)}x_1 + \cdots + \Theta_{n,1}^{(1)}x_n) \\
\alpha_2^{(2)} &= f(\Theta_{0,2}^{(1)}x_0 + \Theta_{1,2}^{(1)}x_1 + \cdots + \Theta_{n,2}^{(1)}x_n) \\
&\vdots \\
\alpha_m^{(2)} &= f(\Theta_{0,m}^{(1)}x_0 + \Theta_{1,m}^{(1)}x_1 + \cdots + \Theta_{n,m}^{(1)}x_n)\ .
\end{aligned}$$

Por lo tanto, la salida de la red neuronal, que es la activación de la neurona de la capa final, puede expresarse como

$$\begin{aligned}
h_\Theta(x) &= \alpha_0^{(3)} \\
&= f(\Theta_{0,0}^{(2)}\alpha_0^{(2)} + \Theta_{1,0}^{(2)}\alpha_1^{(2)} + \cdots + \Theta_{m,0}^{(2)}\alpha_m^{(2)})\ .
\end{aligned}$$

De manera más general, la arquitectura de una ANN, es decir, la forma en que están compuestas, puede estar conformada por tantas capas como uno desee y cada capa puede tener cualquier número de unidades. Además, cada capa de la red puede tener cualquier función de activación y la capa final puede estar conformada no sólo por una neurona, la cual puede ser usada para problemas binarios o de regresión, sino que puede constar de varias unidades y fungir para problemas de clasificación multiclase. Las capas neuronales donde todas las neuronas de una capa están conectadas a todas las

salidas de la capa anterior, son conocidas como **capas completamente conectadas** (FC, por sus siglas en inglés - *Fully Connected*), y las redes neuronales compuestas en su totalidad de este tipo de capas, son conocidas como redes completamente conectadas. Las ANN que están compuestas por gran cantidad de capas ocultas son llamadas redes neuronales profundas (DNN, por sus siglas en inglés - *Deep Neural Networks*), o simplemente redes profundas. No está definida una cantidad de capas para considerar una red como profunda, ya que esto cambia con el tiempo, hace 8 años la AlexNet [28] era considerada como una red profunda con sus 5 capas de convolución, en la actualidad existen modelos, como la SE-Net [5], que pueden llegar a tener más de 150 capas.

Redes Neuronales Multiclase

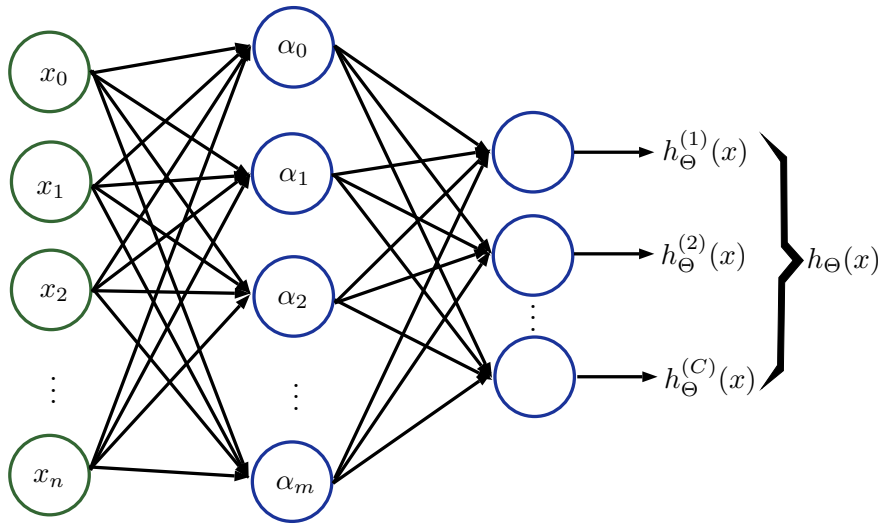


Figura 2.5: Representación de una red neuronal artificial multiclase.

Las redes neuronales diseñadas para clasificación multiclase constan de varias unidades de salida, donde el número de unidades es igual al número de clases. Las múltiples unidades "compiten" en el modo *one-vs-all*, de tal manera que la activación con mayor valor nos indica a que clase pertenece la entrada dada. En la Figura 2.5 vemos la reprepresentación de una ANN multiclase, donde h_Θ es la salida global de la ANN y C es el número de unidades de la capa final, que es el

número de clases, por lo que $h_\Theta \in \mathbb{R}^C$.

2.4 Entrenamiento de Redes Neuronales Artificiales

Entrenar una red neuronal es un trabajo complejo, matemáticamente hablando, ya que los pesos entre capas están conectados entre sí y son dependientes uno del otro. Si pensamos en una red neuronal con capas ocultas, la modificación de un sólo peso de la capa de entrada repercutirá no sólo en la neurona directamente conectada, sino que se propagará a todas las capas siguientes y por consiguiente a la capa final o salida de la red.

Muchas de las bibliotecas computacionales dedicadas a aprendizaje automático, y más específicamente a redes neuronales, tienen implementado funciones y métodos que automatizan el entrenamiento, por lo que uno puede implementar y entrenar su propia red neuronal sin conocer a fondo las técnicas y métodos que se usan para entrenar dichas redes. Sin embargo, es importante conocer dichos procesos para entender cómo las redes adquieren este aprendizaje y sacarles un mayor provecho, pudiendo reconfigurar las arquitecturas o incluso implementar mejores técnicas de aprendizaje.

2.4.1 Función de Pérdida o Costo

La función de pérdida, o también llamada función costo, es una manera de medir que tan bien está haciendo su trabajo la red neuronal. Este tipo de función no es exclusiva del área de las redes neuronales, muchas otras ramas de la computación y matemáticas usan funciones de pérdida o costo para sus problemas. En el sentido estricto, hay ligeras diferencias en cuanto a definición de lo que es una función costo a una función de pérdida, sin embargo, en el área de redes neuronales artificiales suelen usarse como sinónimos.

No existe una sola función de pérdida para las redes neuronales, ya que dependiendo del pro-

blema al que estén enfocados suele usarse una u otra función de pérdida. Entre las funciones de pérdida más usadas está el error cuadrático medio, usado principalmente para problemas de regresión y la entropía cruzada, usada principalmente para problemas de clasificación [68].

La entropía cruzada suele usarse tanto en problemas de clasificación binaria, como en problemas de clasificación multiclase. Para problemas de clasificación binaria esta función de pérdida suele recibir el nombre de entropía cruzada binaria o también pérdida logística y para problemas multiclase recibe el nombre de entropía cruzada categórica o pérdida *Softmax*. La entropía cruzada binaria no es más que un caso particular de la entropía cruzada categórica, en la que se usan únicamente dos clases.

Siguiendo la misma notación, donde $h_\Theta(x)$ es la salida de la red neuronal dado el vector de entrada x, la función de pérdida de entropía cruzada se calcula como

$$\textit{Cross-Entropy} = -\sum_{i}^{C} y_i log(h_\Theta^{(i)}(x)) \ ,$$

donde y es la verdad fundamental, es decir, la etiqueta o clase real a la que pertenece la entrada x, y C es el número de clases.

2.4.2 Algoritmo de Propagación hacia Atrás - *backpropagation*

El algoritmo de propapagación hacia atrás permite calcular los gradientes de la función de pérdida con respecto a los pesos de una manera eficiente. El gradiente indica en que dirección y magnitud cambiar los pesos para lograr una minimización de la función de pérdida.

Antes de realizar la propagación hacia atrás es necesario realizar la propagación hacia adelante. La propagación hacia adelante consiste en ir calculando los valores de salida de cada neurona de la red. Se empieza con la primera capa que es el vector de entrada, cuyos valores son conocidos, y se propagan hacia la capa siguiente, es decir, se calculan todas las activaciones de la capa siguiente. Con los valores de activación conocidos de la segunda capa, se puede seguir propagando a la

siguiente capa, así sucesivamente hasta llegar a la capa final o salida de la red. Este proceso no es más que ir computando los valores de las activaciones con las activaciones conocidas y los pesos de la red. En notación matemática podemos expresar la propagación hacia adelante como:

$$
\begin{aligned}
\alpha^{(1)} &= x \\
z^{(2)} &= \Theta^{(1)}\alpha^{(1)} \\
\alpha^{(2)} &= f(z^{(2)}) \\
z^{(3)} &= \Theta^{(2)}\alpha^{(2)} \\
\alpha^{(3)} &= f(z^{(3)}) \\
&\vdots \\
z^{(k)} &= \Theta^{(k-1)}\alpha^{(k-1)} \\
\alpha^{(k)} &= f(z^{(k)}) = h_{\Theta}(x)\ ,
\end{aligned}
$$

donde $\alpha^{(l)}$ son las activaciones de la capa l, $\Theta^{(l)}$ es la matriz de pesos que mapean las conexiones de la capa l a la capa $l+1$, $z^{(l)}$ son las entradas de la neuronas de la capa l, también conocida como función de transferencia, definida como el producto matricial $\Theta^{(l-1)}a^{(l-1)}$, f es la función de activación, $h_{\Theta}(x)$ es la salida de la red neuronal, que es la activación de la última capa y k es el número de capas de la ANN. Durante el proceso de propagación hacia adelante se guardan todas las activaciones de la red, ya que sus valores son utilizados durante la propagación hacia atrás.

Una vez realizada la propagación hacia adelante se da paso a la propagación hacia atrás, es decir, se procede a calcular el gradiente de la función costo L con respecto a cada uno de los pesos en Θ. Para calcular el gradiente con respecto a un peso en particular se utiliza la regla de la cadena

aplicada dos veces:

$$\begin{aligned}\frac{\partial L}{\partial \Theta_{i,j}^{(l)}} &= \frac{\partial L}{\partial \alpha_j^{(l+1)}} \frac{\partial \alpha_j^{(l+1)}}{\partial \Theta_{i,j}^{(l)}} \\ &= \frac{\partial L}{\partial \alpha_j^{(l+1)}} \frac{\partial \alpha_j^{(l+1)}}{\partial z_j^{(l+1)}} \frac{\partial z_j^{(l+1)}}{\partial \Theta_{i,j}^{(l)}} \,. \end{aligned} \tag{2.1}$$

Recordando que z es la función de transferencia, para una neurona en particular está definida como

$$\begin{aligned} z_j^{(l+1)} &= \Theta_{:,j}^{(l)} \alpha^{(l)} \\ &= \sum_{i=0}^{n} \Theta_{i,j}^{(l)} \alpha_i^{(l)} \\ &= \Theta_{0,j}^{(l)} \alpha_0^{(l)} + \Theta_{1,j}^{(l)} \alpha_1^{(l)} + \cdots + \Theta_{n,j}^{(l)} \alpha_n^{(l)} \,, \end{aligned} \tag{2.2}$$

donde n es el número de unidades de la capa l, por lo tanto

$$\begin{aligned} \frac{\partial z_j^{(l+1)}}{\partial \Theta_{i,j}^{(l)}} &= \frac{\partial}{\partial \Theta_{i,j}^{(l)}} \left(\Theta_{0,j}^{(l)} \alpha_0^{(l)} + \Theta_{1,j}^{(l)} \alpha_1^{(l)} + \cdots + \Theta_{n,j}^{(l)} \alpha_n^{(l)} \right) \\ &= \alpha_i^{(l)} \,. \end{aligned}$$

Los primeros dos términos de la ecuación 2.1 son conocidos como el **gradiente local**,

$$\delta_j^{(l)} = \frac{\partial L}{\partial \alpha_j^{(l)}} \frac{\partial \alpha_j^{(l)}}{\partial z_j^{(l)}} \,, \tag{2.3}$$

para facilitar notación se ha cambiado el índice $l+1$ por l. Recordar que $\alpha = f(z)$, por lo tanto

$$\frac{\partial \alpha_j^{(l)}}{\partial z_j^{(l)}} = \frac{\partial}{\partial z_j^{(l)}} f(z_j^{(l)}) = f'(z_j^{(l)}) \,, \tag{2.4}$$

que no es más que la derivada de la función de activación f evaluada en $z_j^{(l)}$. Por ello se busca que la función de activación sea diferenciable. La derivada de la función ReLU, a pesar de no ser diferenciable en 0, es muy sencilla por lo que facilita el cómputo de los gradientes.

Calcular el primer término del gradiente local es sencillo cuando se trata de la última capa de la ANN, ya que la derivada de la función de pérdida con respecto a la activación, no es más que la

derivada de la función de pérdida respecto a la salida de la red:

$$\frac{\partial L}{\partial \alpha_j^{(l)}} = \frac{\partial L(h_\Theta)}{\partial h_\Theta^{(j)}} \, .$$

En cambio, para calcular la derivada de la función de pérdida con respecto a la activación en capas intermedias se vuelve a aplicar la regla de la cadena dos veces:

$$\begin{aligned} \frac{\partial L}{\partial \alpha_j^{(l)}} &= \frac{\partial L}{\partial z_j^{(l+1)}} \frac{\partial z_j^{(l+1)}}{\partial \alpha_j^{(l)}} \\ &= \frac{\partial L}{\partial a_j^{(l+1)}} \frac{\partial a_j^{(l+1)}}{\partial z_j^{(l+1)}} \frac{\partial z_j^{(l+1)}}{\partial \alpha_j^{(l)}} \, . \end{aligned} \tag{2.5}$$

Usando la ecuación de la función de transferencia (2.2) se tiene que

$$\begin{aligned} \frac{\partial z_j^{(l+1)}}{\partial \alpha_j^{(l)}} &= \frac{\partial}{\partial \alpha_j^{(l)}} \left(\Theta_{0,j}^{(l)} \alpha_0^{(l)} + \Theta_{1,j}^{(l)} \alpha_1^{(l)} + \cdots + \Theta_{n,j}^{(l)} \alpha_n^{(l)} \right) \\ &= \Theta_{j,j}^{(l)} \, , \end{aligned}$$

y considerando la definición de gradiente local (2.3), sustituimos en (2.5):

$$\begin{aligned} \frac{\partial L}{\partial \alpha_j^{(l)}} &= \frac{\partial L}{\partial a_j^{(l+1)}} \frac{\partial a_j^{(l+1)}}{\partial z_j^{(l+1)}} \frac{\partial z_j^{(l+1)}}{\partial \alpha_j^{(l)}} \\ &= \delta_j^{l+1} \Theta_{j,j}^{(l)} \, . \end{aligned} \tag{2.6}$$

Por lo tanto, para calcular el gradiente local, sustituimos (2.4) y (2.6) en (2.3) y se obtiene:

$$\delta_j^{(l)} = \delta_j^{l+1} \Theta_{j,j}^{(l)} f'(z_j^{(l)}) \, .$$

De esta manera se obtiene una expresión recursiva para calcular los gradientes locales, en donde para calcular los gradientes de una capa en particular se necesitan conocer los gradientes de la capa siguiente. Es por ello que el algoritmo recibe el nombre de propagación hacia atrás, ya que se calculan los gradientes de la última capa, es decir, de la salida de la ANN, y luego se van calculando los gradientes de las capas anteriores.

