UNIVERSIDAD AUTÓNOMA DE MADRID
ESCUELA POLITÉCNICA SUPERIOR

Grado en Tecnologías y Servicios de Telecomunicación

TRABAJO FIN DE GRADO

SEGMENTACIÓN OBJETO-FONDO MEDIANTE REDES CONVOLUCIONALES

Autor: Raúl Arcos Serrano
Tutor: Álvaro García Martín
Ponente: José María Martínez Sánchez

JUNIO 2021

SEGMENTACIÓN OBJETO-FONDO MEDIANTE REDES CONVOLUCIONALES

Autor: Raúl Arcos Serrano
Tutor: Álvaro García Martín
Ponente: José María Martínez Sánchez

Video Processing and Understanding Lab
Dpto. Tecnología Electrónica y de las Comunicaciones
Escuela Politécnica Superior
Universidad Autónoma de Madrid
Junio de 2021

Trabajo parcialmente financiado por el Gobierno de España bajo el proyecto TEC2017-88169-R (MobiNetVideo)

Resumen

Actualmente, las técnicas basadas en aprendizaje profundo o *Deep Learning* han logrado rendimientos realmente buenos en gran cantidad de tareas distintas en las que han sido aplicadas. Respecto a los datos a utilizar, existe una gran disponibilidad de conjuntos distintos que van a permitir el entrenamiento de los modelos. Estos conjuntos de datos pueden estar etiquetados o no, lo cuál es una tarea muy costosa e influye en el tipo de aprendizaje así como en la eficiencia del modelo, siendo aquellos que cuenten con etiquetas más eficientes empleando entonces un aprendizaje supervisado.

Por tanto, el objetivo de este Trabajo de Fin de Grado es la implementación de un sistema de *Deep Learning* basado en redes neuronales convolucionales, el cual tenga la capacidad de resolver un problema de clasificación de imágenes. Para ello, vamos a centrarnos en la evaluación de un caso específico: la segmentación semántica. Sin embargo, vamos a indagar más en esta técnica para lograr diferenciar entre una clase concreta y el resto de la imagen, permitiendo de este modo diferenciar entre un objeto específico y el fondo.

La primera parte del proyecto contendrá el estado del arte, donde veremos los conceptos más generales relacionados con las redes neuronales. Este apartado nos permitirá meternos en el contexto apropiado para así poder entender mejor lo que hemos realizado a lo largo del trabajo.

A continuación, explicaremos cómo funcionará nuestro diseño en cuanto al entorno de desarrollo, las redes empleadas para el entrenamiento de nuestro modelo, los pasos seguidos para la obtención de los resultados y cómo estos pasos han sido modificados a partir del código base.

Finalmente, evaluaremos los experimentos realizados, así como los resultados obtenidos para poder sacar conclusiones acerca del trabajo realizado y, en función de estos, proponer una serie de ideas como trabajo futuro.

Palabras clave

Deep Learning, redes neuronales convolucionales, segmentación semántica, aprendizaje supervisado, funciones de coste, objeto, fondo, *Transfer Learning*, *Fine Tuning*.

Abstract

Currently, techniques based on deep learning or Deep Learning have achieved really good performances in a large number of different tasks in which they have been applied. Regarding the data to be used, there is a great availability of different sets that will allow the training of the models. These data sets may or may not be labeled, which is a very expensive task and influences the type of learning as well as the efficiency of the model, being those that have more efficient labels then employing supervised learning.

Therefore, the objective of this Final Degree Project is the implementation of a Deep Learning system based on convolutional neural networks, which has the ability to solve an image classification problem. To do this, we are going to focus on the evaluation of a specific case: semantic segmentation. However, we are going to investigate further in this technique to be able to differentiate between a specific class and the rest of the image, thus allowing us to differentiate between a specific object and the background.

The first part of the project will contain the state of the art, where we will see the most general concepts related to neural networks. This section will allow us to put ourselves in the appropriate context in order to better understand what we have done throughout the work.

Next, we will explain how our design will work in terms of the development environment, the networks used to train our model, the steps followed to obtain the results and how these steps have been modified from the base code.

Finally, we will evaluate the experiments carried out as well as the results obtained in order to draw conclusions about the work carried out and, based on these, propose a series of ideas as future work.

Key Words

Deep Learning, convolutional neural networks, semantic segmentation, supervised learning, cost functions, object, background, Transfer Learning, Fine Tuning.

Agradecimientos

En primer lugar, quiero dar las gracias a mi tutor, Álvaro García Martín, por ayudarme durante todo el desarrollo del Trabajo de Fin de Grado y por la dedicación y apoyo que ha tenido en mi cuando lo he requerido.

A continuación, y obviamente no por ello menos importante, agradecer de todo corazón a los integrantes de mi familia más cercana compuesta por mis padres, mi hermana Laura, así como por Elvis y Yanira; todo el apoyo, la ayuda, la confianza y la motivación que me han proporcionado durante este largo y duro viaje, aguantándome, respetándome, valorándome y escuchándome en todo momento que lo he necesitado, enseñándome que una familia unida jamás será vencida. También, dedicar este logro a mi abuela fallecida que seguro estaría orgullosísima de mi al verme alcanzarlo.

A todos y cada uno de mis compañeros de la carrera que en menor o mayor medida también me han ayudado a lo largo de esta época de mi vida, en especial a mi grupo más cercano formado en el primer año y con los que he compartido innumerables momentos tanto malos como buenos. Adicionalmente, al equipo de fútbol de la Universidad por esos instantes de desconexión cada semana.

Finalmente, dar las gracias a mis amigos externos de la Universidad por el interés mostrado en mis estudios.

Raúl Arcos Serrano

Julio 2021

Índice general

Índice de Figuras XI

Índice de Tablas XII

1. Introducción 1

1.1. Motivación . . . 1
1.2. Objetivos . . . 1
1.3. Organización de la memoria . . . 2

2. Estado del arte 3

2.1. Introducción . . . 3
2.2. Segmentación semántica . . . 3
2.3. *Deep Learning* . . . 4
2.3.1. Redes neuronales . . . 4
2.3.2. Tipos de aprendizaje . . . 5
2.3.3. Conjunto de datos . . . 6
2.3.4. Parámetros . . . 6
2.3.4.1. Pesos de una red neuronal . . . 7
2.3.4.2. Términos de sesgo y varianza . . . 7
2.3.5. *Data Augmentation* . . . 8
2.3.6. Hiperparámetros . . . 8
2.3.6.1. Tasa de aprendizaje . . . 9
2.3.6.2. Selección tamaño *batch* . . . 9
2.3.6.3. Función de activación . . . 10
2.3.6.4. Iteración y época . . . 11
2.3.7. Función de coste . . . 12
2.3.7.1. Entropía cruzada . . . 12
2.3.7.2. *Dice loss* . . . 12
2.3.7.3. *Tversky loss* . . . 13
2.3.8. Descenso de gradiente . . . 13

2.3.9. Redes neuronales convolucionales (*CNNs*) 14
2.3.9.1. Capas convolucionales 14
2.3.9.2. Capas *Pooling* 15
2.3.9.3. Capas *Fully Connected* 16

3. Diseño y Desarrollo **17**
3.1. Introducción 17
3.2. Entorno de desarrollo 17
3.3. Conjunto de datos empleado 18
3.4. *Transfer Learning* 18
3.5. *CNNs* estudiadas 19
3.5.1. *Resnet18* 19
3.5.2. *VGG16* 20
3.6. Código base 21
3.7. Modificación código base 24

4. Evaluación **25**
4.1. Introducción 25
4.2. Métricas 25
4.3. Comparativa redes 27
4.4. Resultados 27
4.4.1. Entropía cruzada 28
4.4.2. *Dice* 29
4.4.3. *Tversky* 31
4.5. Comparativa resultados 33

5. Conclusiones y Trabajo Futuro **35**
5.1. Conclusiones 35
5.2. Trabajo Futuro 36

Bibliografía **39**

Índice de Figuras

2.1. Segmentación semántica 4
2.2. Redes neuronales 5
2.3. Conjunto de datos 6
2.4. *Underfifting* y *overfifting* 7
2.5. *Data Augmentation* 8
2.6. Tasa aprendizaje 9
2.7. Funciones de activación 11
2.8. Ejemplo de aplicación de *kernel* 15
2.9. Aplicación de *padding* 15
2.10. Comparación entre *max-pooling* y *average-pooling* 16
2.11. Arquitectura clásica de las CNNs 16

3.1. Imagen perteneciente al *dataset Camvid.* etiquetada 18
3.2. *Transfer Learning* 19
3.3. Bloque residual 19
3.4. Arquitectura de *Resnet18* 20
3.5. Arquitectura de *VGG16* 21
3.6. Diagrama de flujo del código base. 23

4.1. Visualización de métrica *IoU* 26
4.2. Gráfica comparativa entre redes en función de la precisión y la cantidad de operaciones realizadas 27
4.3. Imagen seleccionada del conjunto de prueba para la evaluación 28
4.4. Ejemplo de entropía cruzada para multiclase, 'coche', 'ciclista' y 'señal de tráfico'. . . . 29
4.5. Ejemplo de *Dice* para multiclase, 'coche', 'ciclista' y 'señal de tráfico'. 30
4.6. Ejemplo de *Tversky* para multiclase, 'coche', 'ciclista' y 'señal de tráfico'. 32
4.7. Comparación de clasificación de la clase 'ciclista' para cada función de coste. 33

Índice de Tablas

4.1. Resultados obtenidos para las redes *Resnet18* y *VGG16* con función de coste entropía cruzada y caso multiclase con mismos hiperparámetros. 27

4.2. Resultados obtenidos para la red *Resnet18* y función de coste entropía cruzada a nivel *dataset* 28

4.3. Resultados obtenidos para la red *Resnet18*, función de coste entropía cruzada y única clase frente multiclase 29

4.4. Resultados obtenidos para la red *Resnet18* y función de coste *Dice* a nivel *dataset* 30

4.5. Resultados obtenidos para la red *Resnet18*, función de coste *Dice* y única clase frente multiclase 30

4.6. Resultados obtenidos para la red *Resnet18* y función de coste *Tversky* a nivel *dataset* . . 31

4.7. Resultados obtenidos para la red *Resnet18*, función de coste *Tversky* y única clase frente multiclase 31

1
Introducción

1.1. Motivación

La segmentación fondo-frente tiene como finalidad la discriminación entre objetos pertenecientes al primer plano (frente o *foreground*) de una imagen del resto de los objetos (fondo o *background*).

Sin embargo, la segmentación objeto-fondo consiste en la identificación de aquellas zonas de la imagen que con mayor probabilidad no son un objeto determinado y, por tanto, pertenecen al fondo de la escena. De este modo, se consigue una segmentación enfocada en la determinación de zonas presentes en la escena que no son objetos mientras que los segmentadores fondo-frente están orientados a la correcta clasificación de los objetos de frente, por lo que son orientaciones completamente distintas.

Este tipo de segmentación puede ser de mucha utilidad en muchas aplicaciones de procesamiento automático de vídeo. Puede ser usado tanto como un pre-procesado como post-procesado en algoritmos de tracking, estimación de densidad de objetos, *etc*.

Actualmente, la extensión del uso de grandes cantidades de datos de entrenamiento o *Big Data* para modelar cualquier tipo de objeto y su detección ha sido bastante notoria. Concretamente, el uso de redes convolucionales presenta en diversos casos mejoras considerables en la tarea de detección de cualquier tipo de objeto.

Por este motivo, el objetivo será el estudio de un algoritmo capaz de evaluar correctamente la segmentación objeto-fondo.

1.2. Objetivos

El desafío de este Trabajo de Fin de Grado es el diseño e implementación de un algoritmo de segmentación objeto-fondo basado en segmentación semántica, que permita la distinción de diferentes objetos presentes en la imagen, mediante el uso de redes neuronales convolucionales y su evaluación para distintas funciones de coste.