2.4.3 Optimizadores

Los optimizadores, como su propio nombre lo dice, buscan optimizar una función, en este caso en específico, buscan optimizar los valores de los pesos de la ANN para minimizar su función de pérdida. Esta optimización se logra a través del algoritmo de descenso de gradiente, muy conocido en el área de aprendizaje máquina.

2.4.3.1 Descenso de Gradiente

El descenso de gradiente es un algoritmo iterativo que modifica los valores de los pesos en función al gradiente de la función de pérdida, y está expresado como

$$\Theta = \Theta - \alpha \nabla J \ , \tag{2.7}$$

en donde Θ son los pesos de la ANN, α es la taza de apredizaje y ∇J es el gradiente total. J es la función objetivo y lo más común es que sea el promedio de la pérdida de todos los ejemplos sobre los cuales se está entrenado y se expresa como

$$J(\Theta) = \frac{1}{n} \sum_{i}^{n} L(\Theta, x_i, y_i) \ ,$$

en donde n es la cantida de ejemplos, L es la función de pérdida y x_i, y_i son los ejemplos, es decir, la entrada x de la red y su respectiva etiqueta y. La **tasa de aprendizaje** α es un hiperparámetro que hay que ajustar manualmente, valores muy pequeños hacen que el aprendizaje sea demasiado lento y valores muy grandes pueden hacer que el modelo no converja o fluctuar dentro de un mínimo local sin nunca alcanzarlo.

El algoritmo de descenso de gradiente es óptimo para sistemas convexos, sin embargo, las ANN son sistemas sumamente irregulares y no lineales por lo que es fácil caer en un mínimo local. Debido a esta no-linealidad, se han implementado distintas versiones conocidas como descenso de gradiente estocástico, descenso de gradiente por lotes y descenso de gradiente por mini-lotes.

El descenso de gradiente por lotes es equivalente al descenso de gradiente, en donde para actualizar los pesos se calcula el gradiente total. Como ya se mencionó, con este método es fácil caer en un mínimo local, además, calcular el gradiente total requiere de una gran cantidad de memoria en la computadora, y es posible que, con bases de datos muy grandes, esto sea prácticamente imposible de computar.

El **descenso de gradiente estocástico** (**SGD**, por sus siglas en inglés - *Stochastic Gradient Descent*) consiste en aplicar el descenso de gradiente y actualizar los pesos ejemplo por ejemplo. Esto acelera la velocidad de convergencia del método y es menos propenso a estancarse en mínimos locales. El aplicar este método en bases de datos muy extensas puede alentar el entrenamiento de la ANN, ya que el actualizar los pesos con cada ejemplo se vuelve computacionalmente pesado, además, hace fluctuar mucho la función de pérdida

El **descenso de gradiente por mini-lotes** (*Mini-batch GD*) es un punto intermedio entre descenso de gradiente por lotes y SGD. En lugar de calcular el gradiente uno por uno o calcular el gradiente total directamente, se subdividen todos los ejemplos en N mini-lotes de igual tamaño y se aplica el descenso de gradiente sobre cada uno de los N mini-lotes. Al igual que el SGD, permite salirse de mínimos locales, además, tiene buena velocidad de convergencia y más estable.

El *mini-batch GD* es el algoritmo más usado en entrenamiento de ANN y generalmente suelen usar los nombres de las distintas versiones como sinónimos, comúnmente se utiliza el nombre de SGD.

2.4.3.2 Impulso - *Momentum*

El *momentum* [69] es una variación del descenso de gradiente en donde la actualización de parámetros no sólo depende del gradiente actual, sino también del paso anterior. Por ello recibe su nombre, ya que el gradiente toma un "impulso" que afecta al siguiente paso. Tomando en cuenta

la ecuación para el descenso de gradiente 2.7 se puede reescribir como

$$\Theta_t = \Theta_{t-1} - \alpha \nabla J(\Theta_{t-1}) \, ,$$

en donde t es el indicador del número de paso en el tiempo. Por lo tanto, las ecuaciones para el descenso de gradiente con *momemtum* quedan expresadas como

$$m_t = \beta m_{t-1} + \alpha \nabla J(\Theta_{t-1})$$

$$\Theta_t = \Theta_{t-1} - m_t \, ,$$

donde β es un hiperparámetro que nos indica que tanto *momemtum* aplicar. Mientras mayor sea β más "impulso" toma el gradiente del paso anterior, en cambio, si $\beta = 0$, no hay "impulso" alguno y es equivalente a la fórmula original del descenso de gradiente.

2.4.3.3 Gradiente Acelerado de Nesterov - *NAG*

El gradiente acelerado de Nesterov [70] (NAG, por sus siglas en inglés - *Nesterov Accelerated Gradient*) es una variación a la fórmula del *momentum*. En lugar de calcular el gradiente con respecto al paso de tiempo actual se calcula con respecto a una aproximación del paso siguiente, y queda expresado como:

$$m_t = \beta m_{t-1} + \alpha \nabla J(\Theta_{t-1} - \beta m_{t-1})$$

$$\Theta_t = \Theta_{t-1} - m_t \, .$$

Tanto el *momemtum* como el NAG aceleran el proceso de aprendizaje y también ayudan a caer menos en mínimos locales, sin embargo, siguen siendo dependientes del hiperparámetro de la tasa de apredizaje α.

2.4.3.4 Métodos adaptativos

Los anteriores optimizadores son todos dependientes del hiperparámetro α, la tasa de aprendizaje, encontrar el valor ideal puede resultar tedioso y consumir mucho tiempo en pruebas. Como

primer intento para resolver este problema, se propuso iniciar con una tasa de aprendizaje e ir reduciéndola cada número determinado de épocas de entrenamiento, esto mejora tanto la velocidad de entrenamiento como la precisión al final del entrenamiento, sin embargo, ya no sólo se depende del hiperparámetro de la tasa de aprendizaje, sino que también se tiene que decidir a qué ritmo ir reduciendo dicha tasa, la cual supone otro ajuste de hiperparámetro.

Para resolver el problema del ajuste manual de la tasa de aprendizaje, se propuso un método adaptativo, el cual automáticamente reduce o aumenta la tasa de aprendizaje en función de los gradientes antes computados. Este primer método adaptativo recibió el nombre de **Adagrad** [71] (del inglés *Adaptative Gradient Algorithm*), y su fórmula está expresada como

$$\Theta_{t+1} = \Theta_t - \frac{\alpha}{\sqrt{G_t + \epsilon}} \nabla J(\Theta_t) \, ,$$

en donde G_t es la suma de los cuadrados de los gradientes de pasos anteriores y ϵ es un valor muy pequeño para evitar división entre cero. De esta manera, se observa que cada peso $\theta \in \Theta$ tiene su propia tasa de aprendizaje para cada paso de tiempo t. Los pesos que tengan valores altos de gradientes reducirán en mayor cantidad su tasa de aprendizaje con respecto a aquellos que tengan valore bajos, e incluso, aquellos pesos con valores de gradientes muy bajos aumentarán su tasa de aprendizaje.

Con el Adagrad se eliminó la necesidad de ajustar manualmente la tasa de aprendizaje, sin embargo, el descenso de la tasa de aprendizaje resulta ser demasiado, al punto de que la ANN deja de aprender, por lo que surgieron nuevos métodos para corregir este problema. **Adadelta** [72] ajusta la tasa de aprendizaje en función a la media cuadrática (RMS, por sus siglas en inglés - *Root Mean Square*) de los gradientes y el gradiente actual, y está expresado como

$$\begin{aligned} \Theta_{t+1} &= \Theta_t + \Delta\Theta_t \\ \Delta\Theta_t &= -\frac{RMS[\Delta\Theta]_{t-1}}{RMS[\nabla J]_t} \nabla J \, . \end{aligned}$$

Similar a Adadelta, se propuso el método **RMSprop** (del inglés *Root Mean Square Propagation*) para resolver el problema del descenso excesivo de la tasa de aprendizaje de Adagrad. RMSprop ajusta la tasa de aprendizaje usando una media móvil de los gradientes al cuadrado, cuya fórmula está expresada como

$$E[g^2]_t = \beta E[g^2]_{t-1} + (1-\beta)(\nabla J)^2$$

$$\Theta_t = \Theta_{t-1} - \frac{\alpha}{\sqrt{E[g^2]_t}} \nabla J \, ,$$

en donde $E[g^2]$ es la media móvil de los gradientes al cuadrado y β es el parámetro de la media móvil. Originalmente β está definida con el valor 0.9, pero puede fungir como un hiperparámetro con valores entre 0 y 1.

Adam [73] (del inglés *Adaptive Moment Estimation*) es un método muy similar al RMSprop con *momentum*, pero su principal diferencia es que en lugar de tomar el *momentum* del gradiente, toma el promedio del primer y segundo momento del gradiente. Por último, surge un método más que resulta de incorporar Nesterov a Adam, como resultado se obtiene el optimizador **Nadam** [74].

2.4.4 Sobreajuste - *Overfitting*

El sobreajuste, o comúnmente llamado por su nombre en inglés *overfitting*, es un problema que presentan frecuentemente los modelos de aprendizaje supervisado que consiste en ajustar demasiado su aprendizaje a un conjunto de datos en particular. El *overfitting* provoca que el modelo "memorice" los datos de entrenamiento y da la apariencia de haber aprendido al nivel máximo, sin embargo, esto a su vez ocasiona que el modelo no sea capaz de generalizar correctamente los datos aprendidos por lo que no podrá predecir o clasificar correctamente datos nuevos.

En las redes neuronales artificiales se presenta *overfitting* sin importar la arquitectura u optimizadores utilizados por lo que se han implementado técnicas para reducir este problema. El primer paso para combatir el *overfitting* es saber identificarlo. Una de las maneras más fáciles de identi-

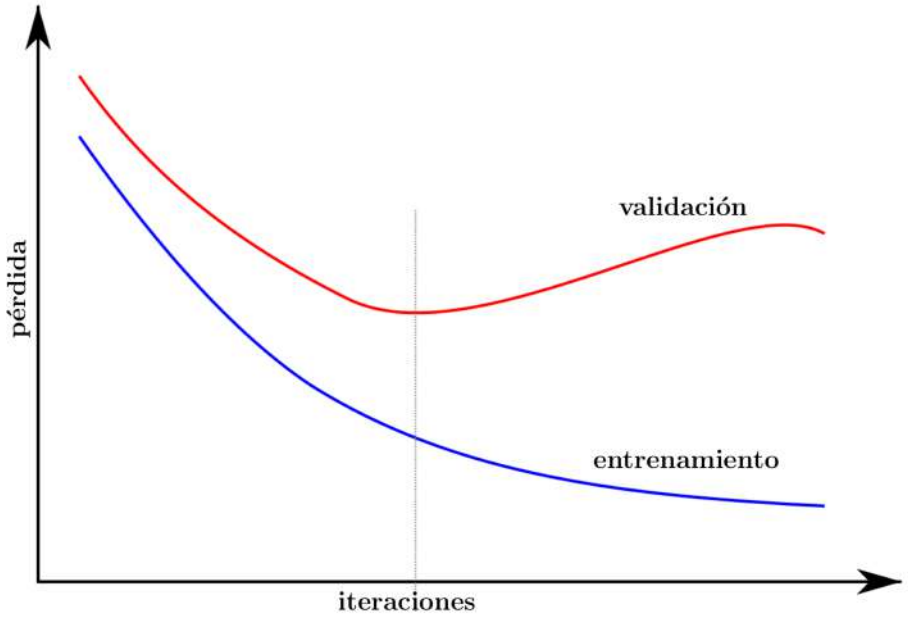


Figura 2.6: Curvas de la función de périda con *overfitting*.

ficar el overfitting es dividir la base de datos en dos conjuntos, uno de entrenamiento y el otro de validación. Teniendo los conjuntos de entrenamiento y validación se pueden obtener las gráficas de la función de pérdida de cada uno de ellos durante el entrenamiento de la ANN. En las gráficas de la función de pérdida se podrá apreciar si hay *overfitting*, esto implica que la gráfica correspondiente al conjunto de validación llega a un punto en el que su función de pérdida empieza a aumentar, como se observa en la Figura 2.6.

2.4.4.1 Regularización

La regularización es un método para combatir el overfitting. La regularización penaliza los pesos de valores muy altos, ya que estos hacen que las curvas de aprendizaje se sobre ajusten a los datos de entrenamiento. Esta penalización se logra a través de la adición de un término a la función

objetivo J, que estaba dada por:

$$J(\Theta) = \frac{1}{n} \sum_{i}^{n} L(\Theta, x_i, y_i) \ .$$

El término que se le añade la función objetivo es conocido como término de regularización y el más común es el **regularizador L2** que está dado por:

$$R_\lambda(\theta) = \frac{\lambda}{2} \sum_{\theta \in \Theta} \theta^2 \ ,$$

en donde λ es el hiperparámetro que controla la influencia del regularizador sobre la función objetivo. Por lo tanto, la función objetivo con regularizador queda expresada como:

$$J(\Theta) = \frac{1}{n} \sum_{i}^{n} L(\Theta, x_i, y_i) + R_\lambda(\theta) \ .$$

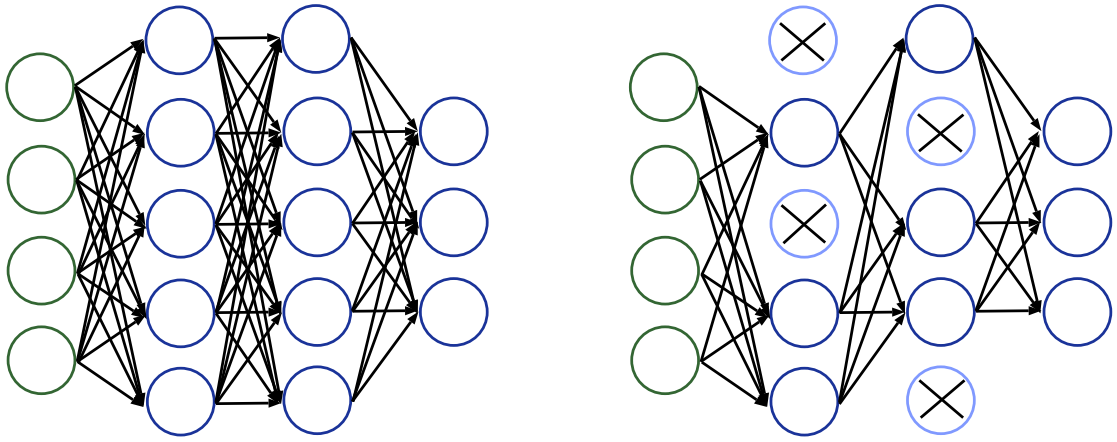
Existe una implementación del regularizador L2 con el optimizador SGD llamada en inglés como ***weight decay***. Esta implementación consiste en derivar el regularizador L2 con respecto a θ y aplicar el resultado en la función de descenso de gradiente, es decir, directamente en la actualización de pesos. El *weight decay* está expresado como:

$$\Theta = \Theta - \alpha \nabla J - \lambda \Theta \ .$$

Implementar el *weight decay* resulta más sencillo que la implementación del regularizador L2 en la función objetivo, además, requiere de menos cómputo.