Para ello, se realizarán las pruebas necesarias para una correcta valoración con objetos de tamaño razonadamente grandes, así como menos voluminosos.

El objetivo final mencionado puede diferenciarse en una serie de retos parciales que han ido apareciendo a lo largo del desarrollo del proyecto:

- Investigación del estado del arte.
- Progreso del algoritmo aplicado para el entrenamiento del modelo y definición de las tareas en torno a las cuales se van a realizar los posteriores experimentos.
- Evaluación de los experimentos.

1.3. Organización de la memoria

El presente trabajo se divide de la siguiente manera:

- Capítulo 1. Introducción del Trabajo de Fin de Grado donde definiremos tanto las motivaciones como los objetivos.
- Capítulo 2. Explicación de la técnica de segmentación semántica así como los conceptos básicos empleados en *Deep Learning*, prestando especial atención a las funciones de coste que serán muy relevantes en este trabajo. Finalmente, analizaremos también detalladamente las redes neuronales convolucionales puesto que van a ser empleadas en este proyecto.
- Capítulo 3. Visualización del entorno de desarrollo aplicado y del conjunto de datos empleado. Además, analizaremos la técnica de *Transfer Learning* y comentaremos las redes neuronales convolucionales que se estudiarán en el trabajo. Para terminar, explicaremos tanto el código base empleado como las modificaciones realizadas en él para obtener los posteriores resultados.
- Capítulo 4. Observación de las métricas en torno a las que se van a evaluar los posteriores resultados para cada función de coste y se realizará la comparación de estos. También se argumentará la elección de la red neuronal convolucional aplicada.
- Capítulo 5. Conclusiones respecto a los resultados obtenidos en el capítulo anterior y posibles trabajos futuros a llevar a cabo.
- Bibliografía.

2

Estado del arte

2.1. Introducción

En este capítulo, vamos a diferenciar entre un par de secciones donde la primera será bastante breve, la cual explicará en que consiste la técnica de segmentación semántica, y la segunda será considerablemente extensa que englobará una gran cantidad de conceptos de *Deep Learning* que serán indispensables para ponernos en situación y así poder entender de manera clara los posteriores apartados.

2.2. Segmentación semántica

La segmentación semántica [1] se refiere al algoritmo empleado en *Deep Learning* el cual proporciona a cada píxel de una imagen una categoría o etiqueta específica. De esta forma, podemos diferenciar distintos tipos de clases en cada imagen lo que abre bastante el abanico de aplicaciones que tiene esta técnica como pueden ser:

- **Conducción autónoma:** permitiendo de esta forma diferenciar entre diversos obstáculos que puedan influir en la conducción (peatones, señales de tráfico...) y de la propia carretera, evitando de esta forma posibles accidentes.
- **Generación de imágenes vía satélite:** diferenciando entre diferentes tipos de terrenos como desiertos, ríos
- **Generación de imágenes médicas:** con el fin de detectar e identificar entre diferentes anomalías cancerosas en las células.

Cabe mencionar que para realizar clasificación de imágenes, también es posible el uso de la técnica de detección de objetos (como podemos apreciar en la Figura 2.1). Esta última se encarga de clasificar en función de una ventana rectangular como resultado, indicando donde se ha detectado un objeto y cuya evaluación depende de la precisión de estas ventanas, las cuales limitan la clasificación a objetos que encajen en ellas, por lo que es más útil la segmentación semántica al permitir la detección de objetos de manera clara con formas irregulares y arbitrarias.

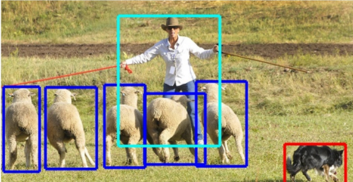

Figura 2.1: Comparación de la técnica detección de objetos y de segmentación semántica para la misma imagen. Extraído de [2].

Atendiendo al funcionamiento de la segmentación semántica, debemos tener en cuenta el transcurso del entrenamiento de una red para clasificar imágenes, el cual consta de una serie de pasos:

1. Análisis del conjunto de imágenes con píxeles ya etiquetados.
2. Creación de la red de segmentación semántica.
3. Entrenamiento de la red para la clasificación de de las imágenes en las distintas categorías.
4. Evaluación de la precisión de la red.

2.3. *Deep Learning*

Para poder entender el concepto de aprendizaje profundo o *Deep Learning* [3], previamente debemos definir otro concepto, el de *Machine Learning* o aprendizaje automático, el cual corresponde a una disciplina científica del ámbito de la Inteligencia Artificial que proporciona a las máquinas la capacidad de poder identificar distintos patrones en cantidades de datos enormes con el fin de realizar predicciones. De esta manera, las máquinas pueden desempeñar tareas concretas sin la necesidad de ser programados para ello.

Una vez visto en que se basa el aprendizaje automático, *Deep Learning* lo emplea basándose en redes neuronales o *Neuronal Networks*. Cada red neuronal artificial se compone de un número de capas o niveles jerárquicos. En el primer nivel, la red aprende las cosas más simples y las envía al siguiente nivel, el cual la recopila y la combina para obtener información cada vez más compleja, y se lo pasa a la siguiente capa, y así sucesivamente.

Como idea general, *Deep Learning* se basa en el propio mecanismo del cerebro humano para intentar realizarlo informáticamente, más concretamente reproducir el funcionamiento de las neuronas cerebrales que están interconectadas entre ellas recibiendo a su entrada información, procesandola y transmitiendola a otras, con el fin de obtener la mejor salida o respuesta.

2.3.1. Redes neuronales

Cada una de las redes neuronales puede presentar un número concreto tanto de neuronas como de capas (que se compone de conjunto de neuronas cada una), lo que supondría que a mayor número la red sería más compleja, significando así ser capaz de realizar tareas de mayor dificultad pero como desventaja tendría un entrenamiento muy costoso computacionalmente así como temporal, necesitando un mayor conjunto de datos de entrada.

En cuanto a la estructura de una red [4], tenemos que tener en cuenta la existencia de tres tipos distintos de capas: capa de entrada, oculta o de salida. La primera de ellas no presenta ramificaciones a la entrada de sus neuronas sino que únicamente reciben estímulos de entrada desde el exterior. La capa o capas ocultas (intermedias) se componen de ramificaciones tanto a la entrada como a la salida de

sus neuronas. Finalmente, la capa de salida la constituyen neuronas con ramificaciones solamente a la entrada ya que la salida de estas presentarán el estímulo de salida de la red.

Además, en función del número de capas ocultas que presente la red, podemos diferenciar entre redes neuronales simples (una capa) o complejas (varias), como podemos ver en la Figura 2.2. Las redes neuronales complejas son las empleadas en *Deep Learning*.

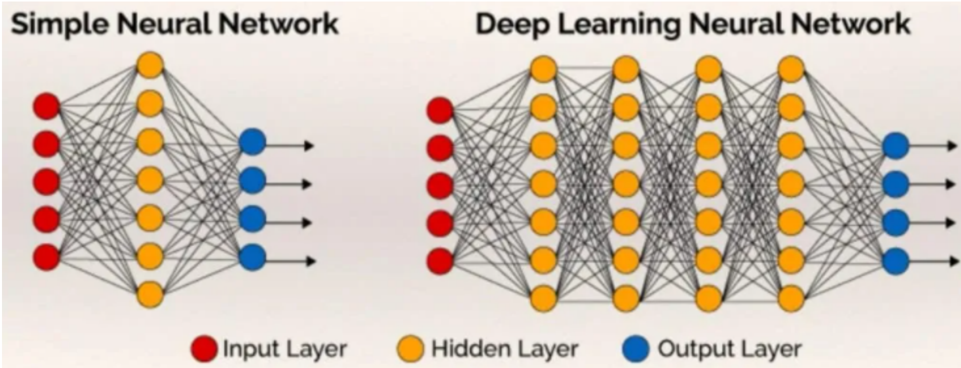

Figura 2.2: Ejemplo de una red neuronal simple y otra de *Deep Learning*. Extraído de [4].

Cada una de estas neuronas va a tener asociado un valor numérico, denominado peso, que determina la intensidad de interacción entre neuronas de capas adyacentes, lo que permitirá que todas las neuronas de la red estén conectadas. Estos pesos son fundamentales para que la red aprenda correctamente aunque encontrar los valores que mejores resulten pueda ser complicado.

Un algoritmo de gran importancia en las redes neuronales es el *forwardpropagation* (o propagación hacia delante), mediante el cual se crean las predicciones tras pasar los datos del conjunto de entrenamiento por todas las neuronas, desde la capa de entrada hasta finalmente la capa de salida, donde la red neuronal consigue predecir la clase a que pertenece cada dato.

Además, para evaluar el aprendizaje de una red neuronal, tendremos que prestar atención a la función de pérdida o de coste (la cuál veremos con más detalle más adelante), donde el objetivo es minimizarla para que el error de predicción disminuya. Para obtener los gradientes de la función de coste a cada peso, podemos emplear el algoritmo de *backpropagation* (propagación hacía atrás) que consiste en recorrer la red en sentido contrario (desde la capa de salida hasta la de entrada) para utilizar los cómputos de gradientes de la capa actual para obtener los de las capas anteriores y, en último lugar, obtener los de la capa de entrada. Así, las neuronas de la red reciben una señal de error que define su contribución relativa al error total.

2.3.2. Tipos de aprendizaje

Cuando hablamos de aprendizaje en las redes neuronales, nos referimos al entrenamiento de los parámetros o pesos, los cuales se calculan y se optimizan con la finalidad de obtener la respuesta adecuada. Además, debemos distinguir entre varios casos [5] en función de sus condiciones:

- **Aprendizaje supervisado**: corresponde a un aprendizaje controlado con un conocimiento previo, basado en etiquetas asociadas a los datos de entrada, que permita clasificarlos correctamente. Así, sabiendo la salida que deseamos obtener gracias al etiquetado, podemos corregir la red cuando no obtengamos la salida correcta. Un ejemplo sería la elección de un correo como spam o no.

- **Aprendizaje no supervisado**: en este caso no se parte de un conocimiento previo etiquetado, sino que únicamente están presentes los datos de entrada, los cuales se analizan con el objetivo de poder obtener algunas características en común para poder agruparlos en diferentes categorías. Se

suele emplear en el marketing para analizar datos y posteriormente hacer publicidad acorde a los resultados.

- **Aprendizaje por refuerzo**: asociado al algoritmo que permite aprender a partir de la experiencia, con la finalidad de obtener la solución más óptima frente a diferentes situaciones gracias a un proceso de prueba y error, premiando los pesos de la red en caso de acierto y penalizandolos en caso erróneo. El reconocimiento facial es un ejemplo que aplica dicho aprendizaje.

2.3.3. Conjunto de datos

En el momento que decidimos emplear las redes neuronales, a la entrada de la primera capa de la red debemos proporcionar una serie de datos que nos sirvan de base para comenzar a entrenar un algoritmo, con el fin de que una máquina pueda aprender, a los cuáles nombraremos conjunto de datos o *dataset*. Para alcanzar el anterior objetivo, debemos dividir dicho *dataset* en tres conjuntos diferentes (ver la Figura 2.3 para entenderlo visualmente):

- **Conjunto de entrenamiento (*training*):** en la gran mayoría de casos se corresponde con el conjunto de mayor tamaño (cercano a 3 quintos) y se emplea para entrenar, es decir, ajustar los parámetros de la red en la etapa de aprendizaje para que las respuestas producidas en la capa de salida se ajusten lo más posible a los datos ya conocidos. De esta manera, se elaboran una serie de modelos para solventar el problema propuesto.
- **Conjunto de validación (*validation*):** encargado de seleccionar y ajustar los hiperparámetros de la red neuronal con el fin de validar todos los modelos anteriores y seleccionar el más óptimo atendiendo a la cercanía de la predicción. En relación a su tamaño, normalmente depende de la cantidad de hiperparámetros que tengamos aunque generalmente se corresponde en torno a una quinta parte del conjunto general. Sin embargo, no siempre es necesario dicho conjunto puesto que si no hubiera hiperparámetros lo podríamos omitir.
- **Conjunto de prueba (*test*):** empleado para evaluar el rendimiento del modelo obteniendo así un error de generalización cercano al real. En este caso ya no se modifican los parámetros ni los hiperparámetros. Su tamaño suele ser de dimensión parecida al conjunto de validación, es decir, una quinta parte del *dataset*.

Figura 2.3: Posibilidades de división del conjunto de datos. Extraído de [6].

2.3.4. Parámetros

Los parámetros de las redes neuronales [7] son aquellos que se obtienen durante el proceso de entrenamiento a partir del conjunto de datos empleado. Se consideran claves ya que permiten realizar

predicciones y definirán cómo es de capaz la red de resolver el problema en cuestión. También, para estimarlos, se emplean algoritmos de optimización donde el más común es el descenso de gradiente.

En función del número de parámetros que contenga el modelo, podemos diferenciar dos tipos: los modelos paramétricos, los cuales presentan un número fijo de parámetros; y los no paramétricos, cuyo número es variable.

Para tener en cuenta, los parámetros más destacables en las redes neuronales son los dos siguientes:

2.3.4.1. Pesos de una red neuronal

Como mencionamos anteriormente en 2.3.1, los pesos son los valores numéricos asociados a las neuronas, los cuales indican en qué magnitud interacciona cada neurona con las de las capas adyacentes, logrando que finalmente todas las neuronas de la red acaben conectadas entre sí.

2.3.4.2. Términos de sesgo y varianza

Relacionado con este par de conceptos y, en función de sus valores, debemos entender los problemas [8] que se puedan ocasionar apoyados de la Figura 2.4:

- **Error de sesgo (o *bias*):** diferencia producida entre la predicción esperada de nuestro modelo y los valores reales. Entonces, para un correcto funcionamiento de nuestro modelo es deseable obtener un bajo sesgo. En caso contrario, si obtenemos un alto sesgo, sufriremos un problema de *underfifting* (o subajuste), significando que la red neuronal no aprendió correctamente sobre el conjunto de datos de entrenamiento, puesto que necesita más suposiciones, produciéndose un desempeño poco eficiente en todas las predicciones.

- **Error de varianza (o *variance*):** valor que define la variabilidad del modelo al probar diferentes conjuntos de datos de entrenamiento. De este modo, es preferible obtener alta varianza con el fin de poder amoldarse a varios conjuntos de entrenamiento. Sin embargo, si se presenta baja varianza, significaría que tendríamos un problema para nada deseable de *overfifting* (o sobreajuste), produciéndose un sobre entrenamiento haciendo que la red neuronal aprenda mucho sobre ese conjunto de datos de entrenamiento, teniendo como consecuencia un mal desempeño en el conjunto de datos de validación o en datos que no ha visualizado con anterioridad.

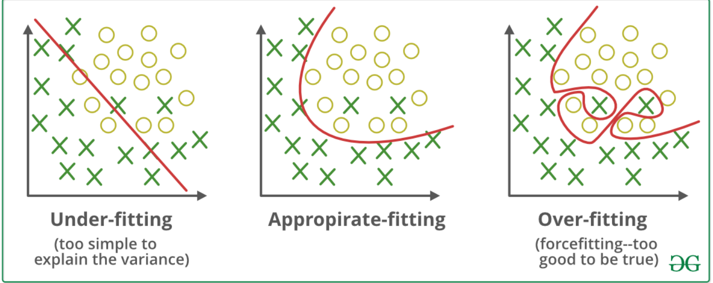

Figura 2.4: Ejemplificación para caso de *underfifting* y *overfifting* así como de un caso deseable. Extraído de [9].

Como conclusión, los modelos más óptimos deben encontrar un buen equilibrio entre *bias* y *variance*, para así, poder presentar un punto intermedio, ya que el aumento del *bias* supondría una disminución de la varianza y viceversa, y evitar los problemas mencionados.

2.3.5. *Data Augmentation*

Data Augmentation [10] se refiere a la técnica empleada durante el entrenamiento de las redes neuronales que, cuyo principal fin es aumentar el *dataset* mediante la generación artificial de datos, realizando modificaciones a los datos del conjunto original, logrando una mayor diversidad para obtener una precisión mejor sin perder el control del *overfifting*.

De esta forma, durante el entrenamiento, nuestro modelo en cuestión en ningún momento apreciará con exactitud la misma imagen en las diversas épocas. A pesar de ser una idea muy simple, es a la par muy potente ya que expone al modelo a más casos distintos lo que permite que pueda realizar mejor generalización.

Entonces, aunque existe una gran variedad de modificaciones (como las apreciadas en la Figura 2.5), las más comunes que se suelen aplicar a las imágenes del conjunto de datos de entrenamiento son:

- Reflexión aleatoria en el eje vertical (arriba-abajo) y/o horizontal (izquierda-derecha)
- Rotación aleatoria de la imagen medido en grados
- Traslación aleatoria horizontal y/o vertical de la imagen de entrada
- Aplicación de factor de escala aleatorio
- Alteraciones en el contraste, saturación y brillo

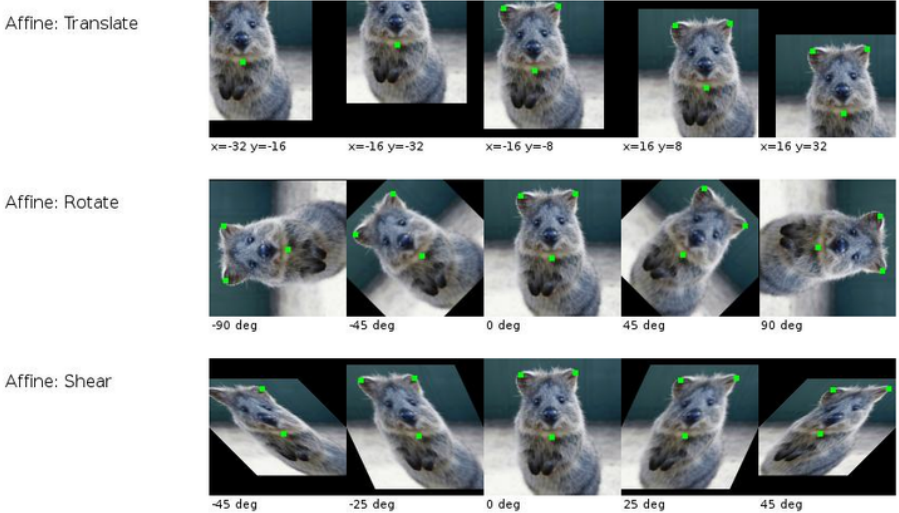

Figura 2.5: Representación de algunos ejemplos de posibles modificaciones mediante *Data augmentation*. Extraído de [11].

2.3.6. Hiperparámetros

Los hiperparámetros [12] de las redes neuronales son aquellos valores modificables con el fin de controlar el entrenamiento de un modelo y que no son obtenidos del conjunto de datos. Por tanto, al ser ajustables, trataremos de encontrar el valor más adecuado para cada hiperparámetro con la finalidad de tratar de conseguir el mejor rendimiento de la red. Este proceso de ajuste es manual y computacionalmente costoso.

Algunos de estos hiperparámetros más comunes, a parte del número de neuronas y capas ocultas de la red, son:

2.3.6.1. Tasa de aprendizaje

También conocida como factor de aprendizaje [13] o *learning rate*, cuyo símbolo es α, es uno de los hiperparámetros el cual indica cómo de rápido se ajusta el modelo concreto al problema en cuestión. Dicho de otra forma, mide el ajuste de los pesos de la red respecto a la función de coste, la cual interesa minimizar lo máximo posible.

Gracias a su valor, podremos observar posteriormente en una gráfica como influye en la función de coste respecto a las épocas (ver la Figura 2.6) y, en consecuencia, saber si el entrenamiento es bueno o no. Además, hay que tener especial cuidado con el valor que se le proporcione ya que se pueden dar varias casuísticas:

- Si la tasa de aprendizaje es alta, podría provocar que se produjeran oscilaciones grandes evitando alcanzar el punto óptimo pudiendo incluso, en caso extremo, llegar a divergir.
- En caso contrario, si el valor es pequeño supondría que no perderíamos ningún mínimo local significando que la función de coste disminuya, pero podría costar demasiado tiempo en llegar a converger al realizar muchas más iteraciones y, en el peor de los casos, no llegar a converger.

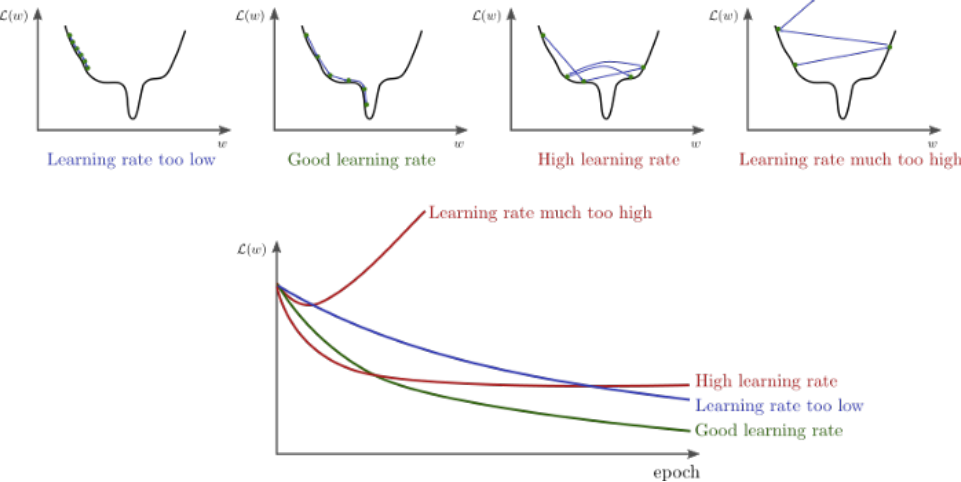

Figura 2.6: Representación del aspecto de la función de coste en función de las épocas según la elección de la tasa de aprendizaje. El caso ideal sería la curva verde. Extraído de [13].

Al tratarse de un hiperparámetro, podremos ir modificándolo para obtener los resultados esperados, es decir, si observamos que su valor es muy alto lo disminuiremos y viceversa, evitando divergir y converger.

2.3.6.2. Selección tamaño *batch*

Para entender este apartado, definiremos en primer lugar el término *batch* o lote. El *batch* consiste en el conjunto de datos que se procesan en cada iteración de cada época. De esta forma, en función del parámetro que define el tamaño del lote (o *batch size*), podemos observar tres casos diferentes [14]:

- **Caso *batch*:** referente a la elección de un tamaño de lote equivalente al número total de datos empleados en el proyecto. De esta forma estaríamos seleccionando el valor máximo posible lo que permitiría bastantes buenos resultados devolviendo el valor más exacto de la función de error, pero como inconveniente tendría un costo computacional inmenso.

- **Caso estocástico:** si en el caso anterior seleccionábamos el valor máximo, en este caso escogeremos el valor mínimo de tamaño, es decir, equivalente a 1. Esto supondría un costo computacional bastante reducido pero sin embargo, al actualizarse los pesos de cada neurona en cada época, supondría una gran inestabilidad al producirse variaciones constantemente por lo que no tendríamos buenos resultados de entrenamiento, obteniendo únicamente una aproximación al valor real de la función de error.