2.4.4.2 Dropout

El *dropout* es una técnica muy usada actualmente en ANN para combatir el overfitting y consiste en "desactivar" aleatoriamente cierto porcentaje de neuronas y sus conexiones durante el entrenamiento como se observa en la Figura 2.7. El porcentaje de neuronas a desactivar es un hiperparámetro al que generalmente se le asigna un valor entre 20 % y 50 %. Después de cada actualización de los pesos se escoge aleatoriamente otro conjunto de neuronas a desactivar.



(a) Red Neuronal Artificial

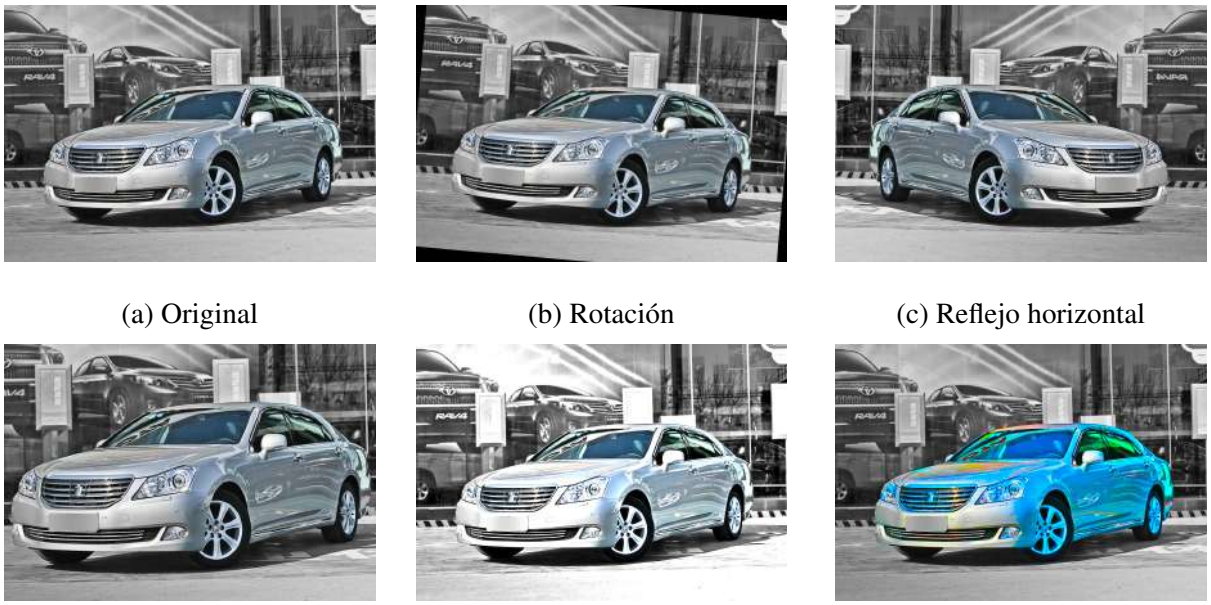
(b) Después de aplicar *dropout*

Figura 2.7: Ejemplo de red neuronal artificial y el resultado de aplicarle *dropout*.

Con el *dropout* se logra evitar que las neuronas se especialicen, es decir, que se ajusten perfectamente a los datos de entrenamiento. Al desactivarse algunas neuronas, la influencia de su aprendizaje sobre la salida de la ANN recae en las demás neuronas, por lo que éstas forman un aprendizaje más general para compensar la ausencia de las demás. El *dropout* puede ser aplicado a toda la ANN o solamente a capas en específico.

2.4.4.3 Aumento de Datos - *Data Augmentation*

El aumento de datos, mejor conocido en inglés como *data augmentation*, es una técnica que permite incrementar el tamaño de la base de datos. Las bases de datos pequeñas son más propensas a generar *overfitting*, ya que es más fácil que los modelos que se entrenen con dichas bases memoricen los datos. El *data augmentation* consiste en crear diversos ejemplos a partir de uno solo, esto se logra introduciendo ruido o pequeñas transformaciones a los datos. En imágenes es común aplicar una o varias transformaciones a la vez como pequeñas rotaciones, recortes, reflejos, ajustes en brillo, contraste y saturación, etc. En cada época del entrenamiento de la ANN se aplican nuevas transformaciones a los datos lo que evita que el modelo memorice un conjunto de datos en particular disminuyendo así el *overfitting*.



(a) Original (b) Rotación (c) Reflejo horizontal

(d) Recorte (e) Ajuste de brillo y contraste (f) Ajuste de saturación

Figura 2.8: Ejemplos de transformaciones a una imagen para generar *data augmentation*.

2.5 Redes Neuronales Convolucionales

Las Redes Neuronales Convolucionales (**CNN**, por sus siglas en inglés – *Convolutional Neural Networks*) son un tipo especial de redes neuronales artificiales enfocadas principalmente al análisis de imágenes, como clasificación, segmentación o procesamiento de imágenes. Las redes neuronales artificiales comunes no son muy buenas para este tipo de tareas, debido a que no existe correlación espacial entre los datos de entrada y no preservan simetría traslacional. Además, las ANN son extremadamente costosas, ya que el número de pesos a entrenar crece demasiado rápido según el tamaño de la imagen. A causa de esta ineficiencia de las ANN para el tratamiento de imágenes se desarrollaron las CNN, que utilizan filtros de convolución sobre las imágenes para extraer características, preservando así la simetría traslacional entre los pixeles y correlacionando a los pixeles cercanos entre sí.

2.5.1 Filtro de Convolución

El filtrado de imágenes es una herramienta muy útil en el procesamiento de imágenes y es utilizado para extraer características o aplicar transformaciones a la imagen. Durante mucho tiempo se han utilizado filtros conocidos para extraer características deseadas de la imagen, como suavizarla, obtener bordes, aplicar derivadas (aproximación discreta del gradiente), etc. La idea de utilizar filtros en las redes neuronales convolucionales es dejar que aprenda por sí misma que filtros usar para extraer las características más significativas que le permitan resolver el problema al que esté enfocada.

Para aplicar el filtrado a una imagen se utiliza una matriz, conocida como filtro o ***kernel***, con la cual se recorre la imagen realizando la operación de convolución sobre cada uno de los pixeles visitados. La operación de convolución discreta en dos dimensiones está definida como:

$$g(x,y) = h(x,y) * f(x,y) = \sum_i \sum_j f(i,j) \cdot h(x-i, y-j)\ .$$

Para aplicar la operación de convolución tal como está definida implica tener que realizar operaciones no contiguas de una matriz con otra, para simplificar este proceso el *kernel* se puede reflejar horizontal y verticalmente, facilitando la implementación y eficiencia en la computadora.

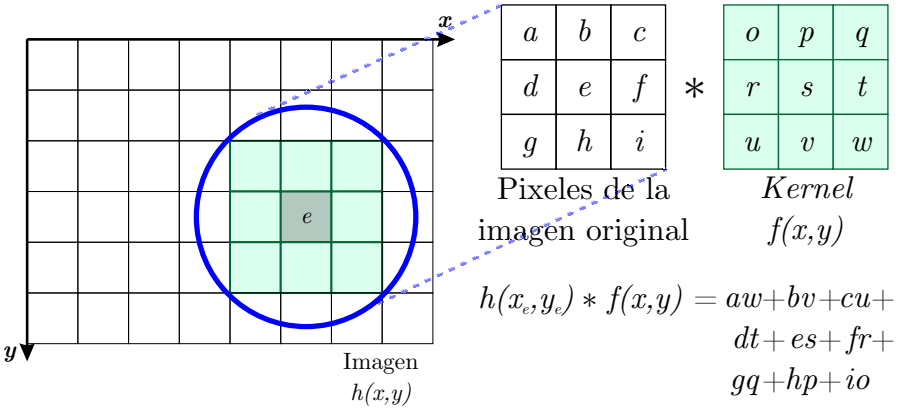


Figura 2.9: Operación de convolución sobre un pixel de una imagen.

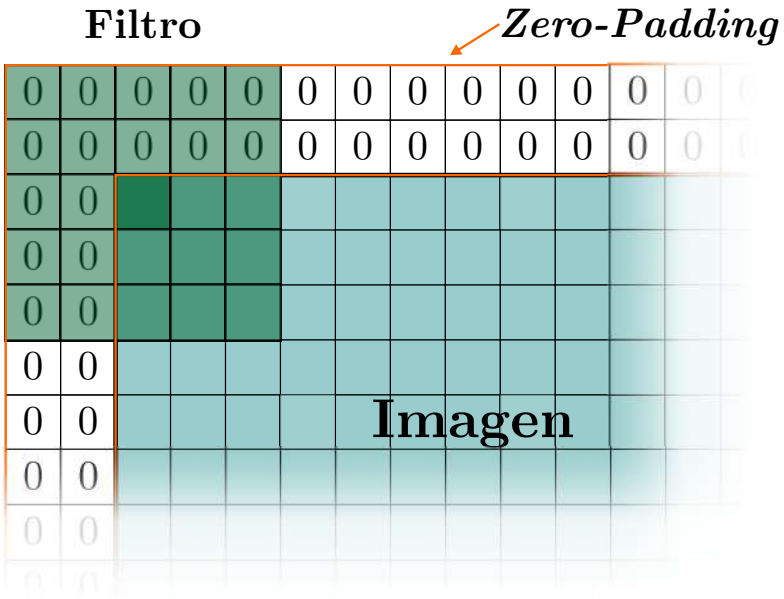


Figura 2.10: *Zero-padding* sobre una imagen a la que se le aplica un filtro de convolución de 5×5.

Se puede apreciar que la operación de convolución no está definida para los píxeles de los bordes de la imagen, debido al tamaño de la ventana del *kernel*. Se han implementado técnicas para el manejo de los bordes tales como el *padding*, replicar bordes o simplemente omitir los bordes, es decir, aplicar el filtro de convolución sólo sobres los píxeles donde sea válida la convolución. El ***padding*** consiste en añadir valores en los bordes para que la convolución pueda aplicarse sobre toda la imagen, el valor más común para utilizar es el cero (*zero-padding*). Si se utiliza un *kernel* de tamaño $F \times F$ sobre una imagen con *padding* de tamaño P, el ancho y alto de la imagen resultado se verán reducidos en $F - 1 - 2P$ unidades. Se puede notar que si el tamaño del *padding* es igual a $\frac{F-1}{2}$, entonces se conserva el tamaño original de la imagen y es conocido como ***same-padding***.

2.5.2 Capa de convolución

Las redes neuronales convolucionales están compuestas en su mayor parte de capas de convolución. Una capa de convolución consta de uno o varios filtros de convolución. Los filtros de

convolución suelen ser cuadrados y de tamaño impar, los más usados son filtros de 1×1, 3×3 o 5×5. Los filtros de gran tamaño suelen ser poco usados debido a su ineficiencia, y es preferible usar varias capas de convolución con filtros pequeños en lugar de una capa con filtros grandes. Por simplicidad, se considerará que los filtros de las CNN son siempre cuadrados.

La entrada de la capa de convolución es conocida como volumen de entrada, esto se debe a que se utilizan arreglos de 3 dimensiones. El volumen de entrada está dado por $H \times W \times D$, donde H y W son el alto y ancho respectivamente, y D es la profundidad. Si el volumen de entrada es una imagen, H y W son las dimensiones de la imagen en pixeles y D es el número de canales de color, 1 si es una imagen en escala de grises, o 3 si se trata de una imagen a color (RGB). La operación que se realiza es la convolución en 2D, para ser aplicada correctamente es necesario que el filtro herede la profundidad del volumen de entrada, es decir, que la profundidad del filtro sea igual a la profundidad del volumen de entrada. Si se tiene una imagen a color la profundidad del filtro será igual a 3, en capas posteriores la profundidad del volumen de entrada puede tomar cualquier valor.

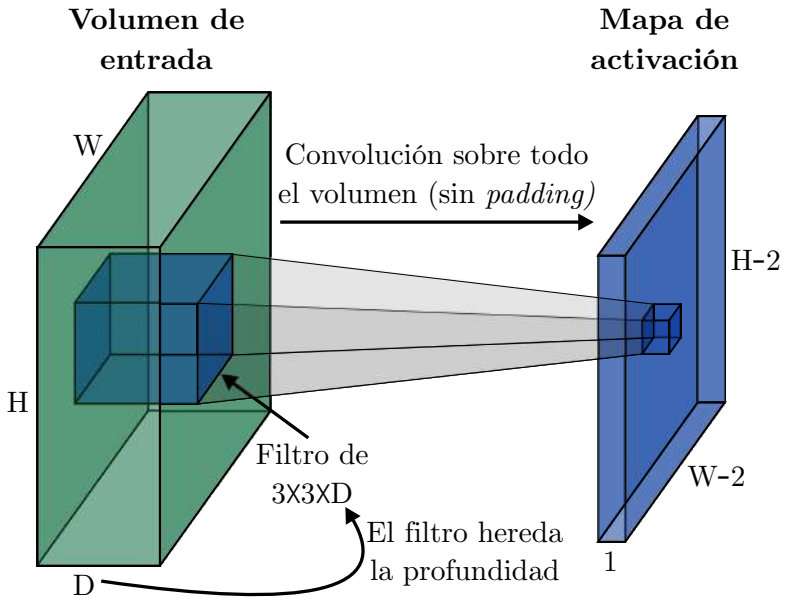


Figura 2.11: Obtención de un mapa de activación con un filtro de convolución de 3×3 sobre un volumen de entrada $H \times W \times D$, sin aplicar *padding*.