- **Caso *mini-batch*:** con el fin de obtener las ventajas de cada una de los anteriores casos, podemos seleccionar un valor intermedio de *batch size* superior a la unidad e inferior al tamaño total. Así, lograríamos un menor coste computacional, lo que permitiría mayor rapidez a la hora de entrenar, y sin embargo, mejores resultados en la función de error. Además, con esta opción, al no pasar todo el conjunto de datos totales, añadiremos ruido lo que hará que de cara a usar diferentes *dataset*, el funcionamiento sea más óptimo para otros conjuntos.

Como resumen, la mejor elección del *batch size*, sería el mayor posible que se pueda soportar computacionalmente hablando (aunque sea más lento), lo que permitiría acercarse a un valor más exacto de la función de error, y en función de la polivalencia que queramos obtener.

2.3.6.3. Función de activación

En una red neuronal tenemos que tener en cuenta tres principales funciones. La primera se conoce como función de entrada, la cual se encarga de conseguir una única entrada global para cada neurona obtenida del resultado de la combinación lineal de los pesos y sus entradas previas.

La tercera función recibe el nombre de función de salida, únicamente encargada de la transmisión del valor de salida de una neurona presente a la próxima neurona a la que se asocie, actuando entonces como entrada en esta última.

Por tanto, la función intermedia entre ambas es la función de activación [4], cuya misión es recibir el valor de la función de entrada y obtener el correspondiente de la función de salida, pero como su propio nombre indica, proporcionando como de activa está cada neurona en función del sesgo (b), el cual mide la distancia entre la predicción y el valor real. Visualmente lo podemos apreciar más fácilmente gracias a la siguiente fórmula que representa esta suma ponderada z:

$$z = b + \sum_i w_i x_i \tag{2.1}$$

El sentido que tiene la inserción de funciones de activación es proporcionar cierta no linealidad al modelo a entrenar para dotarlo de mayor capacidad al momento de resolver otros problemas con mayor complejidad. En caso de que estas funciones fueran lineales, las capas ocultas de la red neuronal carecerían de sentido.

Según la elección tomada, podemos distinguir entre diversas opciones (ver también la Figura 2.7):

- **Función identidad:** transmite sin realizar modificación alguna la combinación lineal.

- **Función threshold:** también conocida como función escalón, clasifica estrictamente con un 0 si el valor z es negativo o 1 en caso contrario, sin opción a valores intermedios.

- **Función sigmoide:** gracias a la regresión logística, cuanto más positivo es el valor de z más cercano será el resultado a 1 y por el contrario, más a 0.

- **Función ReLU:** hace referencia a función rectificadora donde para valores negativos de z se proporcionará un 0 y para valores positivos, el propio valor de z.

- **Función tangente hiperbólica:** muy parecida a sigmoide, pero los valores son comprendidos entre -1 y 1.

Nombre	F(z)	Representación
Identidad	z	
Threshold	$\begin{cases} Z & Si(z>0) \\ 0 & Si(z<0) \end{cases}$	
Sigmoide	$\frac{1}{1+\exp(-z)}$	
ReLU	$\begin{cases} 1 & Si(z>0) \\ 0 & Si(z<0) \end{cases}$	
Tangente hiperbólica	$\frac{1-\exp(-z)}{1+\exp(-z)}$	

Figura 2.7: Resumen visual de las funciones de activación antes mencionadas. Basado en [15].

2.3.6.4. Iteración y época

Dos conceptos que tenemos que conocer en cuanto al entrenamiento de las redes neuronales, así como saber diferenciar ya que se pueden producir confusiones entre ambos, son la iteración y la época [16].

En primer lugar, la iteración podemos definirla como una única actualización de los distintos pesos del modelo a entrenar. Así, es un concepto asociado con el balance de los distintos gradientes correspondientes a cada uno de estos parámetros con respecto a la pérdida para solamente un único lote de datos.

Entonces, cuando queremos entrenar una red neuronal, con el fin de obtener buenos resultados es necesario proporcionar el conjunto de datos a la entrada en varias ocasiones y no solo una vez. Por esta razón surge el concepto de época (o *epoch*), referido al recorrido realizado por todo el conjunto de datos a través de la red neuronal durante el entrenamiento, completando una pasada hacia delante (*forwardpropagation*) y otra hacia atrás (*backpropagation*) con el objetivo de actualizar los pesos, por todos los datos de entrenamiento.

Además, para poder relacionar estos dos conceptos necesitamos conocer el tamaño del *batch*, siendo una época compuesta por tantas iteraciones equivalentes al número del conjunto de datos de entrenamiento entre el tamaño definido del *batch*:

$$iteracciones = \frac{\textit{tamaño conjunto entrenamiento}}{\textit{tamaño batch}} \tag{2.2}$$

De este modo, si el número de datos del conjunto de entrenamiento fuese igual al tamaño del lote, estaríamos en un caso en el que una época se completaría con una sola iteración.

Para concluir este apartado, a medida que aumentamos el número de épocas, los pesos asociados a las distintas capas de la red se actualizarán en más ocasiones, por lo que sería una solución para solucionar

el problema de *underfifting*, aunque si el aumento es muy grande podríamos pasar de un caso ya óptimo a *overfifting*.

2.3.7. Función de coste

La función de coste [4] es otro hiperparámetro cuya misión es la de calcular la diferencia existente entre el valor de salida real del conjunto de datos y el valor de salida obtenido tras el entrenamiento de la red neuronal (valor estimado). Una vez definida, el objetivo para poder realizar un buen entrenamiento será minimizar dicha diferencia en cada iteración, es decir, nos estaríamos acercando más al valor real de salida y, de esta forma, optimizando los parámetros de la red.

En cuanto a los tipos de funciones de coste, el más común para los problemas de regresión es el conocido como Error Cuadrático Medio o *Mean squared error*, el cual calcula el sumatorio para todas las muestras (n) de las distancias elevadas al cuadrado entre la variable definida como objetivo (y_i) y los valores obtenidos en la predicción (y_i^p):

$$MSE = \frac{\sum_{i=1}^{n}(y_i - y_i^p)^2}{n} \tag{2.3}$$

Si hablamos de problemas de clasificación (como el nuestro), algunos de ellos son:

2.3.7.1. Entropía cruzada

La función de coste entropía cruzada [17] (o *cross entropy*), o también conocida como pérdida logarítmica, consiste en la medida del rendimiento de un modelo utilizado para clasificación donde la salida corresponde a un valor de probabilidad comprendido entre 0 y 1. Dicha función aumenta cuando la probabilidad que se ha predicho diverge de la etiqueta real que le corresponde, penalizando logarítmicamente dicha diferencia, por lo que interesa minimizarla lo máximo posible. Por esto, lo más favorable sería obtener una pérdida logarítmica lo más cercana a cero, donde lo ideal sería que fuera cero.

$$loss_{Cross} = \frac{1}{N}\sum_{i=1}^{M} T_i log(X_i) \tag{2.4}$$

La fórmula incluye X_i, que es la respuesta de la red, T_i el valor objetivo, M el número total de respuestas en X (en todas las observaciones y categorías) y N el número total de observaciones en X.

2.3.7.2. *Dice loss*

Esta función de coste está basada en el coeficiente de similitud *Dice* [18], el cual evalúa la superposición existente entre dos muestras segmentadas. El valor de este coeficiente está comprendido entre 0 y 1, donde cuanto más cercano a la unidad esté dicho valor, mejor y más completa será la superposición. La fórmula para obtener dicho coeficiente es:

$$DICE = \frac{2|A \bigcap B|}{|A| + |B|} \tag{2.5}$$

donde en el numerador encontramos el doble de la intersección entre ambos conjuntos A y B, y en el denominador la suma del número de de elementos de cada conjunto.

Una vez obtenido este coeficiente, para obtener la función de coste debemos emplear la siguiente operación:

$$loss_{Dice} = \frac{\sum_{i=1}^{N}(1 - DICE)}{N} \tag{2.6}$$

donde N es el tamaño del batch.

2.3.7.3. *Tversky loss*

En este caso, la función de coste se basa en el índice de *Tversky* [19], que al igual que en el caso anterior, también calcula la superposición entre dos imágenes segmentadas. El índice está comprendido entre 0 y 1, donde interesa valores cercanos a 1 para minimizar la función de coste. Se calcula el índice de la siguiente manera:

$$TI_c = \frac{\sum_{n=1}^{N} Y_{cn}T_{cn}}{\sum_{n=1}^{N} Y_{cn}T_{cn} + \alpha \sum_{n=1}^{N} Y_{cn}T_{\overline{c}n} + \beta \sum_{n=1}^{N} Y_{\overline{c}n}T_{cn}} \tag{2.7}$$

donde T es el valor real de salida, Y el valor de predicción, N el número de elementos del conjunto, c corresponde a una clase determinada y $\overline{c}$ no pertenece a ella, α y β son los factores de ponderación controladores de la contribución a la pérdida de los falsos positivos y falsos negativos de cada clase.

Finalmente, la obtención de la función de coste para C clases sería:

$$loss_{Tversky} = \sum_{c=1}^{C} 1 - TI_c \tag{2.8}$$

2.3.8. Descenso de gradiente

El descenso de gradiente [13] es uno de los algoritmos más conocidos de optimización empleados en el entrenamiento de las redes neuronales, cuyo objetivo es el ajuste de parámetros mientras que también se minimiza la función de coste, lo que supondría reducir el error y que la red sea en consecuencia más efectiva.

También, este algoritmo está relacionado con el tamaño del *batch*. Por tanto, igual que lo razonado anteriormente, lo más adecuado sería el caso *mini-batch*, permitiendo introducir a la red N muestras en cada iteración en vez de una sola (caso estocástico) manteniendo las ventajas de ser un entrenamiento más rápido al no necesitar pasar todo el conjunto de datos de golpe (caso *batch*), por lo que será menos costoso computacionalmente y añadiremos también cierta aleatoriedad a la entrada de la red.

Para aplicar dicho algoritmo, se deben seguir una serie de pasos:

1. Introducción de N muestras aleatorias del dataset de entrenamiento, previamente etiquetado.
2. Obtención de las predicciones de salida mediante algoritmo *forwardpropagation*.
3. Evaluación de la función de coste para el *mini-batch* aplicado, la cual se trata de minimizar para alcanzar el objetivo del descenso de gradiente.
4. Cálculo del gradiente mediante la derivada multivariable de la función de coste respecto a los parámetros de la red. Obtendremos un vector que proporciona tanto el sentido como la dirección en la cual aumenta de manera más rápida la función de coste, por lo que debemos movernos en sentido contrario para lograr minimizarla. El cálculo de este vector se realiza gracias al algoritmo de *backpropagation*, el cual permite obtener más fácilmente mediante la regla de la cadena los

gradientes de cada capa gracias a los de la capa posterior, es decir, partiremos de la capa final y recorreremos la red hacía atrás para finalmente obtener, de capa posterior a capa anterior, el gradiente de la capa inicial.

5. Actualización de los parámetros de la red quitando a su valor actual la multiplicación del valor del gradiente que le corresponda por el factor de aprendizaje.

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta) \tag{2.9}$$

Por tanto, al depender de la tasa de aprendizaje, iremos modificando este hiperparámetro hasta que obtengamos resultados esperados tras el entrenamiento.

2.3.9. Redes neuronales convolucionales (*CNNs*)

Las redes neuronales convolucionales o *CNNs (Convolutional Neural Networks)* [20] son aquellas aplicadas en aprendizajes supervisados, las cuales realizan el procesamiento de sus capas basándose en la corteza visual del cerebro con el fin de identificar una serie de características en las entradas, permitiendo así identificar distintos objetos. Por ello, este tipo de redes se componen de varias capas ocultas especializadas con determinadas jerarquías, donde las primeras capas son capaces de detectar más genéricamente (curvas, líneas) y las más profundas realizan detecciones más complejas (rostros, siluetas...), consiguiendo así buenos resultados para problemas de segmentación y clasificación de imágenes.