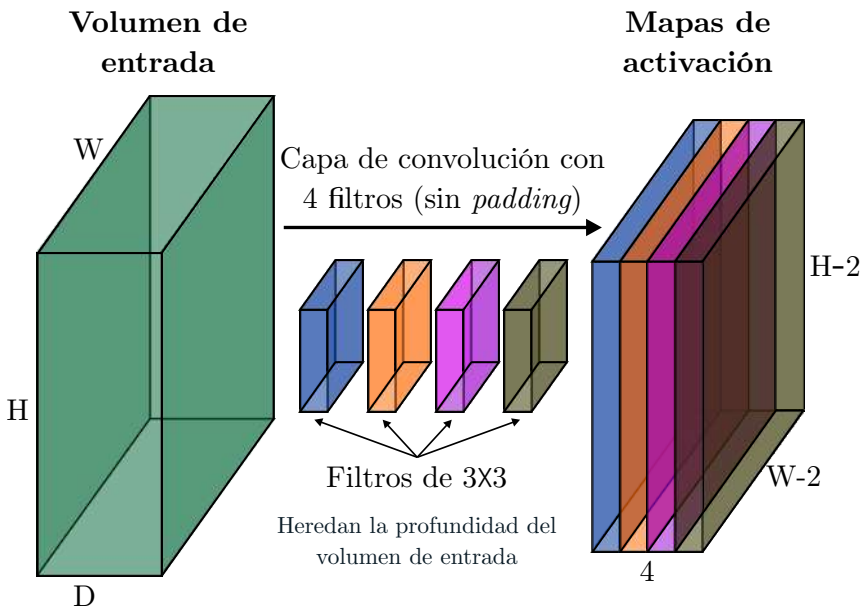


Figura 2.12: Obtención de mapas de activación con una capa de convolución de 4 filtros de 3×3 sobre un volumen de entrada $H \times W \times D$, sin aplicar *padding*.

Si la capa de convolución consta de un filtro de convolución, al aplicarlo sobre el volumen de entrada genera lo que se conoce como **mapa de activación**. Si se utilizan varios filtros de convolución, necesariamente todos de igual tamaño, se obtienen varios mapas de activación, que en conjunto forman un volumen. La profundidad del volumen de la salida será igual al número de filtros utilizados, el ancho y alto dependerán de si se utiliza *padding* o no.

Existe una técnica llamada ***striding***, que consiste en aplicar el filtro de convolución con pasos de determinado tamaño. El tamaño del paso es conocido como ***stride***. Si el *stride* es igual a 1, se aplica el filtro de convolución de manera normal, es decir, sobre toda la imagen, en cambio, si el tamaño del *stride* es igual a 2, el filtro es aplicado dando saltos de dos en dos, y así sucesivamente según el tamaño del *stride*. El *stridding* ocasiona una reducción en el ancho y alto del volumen de salida de la capa de convolución. Por lo tanto, el volumen de salida de la capa de convolución sobre un volumen $H_1 \times W_1 \times D_1$, con K *kernels* de tamaño F, *stride* S y con *padding* de tamaño

P, está dado por $H_2 \times W_2 \times D_2$, en donde:

$$H_2 = \left\lfloor \frac{H_1 - F + 2P}{S} \right\rfloor + 1$$
$$W_2 = \left\lfloor \frac{W_1 - F + 2P}{S} \right\rfloor + 1$$
$$D_2 = K \ .$$

Una capa de convolución siempre va acompañada de una **capa de activación**. La capa de activación toma como entrada la salida de la capa de convolución, es decir, los mapas de activación. La capa de activación aplica una función de activación sobre cada uno de los valores de los mapas de activación. ReLU es la función de activación más usada en redes neuronales convolucionales. En algunas ocasiones puede usarse la activación lineal $f(x) = x$, que prácticamente es equivalente a no tener función de activación, ya que los valores no son modificados.

2.5.3 Capa de Agrupación - *Pooling*

Las capas de agrupación, más conocidas por su nombre en inglés *pooling*, son capas usadas para reducir la dimensionalidad de los volúmenes con los que opera la CNN. El *pooling* es un tipo de filtrado en 2D que realiza una operación específica sobre la ventana en la que actúa. Al igual que con los filtros de convolución, se puede escoger el tamaño de la ventana y el *stride*. Generalmente se suele usar el *pooling* con un *stride* de igual tamaño que el de la ventana.

Los dos tipos de *pooling* más usados en CNN son el ***Max-pooling*** y el ***Average-pooling***. El *max-pooling* devuelve el valor máximo de los valores dentro de la ventana, mientras que el *average-pooling* devuelve el valor promedio. El *pooling* en 2D, que es el que usualmente se aplica, utiliza ventanas de 2D sobre cada una de las capas del volumen, es decir, se aplica sobre el ancho y alto de los volúmenes, por lo que no modifica la profundidad. El caso particular en donde el tamaño de la venta es igual al tamaño de la entrada recibe el nombre de ***Global-pooling***. El Global-pooling

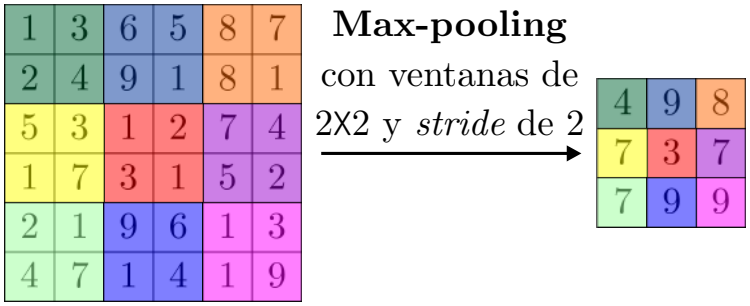


Figura 2.13: *Max-pooling* sobre un arreglo de 6×6 con ventanas de 2×2 y *stride* de 2, que da como resultado un arreglo de 3×3.

regresa un volumen de $1 \times 1 \times D$ y es generalmente usado después de la última capa de convolución de la CNN.

En las CNN generalmente se utilizan bloques de convolución. Los bloques de convolución están compuestos de varias capas de convolución, cada una seguida de su respectiva capa de activación, y luego precedidas por una capa de pooling. Aunque no es una regla utilizar los bloques de convolución de esta manera representan la estructura básica de las CNN.

2.5.4 Capas Completamente Conectadas y *SoftMax*

El bloque final de las CNN suele estar compuesto por capas completamente conectadas y generan la clasificación final de la red. La primera capa completamente conectada de dicho bloque toma directamente las características extraídas por las capas de convolución. Hay que recordar que una capa completamente conectada no es más que el modelo básico de la red neuronal artificial, también son llamadas como capas densas. Este bloque final de capas completamente conectadas generalmente está compuesto de una a tres capas, los modelos más recientes prefieren utilizar sólo una capa.

La última capa de la CNN suele utilizar una capa de activación *softmax*, sobre todo cuando se trata de problemas de clasificación multiclase y se quiere utilizar la función de pérdida de entropía

cruzada. La función ***softmax***, más que ser una función de activación, se utiliza para convertir el vector de salida de la CNN en un vector de distribución de probabilidad, es decir, todas las salidas son mapeadas en el rango [0,1] y la suma de todas ellas es igual a 1. Nótese que la salida de la CNN no indica directamente a que clase pertenece el dato de entrada, sino que indica las probabilidades de pertenencia a cada una de las clases, y la predicción final se escoge tomando la probabilidad de mayor valor.

2.6 EfficientNet

La EfficientNet es un modelo de red neuronal convolucional propuesto por Tan y Le. En su trabajo [3], proponen un nuevo **coeficiente de escalamiento compuesto** para optimizar la precisión de la CNN en función del tamaño de la misma, es decir, la memoria utilizada y el número de operaciones punto flotantes (FLOPS). Para escalar una CNN existen diversas maneras, puede escalarse a lo ancho, a lo profundo o en resolución. El escalamiento a lo ancho consiste en aumentar

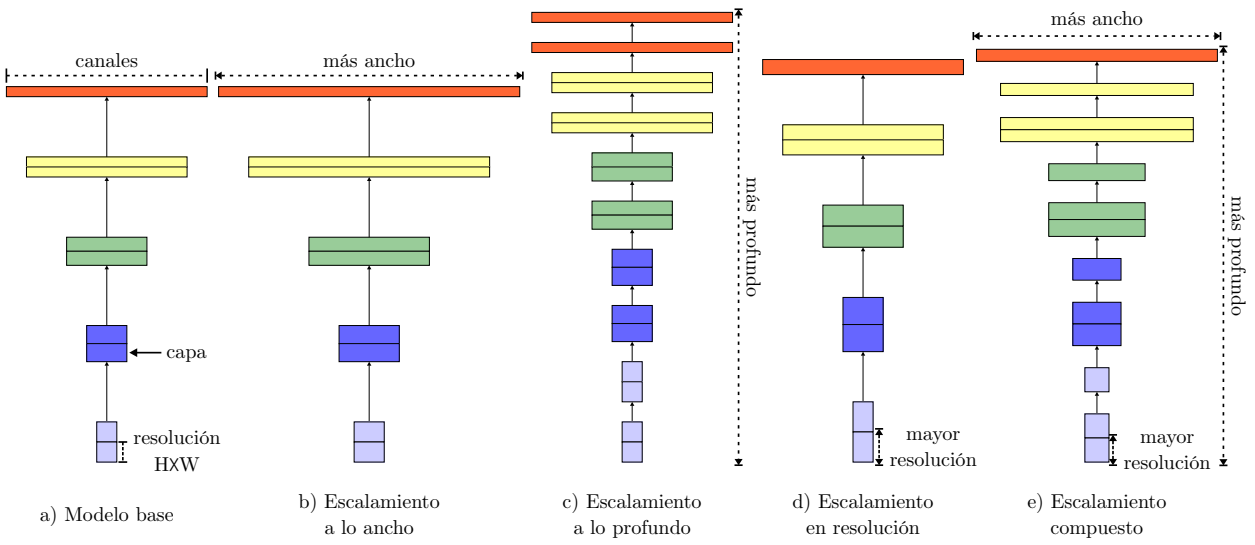


Figura 2.14: Diferentes tipos de escalamiento de CNN. a) Es el modelo base de CNN, b)-d) son escalamientos convencionales y, e) es el escalamiento propuesto por Tan y Le [3].

el número de filtros en las capas de convolución. El escalamiento a lo profundo consiste en agregar más capas de convolución a los bloques de convolución. El escalamiento en resolución consiste en aumentar el tamaño de la imagen de entrada.

En el estado del arte de las CNN es normal ver arquitecturas y modelos en donde utilizan algún tipo de escalamiento para mejorar su precisión, la mayoría de éstas utilizan el escalamiento a lo profundo. Tan y Le realizaron una investigación detallada y un análisis de los efectos de los distintos tipos de escalamiento y propusieron el escalamiento compuesto. El **escalamiento compuesto** es utilizado para escalar una CNN tanto en ancho, profundo y resolución de manera uniforme. El escalamiento compuesto demostró que genera CNN más eficientes en vez de utilizar alguno de los tipos de escalamiento por separado. Con el fin de maximizar la precisión y minimizar recursos, es decir, el tamaño de la CNN, Tan y Le propusieron un par de ecuaciones para hallar el coeficiente de escalamiento compuesto adecuado a la arquitectura de la CNN a escalar.

2.6.1 Arquitectura Base

El coeficiente de escalamiento compuesto propuesto por Tan y Le optimiza la eficiencia de la arquitectura a escalar sin alterar el orden de los operadores de la CNN, por lo tanto, es importante tener una buena arquitectura base. Tan y Le utilizaron el mismo espacio de búsqueda propuesto por otro de sus trabajos [75], con el fin de optimizar la eficiencia de la CNN, cuyo resultado produjo el modelo base llamado **EfficientNet-B0**. La arquitectura es muy similar a la MnasNet [75], propuesta igual por Tan y Le, a excepción de que la EfficientNet-B0 es ligeramente de mayor tamaño. Los bloques de convolución principales están compuestos por **cuellos de botella invertidos móviles** (**MBConv**) [4] con la optimización *squeeze-and-excitation* [5].

El **cuello de botella invertido móvil** propuesto por Sandler et al. [4], es un bloque residual[76] compuesto por una capa de expansión y una **convolución separable en profundidad**. La capa de

expansión no es más que una capa de convolución con td filtros de tamaño 1×1, en donde t es el **factor de expansión** y d es la profundidad del volumen de entrada.

La **convolución separable en profundidad** busca reemplazar una capa de convolución convencional por una versión factorizada de dos capas convolucionales. La primera capa es conocida como **convolución en profundidad** (**Dwise** conv, del inglés *depthwise convolution*), que consiste en aplicar un filtro de convolución en cada canal del volumen de entrada. La segunda capa de convolución es conocida como convolución punto a punto (*pointwise*) que consiste en una capa de convolución con filtros de 1×1. La cantidad de filtros de la segunda capa dependerá de la profundidad del volumen de salida que se desea obtener. Las convoluciones separables en profundidad son más eficientes que las convoluciones convencionales, reduciendo la cantidad de cómputo aproximadamente en un factor de k^2, donde k es el tamaño del filtro, con sólo una pequeña reducción en precisión [77].

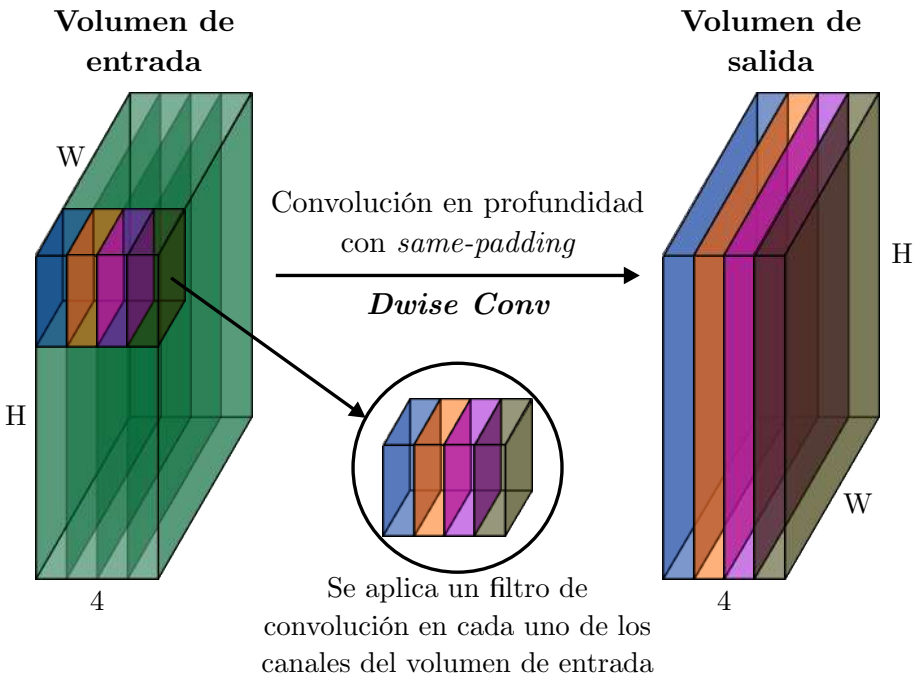


Figura 2.15: Convolución en profundidad con *same-padding* sobre un volumen de $H \times W \times 4$.

Tabla 2.2: EfficientNet-B0, arquitectura base.

Etapa	Operador	Resolución	#Canales	#Capas
1	Conv3×3	224×224	32	1
2	MBConv1,k3×3	122×112	16	1
3	MBConv6,k3×3	122×112	24	2
4	MBConv6,k5×5	56×56	40	2
5	MBConv6,k3×3	28×28	80	3
6	MBConv6,k5×5	14×14	112	3
7	MBConv6,k5×5	14×14	192	4
8	MBConv6,k3×3	7×7	320	1
9	Conv1×1 & GlobalAvgPooling & FC	7×7	1280	1

La Tabla 2.2 muestra la arquitectura de la EfficientNet-B0. La resolución indica el ancho y alto del volumen de entrada de cada etapa. El número de canales es la profundidad del volumen de salida. El número de capas es la cantidad de veces que se repite el operador. Todas las convoluciones se realizan con *same-padding*, para no modificar la resolución de los volúmenes. En las etapas que sí hay reducción de resolución, esta se logra aplicando un *stride* igual a 2 en el operador, específicamente en la capa de convolución en profundidad. En las etapas que hay reducción de resolución y además el operador se repite más de una vez, la reducción de resolución se aplica en el primer operador, por lo que en las demás repeticiones se mantiene la resolución ya reducida por el primer operador.

Los cuellos de botella invertidos móviles utilizados en la EfficientNet-B0, están detallados en la Figura 2.16. MBConv1 utiliza un factor de expansión igual a 1, por lo que se omite la capa de expansión y MBConv6 utiliza un factor de expansión igual a 6. La residual de los bloques residuales[76] sólo puede ser aplicada cuando la dimensión del volumen de entrada es exactamente

igual a la dimensión del volumen de salida. Esto sucede en los operadores que se repiten más de una vez en la misma etapa, ya que el primer operador de cada etapa es el que ajusta la resolución y número de canales. Por consiguiente, en las demás repeticiones de cada etapa los operadores mantienen las dimensiones del volumen por lo que en ellos sí se aplica la residual.

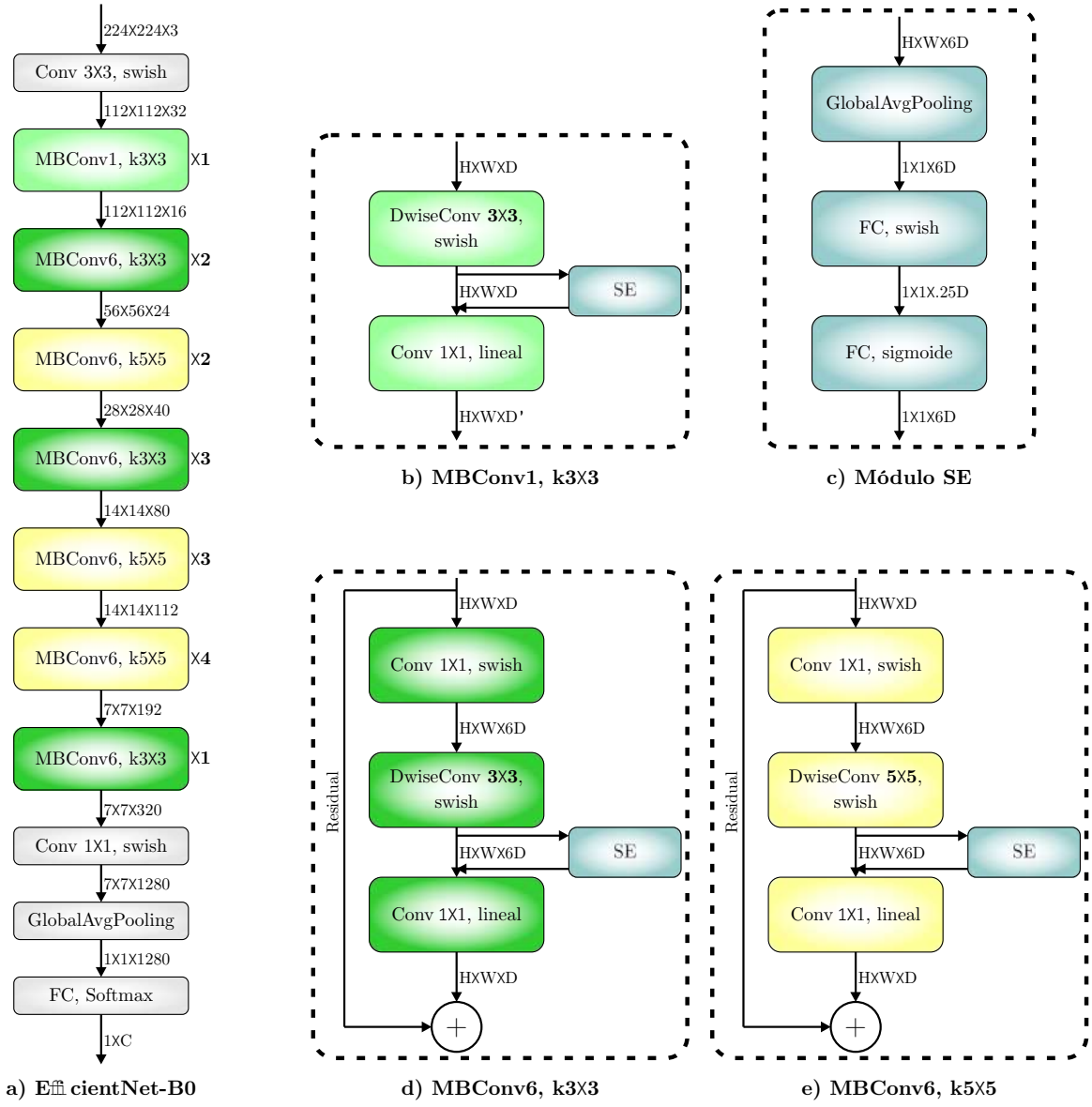


Figura 2.16: Arquitectura de la EfficientNet-B0 [3]. En **a)** se observa el diagrama general de la arquitectura completa. En **b)**, **d)** y **e)** sus respectivos cuellos de botella invertidos móviles **MB-Conv** [4]. Y en **c)** el módulo SE (*squeeze-and-excitation*) [5].

2.6.2 Escalamiento de la arquitectura

Tan y Le calcularon el coeficiente de escalamiento compuesto sobre la arquitectura de la EfficientNet-B0 con lo cual generaron siete modelos más, EfficientNet-B1 a B7. Los coeficientes utilizados para cada uno de los modelos y su respectiva resolución ya escalada se ven en la Tabla 2.3. Tan y Le aclaran que se puede obtener mayor eficiencia si se calcula el coeficiente de escalamiento compuesto para cada una de las nuevas arquitecturas generadas, es decir, que B0 genere a B1 con su respectivo coeficiente de escalamiento compuesto, luego sobre B1 se vuelve a calcular dicho coeficiente para generar B2, y así sucesivamente. Sin embargo, realizar esta búsqueda del coeficiente de escalamiento compuesto en modelos más grandes se vuelve muy costoso computacionalmente, por lo que Tan y Le sólo calcularon el coeficiente de escalamiento compuesto sobre EfficientNet-B0, y con dicho coeficiente realizaron el escalamiento de todos los demás modelos.

Para realizar el escalamiento compuesto de cada uno de los modelos B1-B7 se toman los coeficientes calculados y la resolución dada y se aplica el escalamiento a la arquitectura de la

Tabla 2.3: Coeficientes de escalamiento de las distintas variantes de EfficientNet

Modelo	Ancho	Profundo	Resolución
B0	1	1	224
B1	1	1.1	240
B2	1.1	1.2	260
B3	1.2	1.4	300
B4	1.4	1.8	380
B5	1.6	2.2	456
B6	1.8	2.6	528
B7	2	3.1	600

EfficientNet-B0. Para escalar en resolución se redimensiona la imagen de entrada a la resolución dada. Para escalar en profundidad se multiplica el número de repeticiones de las etapas por el coeficiente de escalamiento y se le aplica la función techo, es decir, se toma el entero mayor o igual más cercano. El escalamiento a lo profundo sólo se aplica en los bloques MBConv, es decir, las etapas $2 - 8$. Para aplicar el escalamiento a lo ancho se utiliza un divisor, la EfficientNet-B0 utiliza como divisor el 8. Este divisor se toma del máximo común divisor del número de canales de cada una de las etapas de la EfficientNet-B0. Este divisor funciona como "unidad" al momento de escalar a lo ancho, es decir, el número de canales se multiplica por su coeficiente de escalamiento y se "redondea" al múltiplo de 8 más cercano. Adicionalmente, en el "redondeo" al múltiplo de 8 más cercano, se verifica que el redondeo hacia abajo no supere el $10\,\%$ del valor escalado, esta es una manera de "forzar" que haya escalamiento en las cantidades relativamente pequeñas.

Capítulo 3

Metodología

Se presenta a continuación el apartado metodológico del trabajo de tesis. Se expone el modelo propuesto para la resolución del problema de reconocimiento del modelo de vehículo, así como los métodos utilizados para entrenar dicho modelo. De igual manera, se presenta la validación del modelo, es decir, las medidas cuantitativas que respaldan que el modelo cumple su tarea de manera adecuada.

3.1 Modelo propuesto

Con base en el estado del arte del reconocimiento de vehículos y redes neuronales convolucionales, se propuso un Ensamble de Redes Neuronales Convolucionales (ENN, por sus siglas en inglés - *Ensemble Neural Networks*), más específicamente, un Ensamble de EfficientNets. Con el objetivo de mejorar la precisión de los trabajos previos y pensando a futuro, poder reutilizar el modelo en un sistema de tiempo real, se tomaron los tres modelos más pequeños de la EfficientNet.

El Ensamble de EfficientNets propuesto se construyó utilizando la EfficientNet B0, B1 y B2. La entrada del ENN recibe una imagen y es pasada a cada una de sus partes. La salida del ENN está

dada por el promedio de cada una de las salidas de sus partes, es decir, se promedian las salidas de la EfficientNet-B0, B1 y B2. Hay que recordar que la salida de cada una de las EfficientNet consta de un vector de distribución de probabilidad, por lo que el promedio de éstas genera otro vector de distribución de probabilidad. Cada uno de los modelos de la EfficientNet consta de un tamaño de entrada de diferente resolución, por lo que fue necesario agregar una capa de redimensión a cada uno de ellos en el ENN. El generador de imágenes automáticamente redimensiona las imágenes de la base de datos a una sola resolución dada y para evitar la mayor pérdida de información entre redimensiones, se ajustó al tamaño de entrada de la EfficientNet-B2, 260x260, por lo tanto, la capa de redimensión en B2, no realiza modificaciones. En B0 y B1 la capa de redimensión ajusta la entrada a 224x244 y 240x240 respectivamente. El diagrama del ENN propuesto puede observarse en la Figura 3.1.

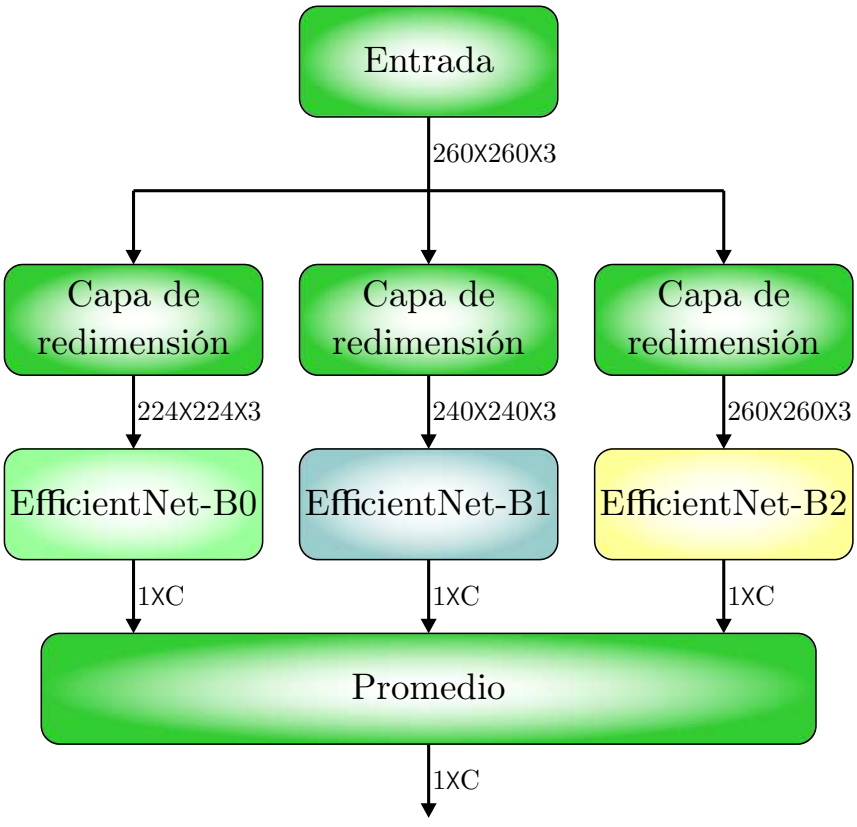


Figura 3.1: Modelo propuesto, Ensamble de EfficientNets.

El ensamble de tres redes neuronales convolucionales resulta conveniente para el problema de reconocimiento de vehículos, ya que las bases de datos existentes para el reconocimiento de vehículos no son muy extensas, por lo que al utilizar un solo modelo muy grande se puede generar *overfitting* más fácilmente. Además, entrenar 3 modelos pequeños resulta más sencillo que entrenar un sólo modelo de mucho mayor tamaño, debido a la cantidad de recursos computacionales que son necesarios y el tiempo de ejecución.

3.2 Entrenamiento del modelo

En esta sección se presenta la base de datos utilizada para entrenar y validar el modelo propuesto, así como las particiones, hiperparámetros seleccionados y demás técnicas utilizadas durante el entrenamiento del mismo.

3.2.1 Base de Datos

Entre las bases de datos públicas encontradas en el estado del arte de reconocimiento de vehículos se escogió la base de datos *CompCars* [49], por ser una de las más desafiantes en categorización fina debido a la cantidad de clases existentes, es decir, modelos de vehículos, y también por ser una de las más extensas. Además, *CompCars* es utilizada en varios trabajos previos [37, 39, 48, 50, 53, 54, 1, 61], lo que permite hacer una comparación directa de los resultados obtenidos.