A diferencia de las redes neuronales tradicionales, a la entrada de las *CNNs* recibiremos los pixeles de las imágenes, las cuáles tendrán un tamaño de WxHxD, correspondiente a los píxeles de ancho y alto de la imagen, así como al número de canales de la imagen. Dicho tamaño definirá el número de neuronas de la capa de entrada ya que por ejemplo una imagen b/n de ancho y alto de 28 píxeles emplearía 784 neuronas en dicha capa, y el triple de ellas si se trata de imagen a color (RGB son 3 canales).

Respecto a su arquitectura (apreciable en la Figura 2.11), es importante diferenciar entre tres tipos de capas donde cada una de ellas tiene una función específica:

2.3.9.1. Capas convolucionales

Dentro de este tipo de redes son las capas más costosas computacionalmente ya que se realizan convoluciones [21], cuyo proceso se produce mediante un producto escalar entre un conjunto de píxeles de la imagen de entrada con una serie de filtros, también conocidos como *kernels*, permitiendo la obtención de una nueva matriz denominada matriz de activación o mapa de características. Siendo más precisos, dicho *kernel* se irá desplazando a través de la imagen horizontalmente (de izquierda a derecha) y, posteriormente, verticalmente (tras completar una fila pasaremos a la siguiente) por cada uno de los píxeles del conjunto de la imagen de entrada obteniendo de esta manera cada píxel de la matriz de activación, siendo así cada uno de ellos combinación lineal de varios píxeles de la imagen de entrada.

Por tanto, en estas capas hay tres conceptos que nos van a ser de utilidad:

- ***Kernel:*** filtro aplicado a los píxeles de la imagen de entrada cuya función es la obtención tanto de características como de patrones de las imágenes (ver Figura 2.8). Respecto a su tamaño, es menor al del conjunto de píxeles de la imagen de entrada que escogemos en cuanto a filas y columnas, manteniendo el número de canales, lo cual influye ya que tendremos que emplear 3 filtros para imágenes RGB, uno idéntico para cada componente, que posteriormente se sumarán para formar una imagen.

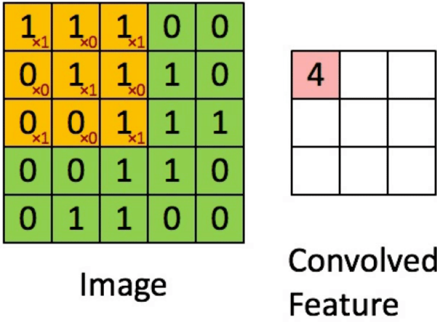

Figura 2.8: Ejemplo de aplicación de *kernel* de tamaño 3x3 para una imagen 5x5 obteniendo imagen resultante 3x3. Extraído de [22].

- ***Stride:*** al igual que hemos mencionado antes que el *kernel* se desplaza por cada píxel (1,1) ya que es lo más común, este parámetro conocido también como zancada puede hacer variar ese desplazamiento, ya que se puede determinar por ejemplo que se desplace de dos en dos píxeles horizontal y verticalmente (2,2), lo que obviamente afectaría también a la matriz de activación resultante.
- ***Padding:*** o lo que es lo mismo relleno, consiste en la adición de píxeles con un valor de 0 tanto en columnas y filas con el fin de que cada píxel de la imagen pueda ocupar el centro del *kernel* y así, conseguir una imagen de salida sin reducción de tamaño ya que sería idéntico al de la de entrada (visualmente en la Figura 2.9). De todos modos es una técnica que se suele aplicar generalmente para adquirir más información de las esquinas y arreglar el efecto de borde.

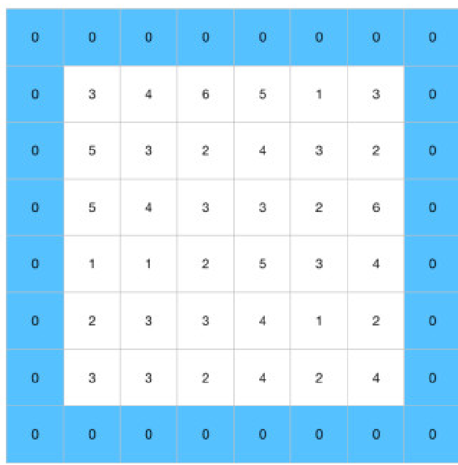

Figura 2.9: Aplicación de *padding* gracias a las zonas de color azul. Extraído de [16].

2.3.9.2. Capas *Pooling*

En estas capas de *pooling* [23], o de reducción, se realiza el proceso de muestreo ya que cogeremos la matriz de activación y nos quedaremos con las características más relevantes, eliminando aquellas de menor relevancia, reduciendo así su tamaño. De este modo, reduciremos considerablemente computacionalmente hablando ya que eliminaremos información que no tenga tanta importancia, disminuyendo así el tiempo a invertir para cálculos en capas posteriores y reduciendo además el problema de *overfitting*.

Este muestreo se puede hacer de dos maneras distintas (visibles también en la Figura 2.10):

- ***Max-pooling:*** con esta técnica escogeremos los valores máximos para cada ventana o región aplicada a la matriz de activación, descartando por tanto el resto de valores.
- ***Average-pooling:*** en este caso, realizaremos previamente la media de los valores de cada ventana y nos quedaremos con el valor resultante.

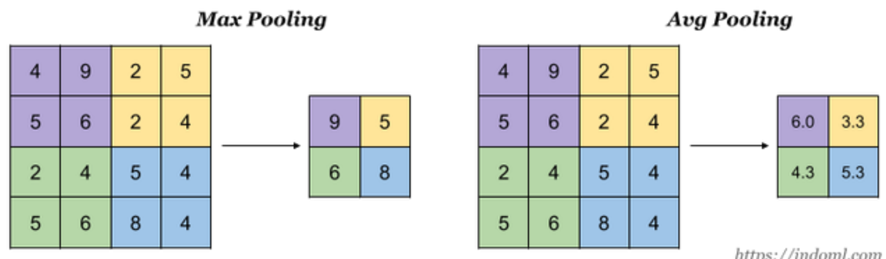

Figura 2.10: Comparación entre *max-pooling* y *average-pooling*. Extraído de [24].

2.3.9.3. Capas *Fully Connected*

Estas capas totalmente conectadas [20] son las que se emplean al final de las redes neuronales convolucionales con la finalidad de clasificar las imágenes por clases.

La función de estas capas consiste en recibir el mapa de características de la capa anterior, el cual ha sido reducido a un vector ya que antes era tridimensional mediante aplanamiento, entonces ya podríamos trabajar como una capa de neuronas de redes neuronales tradicionales. Es en este momento cuando se aplica la función *softmax*, que se encarga de la conexión de esta capa con la de salida final pasándole el vector compuesto de la probabilidad que tiene la imagen de entrada entre 0 y 1 de pertenecer a cada clase. Esta última capa está compuesta de tantas neuronas como clases haya.

Una vez vistas las distintas capas de las *CNNs*, podemos observar, en la Figura 2.11, una arquitectura clásica de las redes neuronales convolucionales:

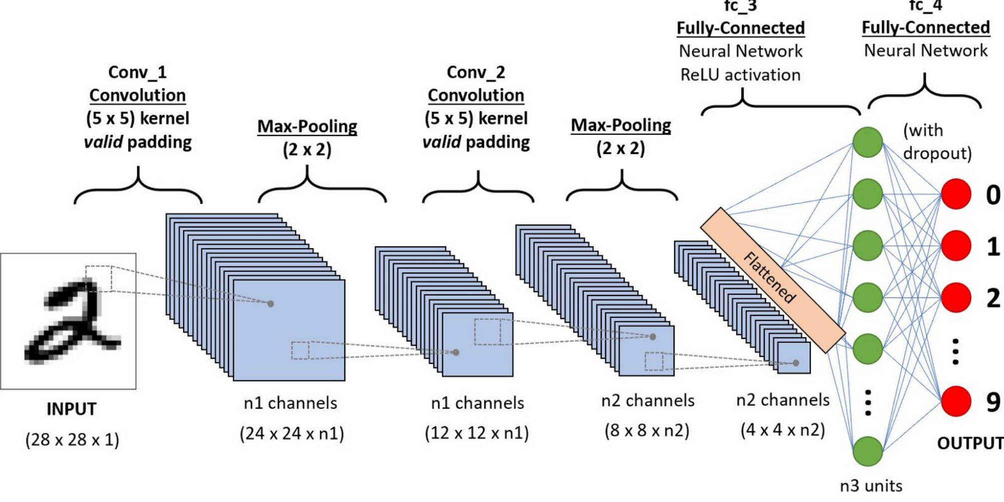

Figura 2.11: Arquitectura clásica de las *CNNs*, donde *flattened* significa aplanado para que la capa anterior tridimensional deje de serlo y trabajar con un vector. Extraído de [25].

3

Diseño y Desarrollo

3.1. Introducción

Una vez explicados los conceptos más generales en el estado del arte para ponernos en el contexto adecuado, veremos a continuación más detalles acerca del trabajo realizado.

En primer lugar, comentaremos el entorno de desarrollo aplicado y familiarizarnos un poco más con él. Posteriormente, observaremos la base de datos empleada a lo largo de este proyecto, así como las arquitecturas de las redes neuronales escogidas para nuestro experimento. Finalmente, explicaremos de forma más minuciosa el desarrollo realizado en este caso para la obtención de nuestros resultados, los cuales trataremos en el siguiente apartado más precisamente.

3.2. Entorno de desarrollo

El entorno empleado para nuestro trabajo ha sido el conocido como Matlab, el cual es un programa computacional caracterizado por realizar ejecuciones de operaciones muy variables así como gran cantidad de tareas matemáticas, que inicialmente fue creado como una herramienta para trabajar tanto con matrices y vectores pero que actualmente permite abarcar muchos ámbitos bastante útiles en ciencia e ingeniería, como por ejemplo el *Deep Learning*.

Además, otra característica importante es la representación en figuras de funciones o resultados que queramos y, de esta forma. poder obtener una visualización bastante clara de cada trabajo que estemos realizando.

Finalmente, también es de gran utilidad el tratamiento con imágenes que permite dicho programa, siendo de gran importancia para nuestro proyecto.

3.3. Conjunto de datos empleado

En este trabajo se ha utilizado el *dataset Camvid* [26] de Cambridge, siendo una base de datos compuesta por 701 imágenes, obtenidas a partir de la grabación de vídeo durante 10 minutos de las vistas a nivel de calle durante la conducción de un vehículo, etiquetadas semánticamente a nivel de píxel permitiendo diferenciar entre 32 clases distintas de objetos. Detalladamente, estas 32 clases son: cielo, puente, edificio, pared, túnel, arco, cono de tráfico, poste de columna, carretera, líneas circulación, líneas de no circulación, acera, estacionamiento, arcén, árbol, vegetación miscelánea, señal de tráfico, texto misceláneo, semáforo, valla, coche, camioneta, camión, tren, vehículo, peatón, niño, carrito de bebés, animal, ciclista, scooter y la opción resto.

El etiquetado de las imágenes a nivel píxel se realizó por una persona y posteriormente se revisó por otra distinta para así garantizar la buena precisión, y poder lograr los objetivos definidos para el algoritmo: reconocimiento de objetos para clases múltiples, detección de peatones y propagación de etiquetas.