La base de datos *CompCars* (*Comprehensive Cars*) [49] contiene imágenes de naturaleza web y de cámaras de vigilancia. Los datos de naturaleza web son recopilados de foros de automóviles, sitios web públicos y motores de búsqueda y contiene 163 marcas de vehículos y 1,716 modelos diferentes. Consta de un total de 136,726 imágenes que muestran el auto completo y 27,618 que muestran partes del vehículo. Los datos de cámaras de vigilancia están compuestos por 50,000

imágenes frontales de vehículos. Los modelos de los automóviles están organizados en una jerarquía de tres niveles, primeramente, la marca, luego el modelo y por último el año de manufactura. Cada modelo de vehículo está etiquetado con cinco características, máxima velocidad, desplazamiento (tamaño volumétrico del motor), número de puertas, número de asientos y tipo de vehículo. Adicionalmente, cada imagen viene etiquetada con el punto de vista del vehículo que consta de cinco diferentes ángulos, frontal, trasera, lateral, frontal-lateral y trasera-lateral.

CompCars proporciona un subconjunto de 30,955 imágenes de naturaleza web dedicado al problema de clasificación de grano fino que consta de 431 modelos de vehículos y 75 marcas. De igual manera, divide dicho subconjunto en dos partes, un conjunto de entrenamiento de 16,016 imágenes y un conjunto de validación de 14,939 imágenes.

(a) Mini Countryman (b) Nissan Tiida (c) Hyundai i30

(d) Chrysler 300C (e) Land-Rover Discovery (f) Kia Forte

Figura 3.2: Ejemplos de imágenes de naturaleza web de la base de datos *CompCars*.

3.2.2 Entrenamiento

Para entrenar el ENN de EfficientNets se entrenó por separado cada una de sus partes, es decir, la EfficientNet-B0, B1 y B2. Las particiones utilizadas para el entrenamiento y validación son las mismas que propone *CompCars* para clasificación de grano fino, es decir, los subconjuntos de 16,016 y 14,939 imágenes respectivamente. Al utilizar las particiones propuestas por *CompCars* se puede realizar una comparación de los resultados obtenidos con los de otros trabajos.

Los diferentes modelos de EfficientNet, B0, B1 y B2, son de distinto tamaño, por lo que se utilizó distinto número de épocas de entrenamiento para cada uno, sin embargo, la estrategia de entrenamiento fue la misma para todos los modelos. La estrategia de entrenamiento consistió en utilizar el optimizador Nadam para las primeras épocas y luego dar paso al optimizador NAG. Se utilizó el optimizador Nadam debido a que es de los optimizadores más recientemente propuestos y que ha mostrado buenos resultados. El optimizador Nadam tiene buena velocidad de convergencia y evita caer en mínimos locales que estén alejados del valor del mínimo global, sin embargo, su comportamiento es algo "errático", ocasionando que en épocas finales difícilmente se llegue al mínimo deseado. Debido al anterior inconveniente del optimizador Nadam, se optó por utilizar el optimizador NAG para las épocas finales. NAG es un optimizador que mantiene la habilidad de evitar caer en los mínimos locales que estén alejados del mínimo global pero es mucho menos "errático" que Nadam, lo que supone poder acercarse más suavemente al mínimo deseado.

La EfficientNet-B0 se entrenó utilizando el optimizador Nadam por 20 épocas y posteriormente 40 épocas más utilizando el optimizador NAG. Se utilizaron los hiperparámetros por defecto del optimizador Nadam, que son los propuestos por Dozat [74]. Para los hiperparámetros del optimizador NAG se utilizó un *momentum* de 0.9 y se asignó un esquema de reducción a la tasa de aprendizaje. El esquema de reducción utilizado ajusta el valor de la tasa de aprendizaje cada 10 épocas, tomando como valor inicial el 0.001 y pasando por los valores 0.0005, 0.0001 y 0.00005.

Esta estrategia es utilizada para acercarse más suavemente al mínimo.

La EfficientNet-B1 y B2 se entrenaron con 30 y 40 épocas respectivamente utilizando el optimizador Nadam con los hiperparámetros por defecto, y 40 épocas adicionales con el optimizador NAG con *momentum* de 0.9. Tanto para la EfficientNet-B1 y B2 se utilizó el mismo esquema de reducción de la tasa de aprendizaje en NAG utilizado con la EfficientNet-B0.

Adicionalmente a las técnicas de entrenamiento mencionadas, se utilizó también la técnica de *data augmentation*. Se utilizaron tres distintas transformaciones de manera aleatoria durante el entrenamiento, es decir, por cada *batch* de entrenamiento se aplican nuevas transformaciones. Se utilizó rotación de la imagen dentro del rango $\pm 5^{\circ}$, recorte al $90\,\%$ y reflejo horizontal. La rotación genera vacíos en la imagen, estos vacíos fueron llenados con el valor cero. Por último, además del *data augmentation*, las imágenes son sometidas a un preprocesamiento de reducción de media. La media fue calculada a través de la suma de cada uno de los canales de color (RGB) de todas las imágenes de entrenamiento dividida entre la resolución de las imágenes y el número de muestras. Los valores que se obtuvieron con el cálculo de la media fueron $[119.63908795908532, 122.180521647965, 125.18300377501365]$. El preprocesamiento de las imágenes es aplicado tanto en el conjunto de entrenamiento como en el de validación, mientras que el *data augmentation* sólo es aplicado al conjunto de entrenamiento.

En las Figuras 3.3, 3.4 y 3.5 se observan las gráficas de la función de pérdida durante el entrenamiento de la EfficientNet-B0, B1 y B2, respectivamente. Se puede apreciar el comportamiento "errático" durante las primeras épocas de entrenamiento, debido al optimizador Nadam, y cómo la gráfica se comporta de manera más suave cuando se utiliza el optimizador NAG en las épocas finales. Las curvas azules representan las gráficas de la función de pérdida utilizando los datos de entrenamiento, que corresponde al conjunto de 16,016 imágenes, y las curvas rojas utilizando los datos de validación, correspondientes al conjunto de 14,939 imágenes.

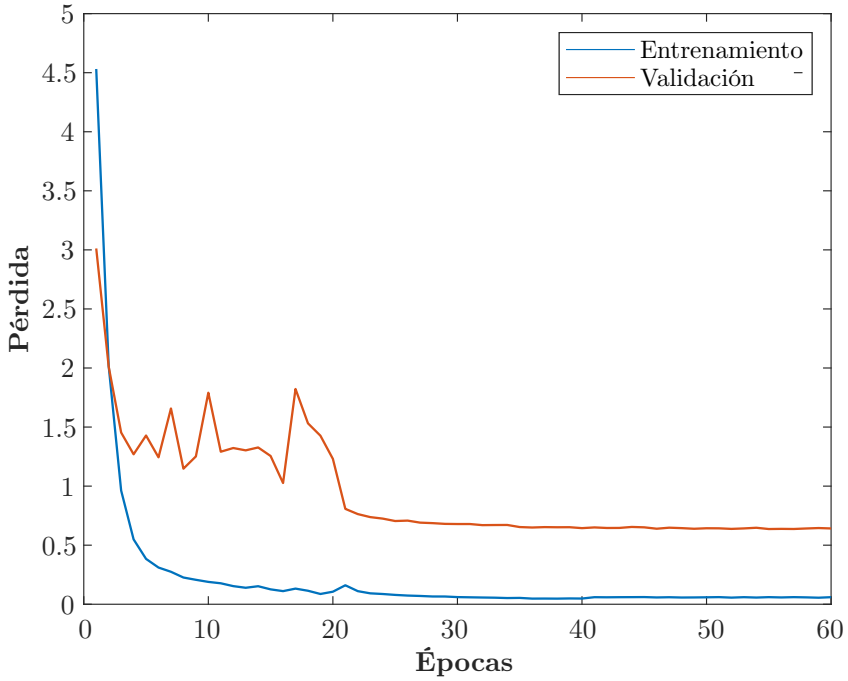


Figura 3.3: Gráfica de la función de pérdida durante el entrenamiento de la EfficientNet-B0 en los conjuntos de entrenamiento y validación.

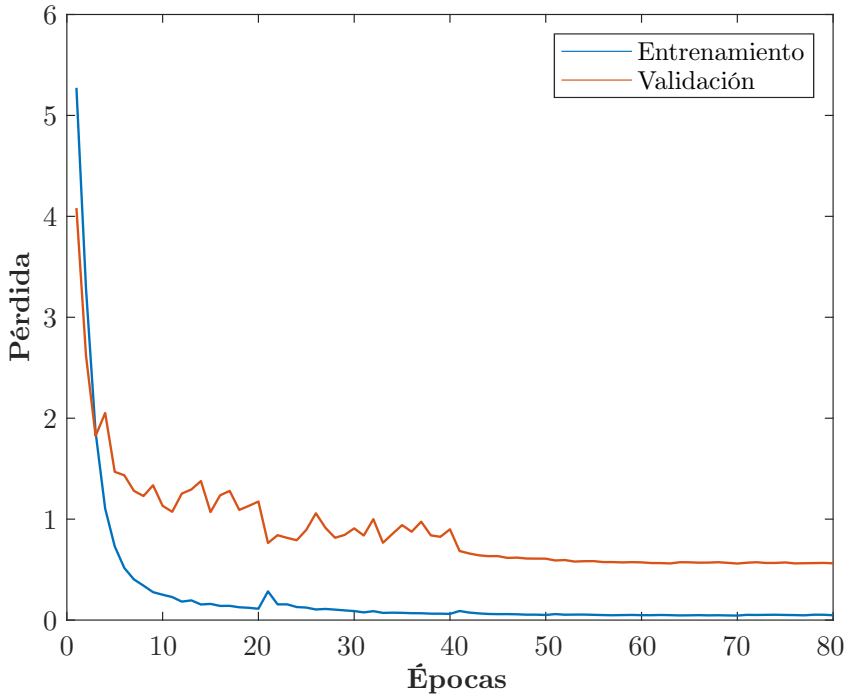


Figura 3.5: Gráfica de la función de pérdida durante el entrenamiento de la EfficientNet-B2 en los conjuntos de entrenamiento y validación.

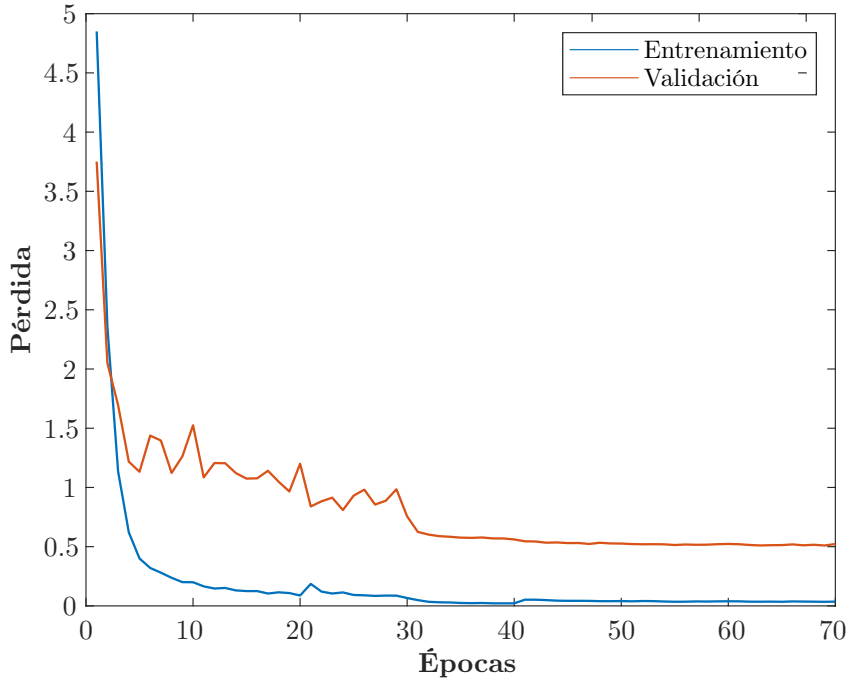


Figura 3.4: Gráfica de la función de pérdida durante el entrenamiento de la EfficientNet-B1 en los conjuntos de entrenamiento y validación.

3.2.3 Validación

Para validar un modelo existen varias métricas, las más comunes en problemas de clasificación multiclase son la exactitud, precisión y *recall*. En la lengua española exactitud y precisión pueden ser vistos como sinónimos, sin embargo, en el área de aprendizaje máquina tienen un significado diferente, y es más fácil diferenciarlos por sus respectivos nombres en inglés, *accuracy* y *precision*. Aunque todas estas métricas son comunes en clasificación multiclase, la *precision* y el *recall* son más utilizadas para problemas binarios o clasificación multiclase con un número bajo de clases. La *precision* y el *recall* suelen ser usados en problemas donde la relación entre falsos positivos o falsos negativos resulta relevante, como lo es en el caso de detección de enfermedades. En el estado del arte de reconocimiento de vehículos es común ver que los autores sólo usen la métrica *accuracy*.

La *accuracy* es una métrica que indica que tan acertado es el modelo, y está dada por la razón del

número de aciertos entre el número total de muestras. Esta métrica también es llamada *accuracy* top-1, y la razón de esto es porque se toma sólo la clase de mayor valor predicha por el modelo para calcular la *accuracy*. Existe otra variación de la *accuracy* conocida como *accuracy* top-N, en donde se toman los primeros N valores predichos por el modelo para calcular la *accuracy*, es decir, si alguno de los N valores mayores predichos por el modelo corresponde a la clase correcta se toma como un acierto. La *accuracy* top-5 es la más usada, además de la top-1, para problemas de clasificación con gran cantidad de clases, así como en el estado del arte de reconocimiento de vehículos. Se utilizó la métrica *accuracy* top-1 y top-5 para validar el modelo propuesto, así como para compararlo con otros trabajos.

Capítulo 4

Resultados

El modelo propuesto logró obtener una *accuracy* top-1 de 94.13 % y top-5 de 98.47 % en clasificación fina de vehículos a nivel modelo utilizando las particiones antes mencionadas de CompCars. Se compararon los resultados obtenidos con los de otras arquitecturas CNN y también con otros modelos del estado del arte de reconocimiento de vehículos que utilizan la misma base de datos con las mismas particiones. En la Tabla 4.1 podemos ver dicha comparación. Se observa que el modelo propuesto de ensamble de EffcientNets supera el estado del arte en reconocimiento de vehículos. La arquitectura EfficientNet-B4 es la que más se le acerca en términos de accuracy top-1, sin embargo, el ensamble de EfficientNets es más eficiente, ya que consiguió mejor accuracy con un poco más de parámetros, pero con menos FLOPS. La EfficientNet-B0, B1 y B2 tienen 5.3M, 7.8M y 9.2M parámetros respectivamente, por lo que el ensamble de ellas tres hace un total de 22.3M de parámetros. De igual manera, la EfficientNet-B0, B1 y B2 tienen 0.39B, 0.70B y 1.0B de FLOPS respectivamente, por lo que el ensamble de ellas tres hace un total de 2.09B FLOPS. La EfficietNet-B4 consta de 19M de parámetros y un total de 4.2B de FLOPS.

Se obtuvo la accuracy a nivel Marca para compararla con el trabajo de Chen et al. [1]. El modelo propuesto alcanzó el 96.67 %., contra 95.14 % del modelo de Chen et al.

Tabla 4.1: Comparación de resultados de categorización fina usando la base de datos CompCars.

Modelo	*Accuracy* Top-1	*Accuracy* Top-5
MobileNetv2 [4]	42.5	66.9
MobileNetv3 Large [78]	39.3	61.9
FM-CNN [2]	91.0	97.8
CNN+MS+MT [1]	91.0	97.8
EfficientNet-B0 [3]	87.4	97.2
EfficientNet-B1 [3]	89.6	97.7
EfficientNet-B2 [3]	90.0	97.7
EfficientNet-B4 [3]	93.1	**98.7**
Modelo propuesto	**94.1**	98.5

Se analizó el nivel de precisión por punto de vista del vehículo, usando los 5 diferentes tipos de vista que proporciona CompCars, fontal, trasera, lateral, frontal-lateral y trasera lateral. En la Tabla 4.2 se observa el resumen de las precisiones por punto de vista. Se aprecia que el punto de vista trasera-lateral obtiene el mayor índice de precisión con 96.42 %, mientras que el punto de vista lateral es el que obtiene el menor índice de precisión con 91.43 %.

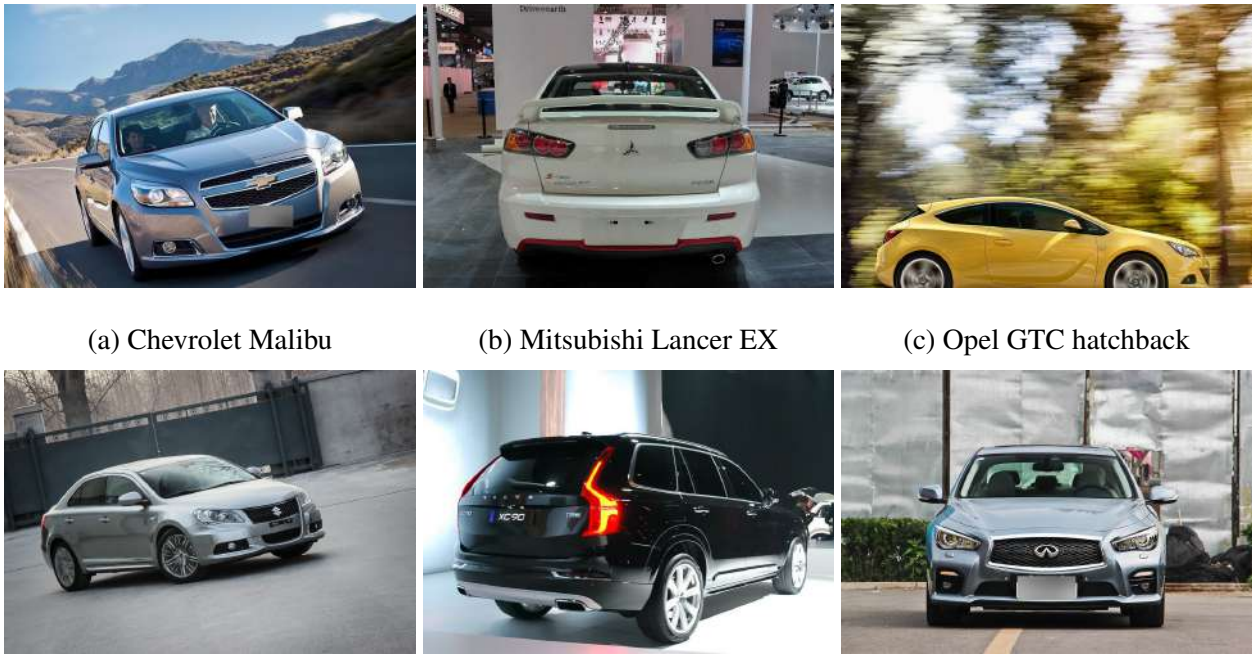
Tabla 4.2: Análisis de precisión de los puntos de vista en la base de datos CompCars.

		Frontal	Trasera	Lateral	Frontal-lateral	Trasera-lateral
Accuracy a nivel:	Marca	97.56	96.28	93.80	97.42	97.56
	Modelo	93.05	94.27	91.43	94.54	96.42

Se presenta en la Figura 4.1 ejemplos de imágenes correctamente clasificadas. En general, se aprecia que el modelo es robusto en cuanto a iluminación, ya que en muchas imágenes el fondo

puede resultar confuso debido a los cambios de luz y de color. En la Figura 4.2 vemos ejemplos de imágenes mal clasificadas a nivel modelo, pero correctamente clasificadas a nivel marca. Este tipo de errores son difíciles de resolver, ya que muchos modelos de una misma marca son bastante similares, y algunos prácticamente son el mismo modelo con un cambio prácticamente imperceptible, incluso para un ojo humano conocedor. Por ejemplo, en la Figura 4.2-(b) vemos una confusión del modelo 3 series, con el 3 series GT, los cuales en apariencia física son casi iguales. Sucede lo mismo con la Figura 4.2-(e), en donde la confusión recae sobre una versión distinta del mismo modelo, en este caso el Regal es confundido con el Regal GS. Otro tipo de error común sucede en modelos que prácticamente son iguales en determinado punto de vista. Por ejemplo, en la Figura 4.2-(f) el Peugeot 207 hatchback es mal clasificado como un Peugeot 207 sedan, estos dos modelos resultan idénticos desde el punto de vista frontal.

En la Figura 4.3 observamos ejemplos que son mal clasificados tanto a nivel marca como a



(a) Chevrolet Malibu (b) Mitsubishi Lancer EX (c) Opel GTC hatchback

(d) Suzuki Kazishi (e) Volvo XC90 abroad (f) Infiniti Q50

Figura 4.1: Ejemplos clasificados correctamente.

(a) Audi A5 hatchback - Audi A7

(b) BMW 3 series - BMW 3 series GT

(c) Benz C Class - Benz S Class

(d) Honda Accord - Honda Civic

(e) Buick Regal - Buick Regal GS

(f) Peugeot 207 hatchback - Peugeot 207 sedan

Figura 4.2: Ejemplos mal clasificados a nivel modelo pero correctamente a nivel marca. De cada inciso el modelo de la izquierda es la imagen de entrada y el de la derecha es un ejemplo del modelo de vehículo predicho.

nivel modelo. Muchos de estas imágenes mal clasificadas corresponden a modelos de vehículos que son bastante parecidos, a pesar de ser de distintas marcas. Por ejemplo, los pares de imágenes de la Figura 4.3-(a),(b),(c),(d), resultan ser modelos de vehículos muy parecidos a simple vista, y se necesita observar a detalle para poder apreciar sus diferencias. Existe otro tipo de imágenes mal clasificadas que corresponden a modelos bastante diferentes, véase Figura 4.3-(e),(f). Este tipo de imágenes mal clasificadas lo que tienen en común es que el vehículo no abarca gran área de la imagen.


(a) BYD L3 - Hyundai Sonata 8


(b) Besturn B90 - JAC Heyue


(c) VW Gran Lavida - Audi A3 Hatchback


(d) Ford Classic Focus Sedan - Skoda Rapid


(e) FIAT 500 - Toyota 86


(f) Jaguar XF - Nissan GT-R

Figura 4.3: Ejemplos mal clasificados tanto a nivel modelo como a nivel marca. De cada inciso el modelo de la izquierda es la imagen de entrada y el de la derecha es un ejemplo del modelo de vehículo predicho.

Capítulo 5

Conclusiones

En la presente tesis se trabajó en el desarrollo de un modelo basado en redes neuronales convolucionales que resolviera el problema de reconocimiento de vehículos utilizando imágenes de múltiples vistas con la finalidad de mejorar los trabajos del estado del arte.

Inicialmente se trabajó en la investigación de los diversos modelos propuestos para el reconocimiento de vehículos para poder obtener una idea de los alcances del trabajo y planterse unos objetivos realistas. Se investigó de igual manera el estado del arte de las redes neuronales convolucionales, ya que éstas han mostrado los mejores resultados en tareas de clasificación de imágenes. Esta investigación nos llevó a la conclusión de utilizar un modelo basado en redes neuronales convolucionales de alta eficiencia, denominadas como Efficient-Net.

El conocimiento adquirido en la fase de investigación del estado del arte nos llevó al planteamiento del modelo propuesto, un ensamble de redes neuronales convolucionales, más específicamente, un Ensamble de EfficientNets. Debido a que el presente trabajo buscaba obtener un modelo capaz de realizar la tarea en tiempo-real, se propuso utilizar las tres versiones más pequeñas de la EfficientNet, la B0, B1 y B2. El Ensamble de EfficientNets logró superar los trabajos del estado

del arte en reconocimiento de vehículos, obteniendo una *accuracy* de 94.1 % en la base de datos CompCars.

A pesar de haber superado el mejor resultado del estado del arte, un análisis de los resultados, específicamente, observando los ejemplos que el modelo no fue capaz de clasificar correctamente, se lograron encontrar algunas debilidades en el modelo propuesto. El modelo no fue capaz de clasificar correctamente algunos modelos que presentan características muy parecidas, este problema podría ser resuelto ampliando las técnicas del aumento de datos para dejar que el modelo aprenda más a detalle, o en su caso, utilizar una base de datos que contenga más ejemplos por modelo. Otra limitante sucede con las imágenes en las que el vehículo no ocupa una gran área, esta debilidad podría ser resulta utilizando un detector de vehículos como fase previa a la clasificación, lo que permitiría que el modelo aprenda más específicamente sobre las características del vehículo sin dar tanta atención al ruido de fondo que pudiese haber en la imagen. Por último, un análisis de los resultados por punto de vista nos deja ver que se obtienen mejores resultados en el punto de vista trasera-lateral, esto resulta relevante para los futuros sistemas inteligentes que se pudiesen implementar, tratando de que las cámaras obtengan imágenes desde dicho punto de vista. Es importante aclarar que la base de datos empleada, CompCars, no está correctamente balanceada, es decir, existen mayor y menor número de imágenes para determinadas clases de vehículos, así como para los puntos de vista. Este problema deja abierta la necesidad de elaborar una nueva base de datos más balanceada para poder realizar un mejor análisis de los resultados.

Por último, se puede decir que se ha conseguido cumplir con los objetivos de esta tesis, ya que se logró implementar un modelo que superara los resultados del estado del arte de reconocimiento de vehículos, y así mismo, se resaltan los puntos que pudieren ser mejorados en trabajos a futuro.

Bibliografía

[1] B. Hu, J.-H. Lai, and C.-C. Guo, "Location-aware fine-grained vehicle type recognition using multi-task deep networks," *Neurocomputing*, vol. 243, pp. 60–68, 2017. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0925231217304691

[2] Z. Chen, C. Ying, C. Lin, S. Liu, and W. Li, "Multi-view vehicle type recognition with feedback-enhancement multi-branch cnns," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 29, no. 9, pp. 2590–2599, 2019.

[3] M. Tan and Q. V. Le, "Efficientnet: Rethinking model scaling for convolutional neural networks," 2019.

[4] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "Mobilenetv2: Inverted residuals and linear bottlenecks," 2018.

[5] J. Hu, L. Shen, S. Albanie, G. Sun, and E. Wu, "Squeeze-and-excitation networks," 2017.

[6] Y. Zhou, L. Liu, L. Shao, and M. Mellor, "Fast automatic vehicle annotation for urban traffic surveillance," *IEEE Transactions on Intelligent Transportation Systems*, vol. 19, no. 6, pp. 1973–1984, June 2018.

[7] Y. Chen, B. Wu, and C. Fan, "Real-time vision-based multiple vehicle detection and tracking for nighttime traffic surveillance," in *2009 IEEE International Conference on Systems, Man and Cybernetics*, Oct 2009, pp. 3352–3358.

[8] Z. Liu, M. Sang, and L. Wu, "Developing history and trends of chinese expressway's toll collection techniques," in *2017 2nd IEEE International Conference on Intelligent Transportation Engineering (ICITE)*, Sep. 2017, pp. 219–222.

[9] T. L. McDaniel, "The (r)evolution of toll-collection technology," *IEEE Potentials*, vol. 34, no. 5, pp. 34–39, Sep. 2015.

[10] A. Athira, S. Lekshmi, P. Vijayan, and B. Kurian, "Smart parking system based on optical character recognition," in *2019 3rd International Conference on Trends in Electronics and Informatics (ICOEI)*, April 2019, pp. 1184–1188.

[11] R. Grodi, D. B. Rawat, and F. Rios-Gutierrez, "Smart parking: Parking occupancy monitoring and visualization system for smart cities," in *SoutheastCon 2016*, March 2016, pp. 1–5.

[12] D. Yang, L. Zhu, Y. Liu, D. Wu, and B. Ran, "A novel car-following control model combining machine learning and kinematics models for automated vehicles," *IEEE Transactions on Intelligent Transportation Systems*, vol. 20, no. 6, pp. 1991–2000, June 2019.

[13] X. Feng, X. Ling, H. Zheng, Z. Chen, and Y. Xu, "Adaptive multi-kernel svm with spatial–temporal correlation for short-term traffic flow prediction," *IEEE Transactions on Intelligent Transportation Systems*, vol. 20, no. 6, pp. 2001–2013, June 2019.

[14] Z. Tian, Y. Cai, S. Huang, F. Hu, Y. Li, and M. Cen, "Vehicle tracking system for intelligent and connected vehicle based on radar and v2v fusion," in *2018 Chinese Control And Decision Conference (CCDC)*, June 2018, pp. 6598–6603.

[15] H. Muslim and M. Itoh, "Effects of human understanding of automation abilities on driver performance and acceptance of lane change collision avoidance systems," *IEEE Transactions on Intelligent Transportation Systems*, vol. 20, no. 6, pp. 2014–2024, June 2019.

[16] X. Li, K. Wang, Y. Tian, L. Yan, F. Deng, and F. Wang, "The paralleleye dataset: A large

collection of virtual images for traffic vision research," *IEEE Transactions on Intelligent Transportation Systems*, vol. 20, no. 6, pp. 2072–2084, June 2019.

[17] X. Ke, L. Shi, W. Guo, and D. Chen, "Multi-dimensional traffic congestion detection based on fusion of visual features and convolutional neural network," *IEEE Transactions on Intelligent Transportation Systems*, vol. 20, no. 6, pp. 2157–2170, June 2019.