A continuación, podemos observar una de las imágenes del *dataset* en la Figura 3.1, donde en primer lugar se aprecia la imagen original, en el medio la imagen correspondiente de etiquetas, y por último, la imagen de etiquetas superpuesta en la original, obteniendo una imagen etiquetada:

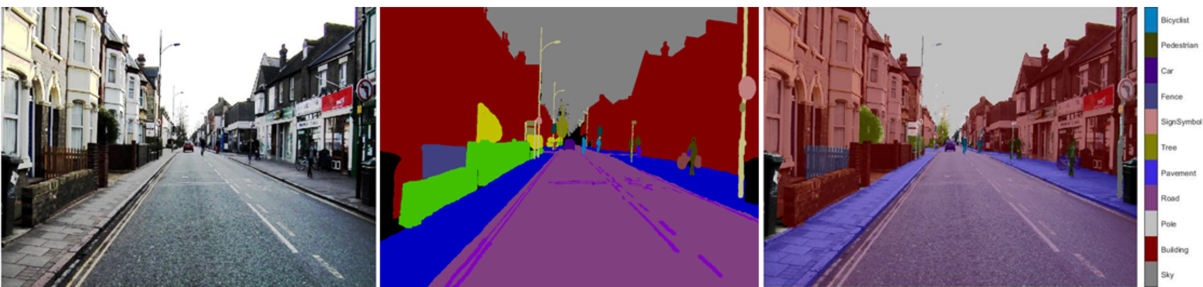

Figura 3.1: Imagen perteneciente al *dataset Camvid* a la que se le ha aplicado el proceso de etiquetado necesario para el aprendizaje supervisado.

3.4. *Transfer Learning*

Este concepto [27] nos va a facilitar bastante la tarea de entrenamiento de nuestra red neuronal convolucional computacionalmente ya que no realizamos este entrenamiento desde cero, sino que mediante esta técnica escogeremos e implementaremos a nuestro caso una red ya pre-entrenada, por lo que no requiere una inversión de tiempo excesivamente costosa al adaptar dicha red a nuestra tarea concreta, quedándonos con las capas que realizan similares funciones (para características más generales) y reemplazando el resto (más específicas) por aquellas que se ajusten a nuestra tarea.

De este modo, las redes pre-entrenadas han utilizado un conjunto de datos enorme para su entrenamiento, que puede durar meses aún usando GPU, con el fin de ser reutilizada posteriormente como inicialización para otras tareas relacionadas con la que se ha entrenado o como extractor de características.

Finalmente, mencionar que implementado en *Transfer Learning* se ha empleado la técnica de *Fine Tuning* o ajuste fino (ver Figura 3.2), consistente en no solo la actualización de la arquitectura de la *CNN*, sino que además se re-entrenará para poder realizar una clasificación diferente a la de la red pre-entranada y adaptarse a nuestra nueva tarea de clasificación.

Figura 3.2: Comparación entre *Transfer Learning* y entrenamiento desde cero. Extraído de [28].

3.5. *CNNs* estudiadas

Las redes neuronales convolucionales que se han valorado para nuestro trabajo son las siguientes:

3.5.1. *Resnet18*

Cuando mencionamos el término *Resnet*, proveniente de *Residual Neural Network* [29], nos referimos a un modelo de red neuronal artificial que implementa tipos de conexiones basadas en atajos, con el fin de poder omitir algunas capas sustituyendolas por bloques residuales, los cuales suelen tratarse de saltos dobles o triples, ya que se observó que cuando se le añadían un mayor número de capas a las redes neuronales convolucionales, estas no optimizaban la precisión final sino que además la disminuían y por tanto no se lograba ninguna mejoría.

Empleando estos bloques residuales (ver Figura 3.3), la información relevante podía ser transmitida de capas anteriores a las posteriores evitando así el inconveniente de desvanecimiento de gradientes. Esto se debe a que las capas de dichos bloques pueden aprender las diferencias entre salida y entrada, permitiendo añadir un mayor número de capas sin influir negativamente logrando que el mapa de activación sea el óptimo a la entrada. Sorprendentemente, aunque pueda parecer un paso adicional en las redes neuronales convolucionales, lo cierto es que acelera el proceso de entrenamiento de la red.

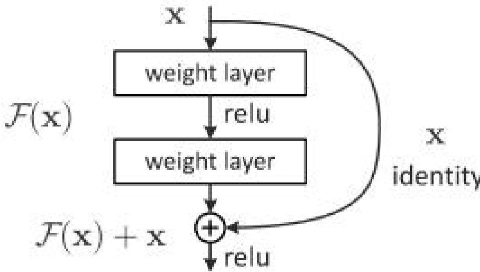

Figura 3.3: Bloque residual. La información se actualiza con la función residual o pasa directamente por la identidad. Extraído de [29].

Por tanto, el gradiente podrá tomar dos posibles caminos en función de si la entrada es óptima o no.

En caso negativo, tomará el camino de la identidad, y en caso afirmativo, se podrá actualizar gracias a la comparación con la función residual *F(x)* sin la necesidad de rehacer el mapa.

Relativo a la arquitectura *Resnet18* compuesta por 18 capas (ver Figura 3.4), la imagen entrante en la primera capa convolucional de la red tiene que ser RGB de tamaño 224x224x3. Posteriormente, observamos la presencia de cuatro capas convolucionales, las cuales fueron resultado de los bloques residuales. Después, aparece una capa *average pooling* para lograr la reducción de la salida obtenida en la capa anterior. Para finalizar, se utiliza en primer lugar una capa *fully connected* asociada al número total de clases con el que estamos trabajando y, al resultado de este proceso, se emplea una función de activación de softmax que permita obtener las probabilidades de salida definitivas, y en función de ellas, clasificar para una clase u otra.

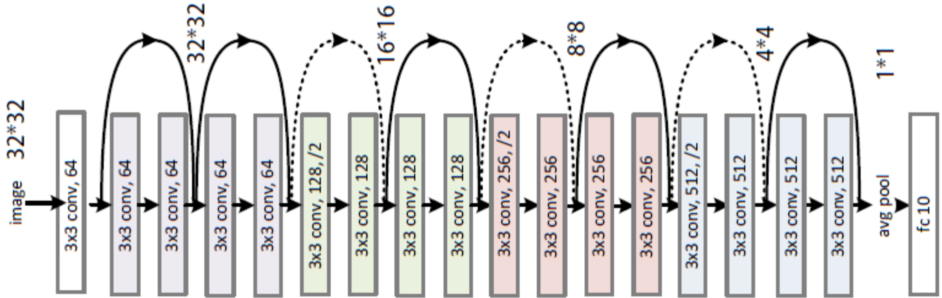

Figura 3.4: Arquitectura de *Resnet18*. Extraído de [29].

3.5.2. *VGG16*

El modelo de red neuronal convolucional *VGG16* [30] debe su existencia a K. Simonyan y A. Zisserman, pertenecientes a la Universidad de Oxford, quienes lo presentaron en la competición *ILSVRC-2014* (*ImageNet Large Scale Visual Recognition Competition* de la convocatoria de ese año correspondiente) convirtiéndolo en uno de los más famosos de la competición. En las pruebas realizadas para este modelo, *VGG16* logra una precisión entre el 96 % y 97 % para un conjunto de datos enorme compuesto por una cantidad superior a los 14 millones de imágenes pertenecientes a 1000 clases distintas, el cual es conocido como *ImageNet*. No fue una tarea temporalmente breve ya que *VGG16* se entrenó durante muchas semanas y empleaba *NVIDIA Titan Black GPU*.

El principal propósito de la aparición de esta red es su pre-entrenamiento para poder solventar un problema bastante amplio para posteriormente poder resolver un caso más específico basándose en dicha red. En otras palabras, el objetivo es la utilización de redes previamente pre-entrenadas (*VGG16*) con bases de datos grandes para finalmente adaptarlas a nuestro problema de interés.

En cuanto a las características de esta red convolucional pre-entrenada, destacan:

- Facilidad de comprensión e implementación de su arquitectura.
- Composición de 16 capas como su nombre indica, donde 13 de ellas son convolucionales y el resto densas.
- Gran precisión para problemas de clasificación

Relativo a la arquitectura (ver Figura 3.5), podemos observar en la imagen que la entrada a la capa inicial se compone de una imagen de tamaño fijo 224x224x3 RGB. Dicha imagen se va tratando a través de un conjunto de capas convolucionales cuyos filtros aplicados son de tamaño 3x3 (lo más pequeño permitido para englobar la noción arriba-abajo, derecha-izquierda, centro). En un momento dado del

proceso se emplea un filtro convolucional de 1x1, aplicando así una transformación lineal de los canales de entrada.

Todas y cada una de estas capas convolucionales utilizadas incluyen *max pooling* al final para reducir las dimensiones de la salida de cada una y poder ser procesadas correctamente después. También, todas las capas ocultas cuentan con la función de activación *ReLU*.

Además, en la arquitectura se emplean tres capas *fully connected* al final, donde las dos primeras utilizan 4096 neuronas, y la tercera reduce el número a 1000, igualando de esta forma al número de clases para las que fue pre-entrenada con *ImageNet*. Para terminar, la capa *softmax* permite la clasificación en las diferentes clases según las probabilidades establecidas.

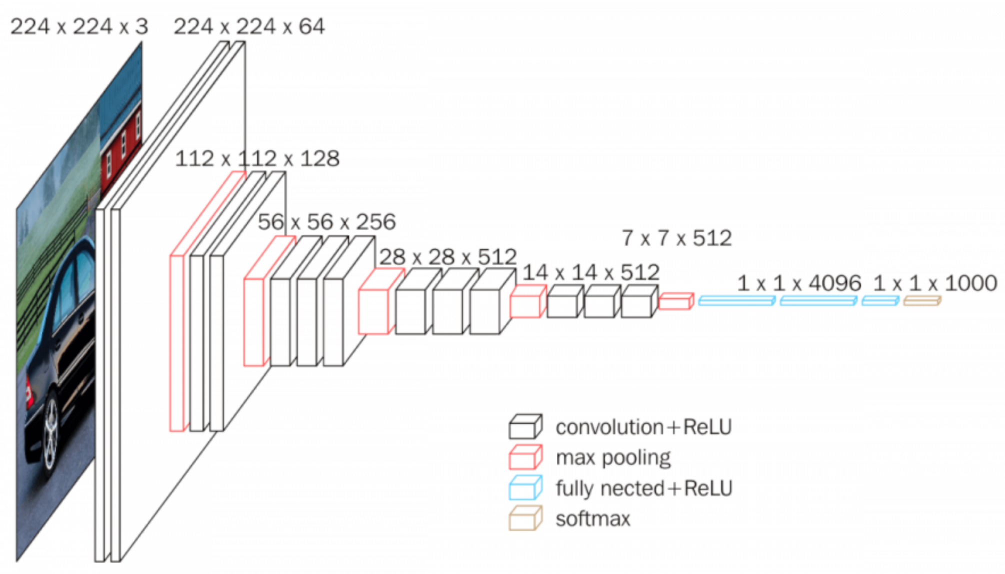

Figura 3.5: Arquitectura de *VGG16*. Extraído de [30].

3.6. Código base

En función de la *CNN* que empleemos, partiremos del código base, que emplea el *dataset CamVid* para 11 clases distintas en lugar de 32 como inicialmente dicho *dataset*, de [31] para *Resnet18* o de [32] para *VGG16*. Ambos códigos comparten gran cantidad de similitudes pero efectivamente tienen diferencias como veremos a continuación.

Para ver un seguimiento más claro de los pasos seguidos, y así poder apreciar mejor las diferencias y semejanzas, lo mostraremos tanto textualmente como visualmente gracias al posterior diagrama de flujo de la Figura 3.6:

1. **Instalación red pre-entrenada:** en función del caso, descargaremos previamente la red *Resnet18* o *VGG16* para su posterior instalación en el programa y poder trabajar con ella.