[18] X. Zhang, Y.-H. Yang, Z. Han, H. Wang, and C. Gao, "Object class detection: A survey," *ACM Comput. Surv.*, vol. 46, no. 1, pp. 10:1–10:53, Jul. 2013. [Online]. Available: http://doi.acm.org/10.1145/2522968.2522978

[19] H. Cheng, J. Shan, W. Ju, Y. Guo, and L. Zhang, "Automated breast cancer detection and classification using ultrasound images: A survey," *Pattern Recognition*, vol. 43, no. 1, pp. 299 – 317, 2010. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0031320309002027

[20] W. Zhao, R. Chellappa, P. J. Phillips, and A. Rosenfeld, "Face recognition: A literature survey," *ACM Comput. Surv.*, vol. 35, no. 4, pp. 399–458, Dec. 2003. [Online]. Available: http://doi.acm.org/10.1145/954339.954342

[21] B. Fasel and J. Luettin, "Automatic facial expression analysis: a survey," *Pattern Recognition*, vol. 36, no. 1, pp. 259 – 275, 2003. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0031320302000523

[22] S. Liu, G. Tian, and Y. Xu, "A novel scene classification model combining resnet based transfer learning and data augmentation with a filter," *Neurocomputing*, vol. 338, pp. 191 – 206, 2019. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0925231219301833

[23] L. Guo, J. Song, X. rui Li, H. Huang, J. jing Du, Y. chao He, and C. zhuang Wang,

"Haze image classification method based on alexnet network transfer model," *Journal of Physics: Conference Series*, vol. 1176, p. 032011, mar 2019. [Online]. Available: https://doi.org/10.1088%2F1742-6596%2F1176%2F3%2F032011

[24] X. Yang, Y. Ye, X. Li, R. Y. K. Lau, X. Zhang, and X. Huang, "Hyperspectral image classification with deep learning models," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 56, no. 9, pp. 5408–5423, Sep. 2018.

[25] P. Tang, H. Wang, and S. Kwong, "G-ms2f: Googlenet based multi-stage feature fusion of deep cnn for scene recognition," *Neurocomputing*, vol. 225, pp. 188 – 197, 2017. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0925231216314047

[26] T. Chan, K. Jia, S. Gao, J. Lu, Z. Zeng, and Y. Ma, "Pcanet: A simple deep learning baseline for image classification?" *IEEE Transactions on Image Processing*, vol. 24, no. 12, pp. 5017–5032, Dec 2015.

[27] D. Mishkin, N. Sergievskiy, and J. Matas, "Systematic evaluation of convolution neural network advances on the imagenet," *Computer Vision and Image Understanding*, vol. 161, pp. 11 – 19, 2017. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1077314217300814

[28] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Commun. ACM*, vol. 60, no. 6, pp. 84–90, May 2017. [Online]. Available: http://doi.acm.org/10.1145/3065386

[29] A. Nag, D. Miller, A. Brown, and K. Sullivan, "A system for vehicle recognition in video based on sift features, mixture models, and support vector machines," vol. 6560, 2007, cited By 1. [Online]. Available: https://www.scopus.com/inward/record.uri?eid=2-s2.0-35948991551&doi=10.1117%

2f12.723746&partnerID=40&md5=4ff0c52d52931f7afead47047cbecfc1

[30] A. Psyllos, C. Anagnostopoulos, and E. Kayafas, "Sift-based measurements for vehicle model recognition," vol. 1, 2009, pp. 84–89, cited By 6. [Online]. Available: https://www.scopus.com/inward/record.uri?eid=2-s2.0-84871533908&partnerID=40&md5=a7491a4f5c97d4cc328c1652997fc656

[31] A. P. Psyllos, C. E. Anagnostopoulos, and E. Kayafas, "Vehicle logo recognition using a sift-based enhanced matching scheme," *IEEE Transactions on Intelligent Transportation Systems*, vol. 11, no. 2, pp. 322–328, June 2010.

[32] Z. Tang, Y. Xian, and J. Chen, "Research of vehicle recognition method under surf feature and bayesian model," in *2013 Sixth International Symposium on Computational Intelligence and Design*, vol. 1, Oct 2013, pp. 252–256.

[33] S. Wang, L. Liu, and X. Xu, "Vehicle logo recognition based on local feature descriptor," *Applied Mechanics and Materials*, vol. 263-266, no. PART 1, pp. 2418–2421, 2013, cited By 3. [Online]. Available: https://www.scopus.com/inward/record.uri?eid=2-s2.0-84872435043&doi=10.4028%2fwww.scientific.net%2fAMM.263-266.2418&partnerID=40&md5=385f0eacd3d9f59d5786a7d0f3a6c722

[34] X. Li and X. Guo, "A hog feature and svm based method for forward vehicle detection with single camera," in *2013 5th International Conference on Intelligent Human-Machine Systems and Cybernetics*, vol. 1, Aug 2013, pp. 263–266.

[35] D. F. Llorca, R. Arroyo, and M. A. Sotelo, "Vehicle logo recognition in traffic images using hog features and svm," in *16th International IEEE Conference on Intelligent Transportation Systems (ITSC 2013)*, Oct 2013, pp. 2229–2234.

[36] S. Dewen, L. Bin, and C. Jing, "Convolutional neural network and the recognition of

vehicle types," *NeuroQuantology*, vol. 16, no. 6, pp. 720–727, 2018. [Online]. Available: http://10.14704/nq.2018.16.6.1641

[37] J. Wang, H. Zheng, Y. Huang, and X. Ding, "Vehicle type recognition in surveillance images from labeled web-nature data using deep transfer learning," *IEEE Transactions on Intelligent Transportation Systems*, vol. 19, no. 9, pp. 2913–2922, 2018.

[38] M. N. Roecker, Y. M. G. Costa, J. L. R. Almeida, and G. H. G. Matsushita, "Automatic vehicle type classification with convolutional neural networks," in *2018 25th International Conference on Systems, Signals and Image Processing (IWSSIP)*, Conference Proceedings, pp. 1–5.

[39] W. Chen, Q. Sun, J. Wang, J. Dong, and C. Xu, "A novel model based on adaboost and deep cnn for vehicle classification," *IEEE Access*, vol. 6, pp. 60 445–60 455, 2018.

[40] L. Zhuo, L. Jiang, Z. Zhu, J. Li, J. Zhang, and H. Long, "Vehicle classification for large-scale traffic surveillance videos using convolutional neural networks," *Machine Vision and Applications*, vol. 28, no. 7, pp. 793–802, 2017. [Online]. Available: https://doi.org/10.1007/s00138-017-0846-2

[41] J. Hsieh, L. Chen, and D. Chen, "Symmetrical SURF and its applications to vehicle detection and vehicle make and model recognition," *IEEE Transactions on Intelligent Transportation Systems*, vol. 15, no. 1, pp. 6–20, 2014.

[42] L.-C. Chen, J.-W. Hsieh, Y. Yan, and D.-Y. Chen, "Vehicle make and model recognition using sparse representation and symmetrical SURFs," vol. 48, no. 6, pp. 1979–1998, 2015. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0031320315000035

[43] H. Bay, A. Ess, T. Tuytelaars, and L. V. Gool, "Speeded-up robust features (SURF)," *Computer Vision and Image Understanding*, vol. 110, no. 3, pp. 346 – 359,

2008, similarity Matching in Computer Vision and Multimedia. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1077314207001555

[44] A. J. Siddiqui, A. Mammeri, and A. Boukerche, "Real-time vehicle make and model recognition based on a bag of SURF features," *IEEE Transactions on Intelligent Transportation Systems*, vol. 17, no. 11, pp. 3205–3219, 2016.

[45] H. He, Z. Shao, and J. Tan, "Recognition of car makes and models from a single traffic-camera image," *IEEE Transactions on Intelligent Transportation Systems*, vol. 16, no. 6, pp. 3182–3192, 2015.

[46] Y. Huang, R. Wu, Y. Sun, W. Wang, and X. Ding, "Vehicle logo recognition system based on convolutional neural networks with a pretraining strategy," *IEEE Transactions on Intelligent Transportation Systems*, vol. 16, no. 4, pp. 1951–1960, 2015.

[47] Y. Gao and H. J. Lee, "Local tiled deep networks for recognition of vehicle make and model," *Sensors (Basel, Switzerland)*, vol. 16, no. 2, pp. 226–226, 2016. [Online]. Available: http://search.ebscohost.com/login.aspx?direct=true&db=mdc&AN=26875983Œ=es&site=ehost-live

[48] M. Biglari, A. Soleimani, and H. Hassanpour, "Part-based recognition of vehicle make and model," *IET Image Processing*, vol. 11, no. 7, pp. 483–491, 2017.

[49] L. Yang, P. Luo, C. C. Loy, and X. Tang, "A large-scale car dataset for fine-grained categorization and verification," in *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Conference Proceedings, pp. 3973–3981.

[50] J. Fang, Y. Zhou, Y. Yu, and S. Du, "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, 2017.

[51] S. Yu, Y. Wu, W. Li, Z. Song, and W. Zeng, "A model for fine-grained vehicle classification based on deep learning," *Machine Learning and Signal Processing for Big Multimedia Analysis*, vol. 257, pp. 97–103, 2017. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S092523121730156X

[52] L. Lu and H. Huang, "A hierarchical scheme for vehicle make and model recognition from frontal images of vehicles," *IEEE Transactions on Intelligent Transportation Systems*, pp. 1–13, 2018.

[53] M. Biglari, A. Soleimani, and H. Hassanpour, "A cascaded part-based system for fine-grained vehicle classification," *IEEE Transactions on Intelligent Transportation Systems*, vol. 19, no. 1, pp. 273–283, 2018.

[54] F. C. Soon, H. Y. Khaw, J. H. Chuah, and J. Kanesan, "Pcanet-based convolutional neural network architecture for a vehicle model recognition system," *IEEE Transactions on Intelligent Transportation Systems*, pp. 1–11, 2018.

[55] Z. Xuze, N. Shengsuo, and T. Huang, "A CNN vehicle recognition algorithm based on reinforcement learning error and error-prone samples," *IOP Conference Series: Earth and Environmental Science*, vol. 153, no. 3, p. 032052, 2018. [Online]. Available: http://stacks.iop.org/1755-1315/153/i=3/a=032052

[56] Z. Dong, Y. Wu, M. Pei, and Y. Jia, "Vehicle type classification using a semisupervised convolutional neural network," *IEEE Transactions on Intelligent Transportation Systems*, vol. 16, no. 4, pp. 2247–2256, 2015.

[57] Y. Peng, J. S. Jin, S. Luo, M. Xu, and Y. Cui, "Vehicle type classification using pca with self-clustering," in *2012 IEEE International Conference on Multimedia and Expo Workshops*, July 2012, pp. 384–389.

[58] C. Szegedy, Wei Liu, Yangqing Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2015, pp. 1–9.

[59] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, "ImageNet Large Scale Visual Recognition Challenge," *International Journal of Computer Vision (IJCV)*, vol. 115, no. 3, pp. 211–252, 2015.

[60] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *CoRR*, vol. abs/1409.1556, 2014. [Online]. Available: http://arxiv.org/abs/1409.1556

[61] L. Xue, X. Zhong, R. Wang, J. Yang, and M. Hu, "Low - resolution vehicle recognition based on deep feature fusion," *Multimedia Tools and Applications*, vol. 77, no. 20, pp. 27 617–27 639, 2018. [Online]. Available: https://doi.org/10.1007/s11042-018-5940-6

[62] J. Krause, M. Stark, J. Deng, and L. Fei-Fei, "3D object representations for fine-grained categorization," in *4th International IEEE Workshop on 3D Representation and Recognition (3dRR-13)*, Sydney, Australia, 2013.

[63] H. S. Bharadwaj, S. Biswas, and K. R. Ramakrishnan, "A large scale dataset for classification of vehicles in urban traffic scenes," in *Proceedings of the Tenth Indian Conference on Computer Vision, Graphics and Image Processing*, ser. ICVGIP '16. New York, NY, USA: ACM, 2016, pp. 83:1–83:8. [Online]. Available: http://doi.acm.org/10.1145/3009977.3010040

[64] L. Hua, W. Xu, T. Wang, R. Ma, and B. Xu, "Vehicle recognition using improved sift and multi-view model," *Journal of Xi'an Jiaotong University*, vol. 47, no. 4, p. 92–99, Apr 2013.

[65] Y. Xu, G. Yu, Y. Wang, X. Wu, and Y. Ma, "A hybrid vehicle detection method based on viola-jones and hog + svm from uav images," *Sensors*, vol. 16, no. 8, 2016. [Online]. Available: http://www.mdpi.com/1424-8220/16/8/1325

[66] J. Hsieh, L. Chen, D. Chen, and S. Cheng, "Vehicle make and model recognition using symmetrical surf," in *2013 10th IEEE International Conference on Advanced Video and Signal Based Surveillance*, Aug 2013, pp. 472–477.

[67] P. Ramachandran, B. Zoph, and Q. V. Le, "Searching for activation functions," *CoRR*, vol. abs/1710.05941, 2017. [Online]. Available: http://arxiv.org/abs/1710.05941

[68] R. D. Reed and R. J. Marks, *Classical Optimization Techniques*. The MIT Press, 1999, p. 155–156.

[69] N. Qian, "On the momentum term in gradient descent learning algorithms," *Neural Networks*, vol. 12, no. 1, pp. 145 – 151, 1999. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0893608098001166

[70] Y. NESTEROV, "A method for unconstrained convex minimization problem with the rate of convergence O(1/kˆ2)," *Doklady AN USSR*, vol. 269, pp. 543–547, 1983. [Online]. Available: https://ci.nii.ac.jp/naid/20001173129/en/

[71] J. Duchi, E. Hazan, and Y. Singer, "Adaptive subgradient methods for online learning and stochastic optimization," *J. Mach. Learn. Res.*, vol. 12, no. null, p. 2121–2159, Jul. 2011.

[72] M. D. Zeiler, "Adadelta: An adaptive learning rate method," 2012.

[73] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2014.

[74] T. Dozat, "Incorporating nesterov momentum into adam," 2016.

[75] M. Tan, B. Chen, R. Pang, V. Vasudevan, M. Sandler, A. Howard, and Q. V. Le, "Mnasnet: Platform-aware neural architecture search for mobile," 2018.

[76] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," 2015.

[77] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," 2017.

[78] A. Howard, M. Sandler, G. Chu, L. Chen, B. Chen, M. Tan, W. Wang, Y. Zhu, R. Pang, V. Vasudevan, Q. V. Le, and H. Adam, "Searching for mobilenetv3," *CoRR*, vol. abs/1905.02244, 2019. [Online]. Available: http://arxiv.org/abs/1905.02244