2. **Implementación *dataset CamVid*:** realizaremos tanto la descarga del conjunto de datos como de las correspondientes etiquetas. A continuación, realizaremos una representación de una imagen para corroborar que se ha producido correctamente.

3. **Implementación etiquetas:** una vez descargadas, especificaremos el número de clases con las que trabajaremos, en este caso 11. Por tanto, se han realizado agrupamientos de las 32 clases iniciales del siguiente modo:

- Cielo
- Edificio: puente, edificio, pared, túnel, arco
- Poste: cono de tráfico, poste de columna
- Carretera: carretera, líneas circulación, líneas de no circulación
- Pavimento: acera, estacionamiento, arcén
- Árbol: árbol, vegetación miscelánea
- Señal de tráfico: señal de tráfico, texto misceláneo, semáforo
- Valla
- Coche: coche, camioneta, camión, tren, otro vehículo
- Peatón: peatón, niño, carrito de bebés, animal
- Ciclista: ciclista, scooter

Una vez tenemos la división en estas clases, se proporcionarán etiquetas a cada clase y posteriormente, realizaremos la representación de una imagen de etiquetas superponiéndola con su imagen asociada encima para comprobar el correcto funcionamiento.

4. **Análisis de estadísticas del *dataset:*** en este momento obtendremos la frecuencia de aparición de cada clase en el *dataset:*. El número de píxeles por etiqueta de cada clase lo obtenemos por un lado respecto a todo el *dataset:* (*PixelCount*) y por otro lado por imagen (*ImagePixelCount*). Entonces, la frecuencia de cada clase se obtendría con:

$$f_c = \frac{PixelCount_c}{\sum_{i=1}^{c} PixelCount_i} \tag{3.1}$$

Apreciamos que las clases en *CamVid* están desequilibradas, puesto que clases como cielo, carretera o edificios al cubrir más área en la imagen tienen considerablemente más píxeles que por ejemplo peatones y ciclistas, dificultando de este modo el entrenamiento de la red ya que el aprendizaje está sesgado a favor de las clases dominantes.

5. **Cambio tamaño de datos:** este paso se realiza únicamente en *VGG16*, en el cual se reduce el tamaño de los datos pasando las imágenes de entrada de 720x960 a 360x480 con el fin de que el entrenamiento sea menos costoso computacional y temporalmente hablando.

6. **División de *dataset:*:** para *Resnet18*, el *dataset:* se divide en 60/20/20 para entrenamiento, validación y prueba respectivamente, mientras que para *VGG16* únicamente empleamos 60 % entrenamiento y 40 % restante para prueba.

7. **Creación de la red:** en este momento es cuando creamos nuestra red a partir de la red pre-entrenada escogida, donde realizaremos tanto la modificación de nuestra red necesaria para transferir los pesos de la red pre-entrenada como la agregación de capas adicionales necesarias para llevar a cabo la segmentación semántica.

8. **Equilibrio mediante ponderación de clases:** se realiza para solventar el problema del desequilibrio antes mencionado en el paso 4 de las clases. Por tanto, ahora calcularemos las ponderaciones de clase de frecuencia media gracias a la mediana del valor denominado frecuencia de imagen resultante de la división para cada clase de *PixelCount* entre *ImagePixelCount*. Posteriormente, crearemos la nueva capa de clasificación pasando, entre otros, estos pesos ponderados de clases como argumentos y actualizaremos la red con esta nueva capa, reemplazandola por la capa de clasificación actual. De este modo, se solventará el favoritismo del aprendizaje hacía clases dominantes. Cabe destacar que la función de coste empleada por defecto es la entropía cruzada.

9. **Selección de opciones de entrenamiento:** especificaremos algunos hiperparámetros para el entrenamiento de la red como por ejemplo la elección del algoritmo de optimización descenso por gradiente, la tasa de aprendizaje fijada en 0,001, el máximo de épocas definido en 10, el tamaño del batch escogido con valor 8. . . Para *Resnet18* añadimos el conjunto de validación puesto que en *VGG16* no lo empleamos.

10. ***Data Augmentation*:** emplearemos esta técnica incluyendo translación aleatoria tanto en el eje horizontal como vertical para un rango de -10 a 10 píxeles y reflexión aleatoria únicamente en el eje horizontal.

11. **Entrenamiento:** para comenzar con el entrenamiento serán necesarios tanto el conjunto de datos de entrenamiento (al cual se aplica previamente *Data Augmentation*), las opciones de entrenamiento y la red creada tras los pasos anteriores.

12. **Evaluación de resultados:** lo veremos en el apartado final.

El diagrama de flujo tiene entonces el siguiente aspecto:

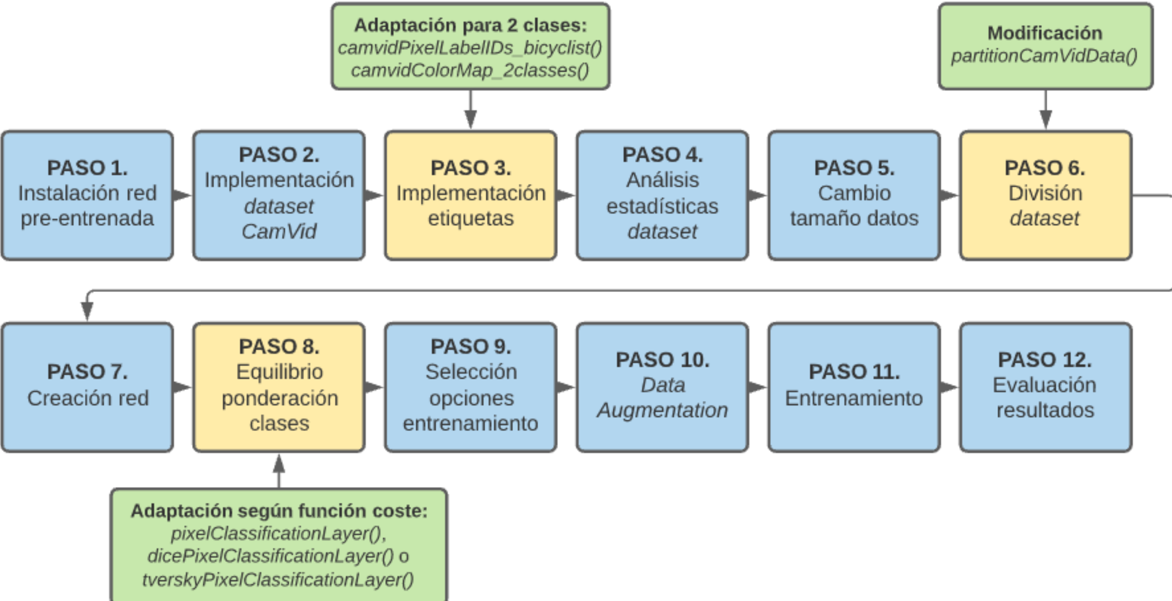

Figura 3.6: Diagrama de flujo del código base donde las zonas azules representan los pasos que no requieren modificación, los bloques naranjas los que sí la requieren y las cajas verdes representan las modificaciones realizadas.

En el siguiente apartado, veremos la modificación que se ha realizado al código según proceda, apoyándonos en el diagrama de flujo expuesto en la Figura 3.6.

3.7. Modificación código base

Para ajustar el código base a nuestro trabajo para realizar la segmentación semántica objeto-fondo, ha sido necesaria la modificación de ciertas partes del código.

Para comenzar, el array compuesto por las 11 clases ha sido modificado de tal forma que quedan únicamente 2 clases: *background* y la clase que queramos evaluar.

Una vez tengamos sólo ese par de clases, debemos modificar internamente la función denominada *camvidPixelLabelIDs()* para que la clase que deseemos obtenga su ID de etiqueta correspondiente y el resto de clases se agruparán bajo un mismo ID de etiqueta y así asociarse como fondo. Tras la modificación se le ha denominado a la nueva función bajo el nombre de, como por ejemplo para el caso de haber escogido la clase ciclista, *camvidPixelLabelIDs_bicyclist()*.

Otra función modificada sería *camvidColorMap()*, donde el mapa de colores se compondrá únicamente de dos, uno asociado al fondo y otro a la clase específica. Tras la modificación, a la nueva función se le proporciona, para cualquier clase escogida, el nombre de *camvidColorMap_2classes()*. Hasta este momento, las modificaciones se realizan en el paso 3, influyendo como consecuencia cada una en los pasos posteriores.

Después, será necesario reemplazar internamente únicamente la línea en la que se hace mención a *camvidPixelLabelIDs()* por el nuevo nombre que le hayamos dado tras la modificación anterior, de la función *partitionCamVidData(imds,pxds)*, esta última devuelve la división del *dataset* en *train, validation* y *test* (en *VGG16* se omite validación). Estos cambios se producen en el paso 6.

Finalmente, en el paso 8, tendremos que adaptar la función *pixelClassificationLayer* dependiendo de la función de coste que decidamos emplear, puesto que por defecto se emplea entropía cruzada. Por tanto, realizaremos la implementación de las funciones *dicePixelClassificationLayer* y *tverskyPixelClassificationLayer* respectivamente.

4

Evaluación

4.1. Introducción

Para el capítulo de evaluación, inicialmente definiremos en torno a que valores vamos a analizar nuestro trabajo para posteriormente realizar comparaciones y sacar las conclusiones finales.

También, realizaremos la comparativa de las redes comentadas en el capítulo anterior para argumentar la selección de solamente una de ellas.

Finalmente, se realizará la exposición de los resultados obtenidos para las funciones de coste definidas en el estado del arte y lograr obtener las conclusiones posteriores.

4.2. Métricas

En primer lugar vamos a describir en base a qué resultados vamos a evaluar el trabajo de segmentación semántica. A nivel de clase, podremos apreciar el impacto que tiene cada clase en el rendimiento general con las siguientes métricas [33]:

- ***Accuracy*****:** corresponde a la precisión mediante el porcentaje de píxeles identificados correctamente para cada clase, por lo que podremos saber cómo de bien cada una de las clases identifica correctamente los píxeles. Por tanto, el cálculo de esta métrica es sencillo mediante:

  $$Accuracyscore = \frac{TP}{TP + FN} \tag{4.1}$$

  donde TP son verdaderos positivos y FN falsos negativos.

- ***IoU*****:** la intersección sobre la unión, o también coeficiente de similitud de *Jaccard*, es la métrica más común si lo que se busca es precisión estadística que penalice los falsos positivos. Por tanto, para cada clase, *IoU* es la proporción de píxeles clasificados correctamente respecto al número total de píxeles reales y predichos en esa clase (graficamente en Figura 4.1). La fórmula difiere

ligeramente a la de *Accuracy* puesto que tenemos en cuenta los falsos positivos (FP) también:

$$IoU score = \frac{TP}{TP + FP + FN} \tag{4.2}$$

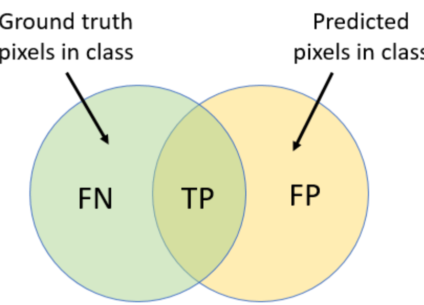

Figura 4.1: Visualización de métrica *IoU*. Extraído de [33].

- ***MeanBFScore*:** consiste en el promedio de la puntuación de coincidencia de contorno de límite F1 (BF) de una clase específica en todas las imágenes, la cual indica cómo de bien se alinea el límite predicho de dicha clase con el límite real. Esta métrica se emplea si se quiere obtener una mejor evaluación cualitativa humana que la métrica de *IoU*.

Por otro lado, en cuanto a *dataset* se refiere, veremos los siguientes conceptos que proporcionarán una descripción general de alto nivel del rendimiento de la red:

- ***GlobalAccuracy*:** identifica la proporción de píxeles clasificados correctamente, independientemente de la clase, con respecto al número total de píxeles. La principal ventaja es que permite una estimación rápida y económica desde el punto de vista computacional del porcentaje de píxeles clasificados correctamente.

- ***MeanAccuracy*:** se trata de la precisión promedio de todas las clases en todas las imágenes del conjunto de datos.

- ***MeanIoU*:** corresponde al promedio de *IoU* de todas las clases en todas las imágenes.

- ***WeightedIoU*:** *IoU* promedio de cada clase, ponderado por el número de píxeles de esa clase concreta. Esta métrica se emplea cuando las imágenes tienen clases de tamaño desproporcionado y así lograr reducir el impacto de los errores en las clases pequeñas en la puntuación de calidad agregada.

- ***MeanBFScore*:** al igual que mencionábamos antes, consiste en el promedio de la puntuación de coincidencia de contorno entre la segmentación predicha y la real, pero orientado esta vez al *dataset*.

Por tanto, para cada entrenamiento de la red para casos objeto-fondo analizaremos las segundas métricas y las compararemos con los casos multiclase mediante las primeras métricas para ver cuánto difieren ambos parámetros.

4.3. Comparativa redes

En el siguiente apartado vamos a analizar las redes expuestas en la sección 3.5 realizando una prueba para ambas redes con los mismos hiperparámetros con un mismo conjunto de datos de validación correspondiente al 20 % y función de coste entropía cruzada. Los resultados sobre el conjunto de *test* para ambas pruebas son los siguientes de la Tabla 4.1:

Red	G. Accuracy	M. Accuracy	M. IoU	W. IoU	M. BFScore
Resnet18	0,88310	0,85865	0,63517	0,81701	0,66315
VGG16	0,89000	0,85708	0,65367	0,82536	0,68367

Tabla 4.1: Representación de los resultados obtenidos para las redes *Resnet18* y *VGG16* con función de coste entropía cruzada y caso multiclase con mismos hiperparámetros.

Podemos observar que los resultados son prácticamente los mismos al diferir en muy poco los de una red respecto a la otra para cada métrica. Sin embargo, la diferencia para ambas pruebas está en el tiempo de ejecución ya que la red *VGG16* es computacionalmente muy costosa respecto a *Resnet18*.

Por esto mismo, la red seleccionada para realizar el resto de pruebas será *Resnet18* ya que, como podemos observar en la Figura 4.2, la precisión para problemas de clasificación es prácticamente la misma para ambas redes pero el número de operaciones es muchísimo más alto, debido a su profundidad, para *VGG16* que para *Resnet18* (en torno a 30G y 5G operaciones respectivamente), produciendo una diferencia de tiempo en entrenamientos de la red de mínimo un par de días frente a solamente 8 horas que necesita *Resnet18*.

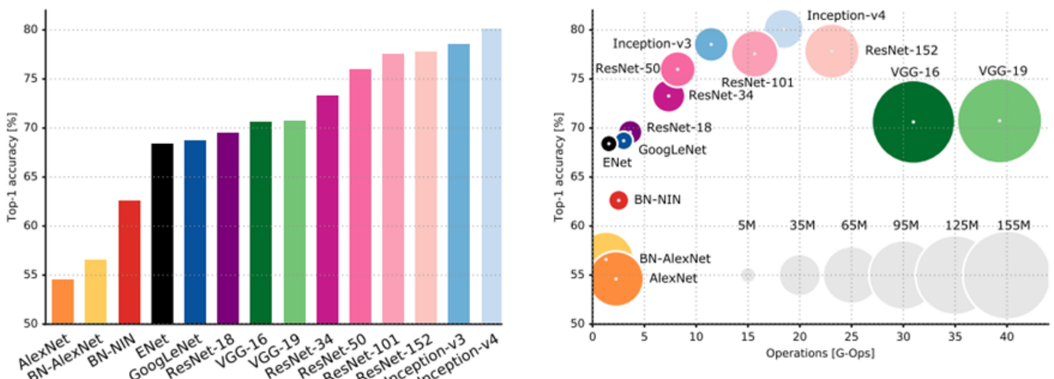

Figura 4.2: Gráfica comparativa entre redes en función de la precisión y la cantidad de operaciones realizadas. Extraído de [34].

4.4. Resultados

Una vez escogida la red que emplearemos, vamos a proceder a la muestra de resultados para su posterior comparación en el próximo apartado. Para ello, vamos a presentar los datos obtenidos para cada una de las funciones de coste con las que trabajaremos (entropía cruzada, *Dice* y *Tversky*) tanto para el caso multiclase como para 4 clases concretas: peatón, señal de tráfico, ciclista y coche.

La exposición de los resultados se realizará acorde a las métricas mencionadas anteriormente (Sección 4.2) para la imagen de la Figura 4.3, es decir, por un lado a nivel *dataset* y por otro a nivel clase.

Figura 4.3: Imagen seleccionada del conjunto de prueba para la evaluación.

En cuanto a *dataset* se refiere, las métricas obtenidas habrán evaluado para todo el conjunto de datos de *test* tanto el caso objeto-fondo para las 4 clases antes mencionadas, donde dichos valores proporcionados representarán la precisión de detección de dicha clase frente al resto de la imagen; como el caso multiclase del que partíamos, indicándonos cómo de bien se han detectado de media en general todas las 11 clases, de las que partíamos inicialmente, para el *dataset* en cuestión.

Si nos referimos a las métricas para nivel clase, estaremos evaluando en este caso la presesencia de cada clase evaluada para la imagen seleccionada (en nuestro caso la Figura 4.3), comparando dichos valores de cada objeto-fondo (únicamente 2 clases) con las métricas obtenidas en el caso de las 11 clases iniciales, es decir, cada clase respecto de las otras 10 presentes y no solamente respecto del fondo. Debido a esto, cada tabla contendrá los casos de cada una de estas clases (caso objeto-fondo) y, además, el caso multiclase para cada objeto evaluado. De este modo, haremos una comparativa entre solamente la selección de un objeto frente al fondo, y la selección de una clase en presencia de otras 10.

Tras lo comentado, procedemos ahora a la presentación de los resultados para cada una de las funciones de coste, donde los valores de cada una de las tablas posteriores destacarán de color verde para resaltar el valor óptimo para cada métrica y, en caso contrario, de color rojo.

4.4.1. Entropía cruzada

Como podemos apreciar en las tablas mostradas a continuación, los objetos que obtienen los mejores resultados a nivel *dataset* (ver Tabla 4.2) son los más voluminosos como es el caso de coches y ciclistas, siendo menos efectivos para peatón y señales de tráfico en ese orden. También, los peores resultados, a pesar de ser buenos, se aprecian para el caso multiclase ya que se evalúan 11 clases en lugar de solamente dos (objeto y fondo) por lo que las métricas son más generales.

Caso	**G. Accuracy**	**M. Accuracy**	**M. IoU**	**W. IoU**	**M. BFScore**
Ciclista	0,99397	0,96571	0,71794	0,99113	0,84573
Peatón	0,98293	0,96543	0,63412	0,97779	0,71104
Coche	0,97461	0,97572	0,82249	0,95718	0,72502
Señal tráfico	0,97154	0,92376	0,63322	0,96198	0,63208
Multiclase	0,88310	0,85865	0,63517	0,81701	0,66315

Tabla 4.2: Representación de los resultados obtenidos para red *Resnet18* y función de coste entropía cruzada a nivel *dataset*.

A nivel clase (ver Tabla 4.3), se aprecia también que los mejores resultados se producen para los objetos más grandes (coche) que para los más pequeños (señal tráfico o peatón) para la imagen evaluada:

Objeto	Caso	Accuracy	IoU	MeanBFScore
Ciclista	Clase	0,93717	0,44193	0,40646
	Multiclase	0,91970	0,54576	0,48988
Peatón	Clase	0,94767	0,28543	0,43666
	Multiclase	0,84418	0,42099	0,58738
Coche	Clase	0,97697	0,67177	0,53685
	Multiclase	0,89814	0,78250	0,73298
Señal tráfico	Clase	0,87465	0,29524	0,36162
	Multiclase	0,78405	0,41887	0,51325

Tabla 4.3: Representación de los resultados obtenidos para red *Resnet18*, función de coste entropía cruzada y caso de única clase frente multiclase.

Visualmente, podemos observar las diferencias entre multiclase y selección de algún objeto frente al fondo en la Figura 4.4:

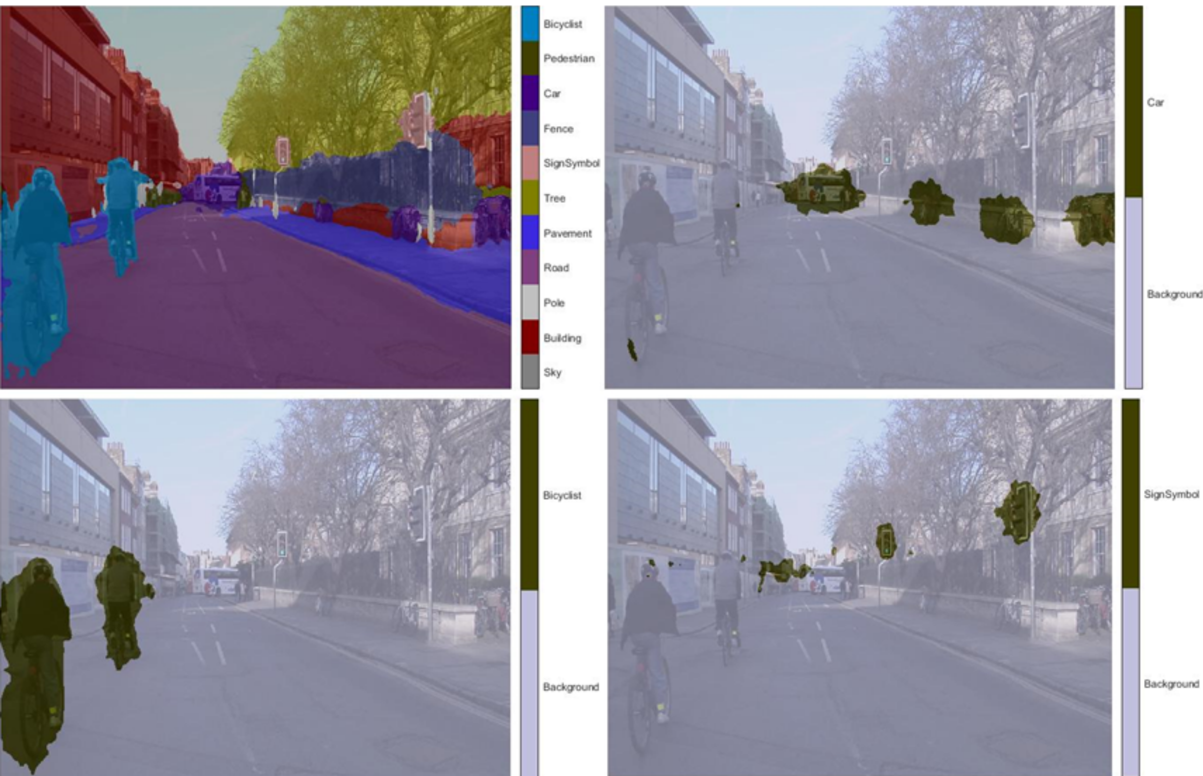

Figura 4.4: Ejemplo de entropía cruzada para multiclase, 'coche', 'ciclista' y 'señal de tráfico'.

4.4.2. *Dice*

A diferencia del caso de entropía cruzada, *Dice* obtiene resultados realmente malos cuando lo aplicamos en multiclase (ver Tabla 4.4). Sin embargo, para objeto-fondo, los resultados son bastantes buenos a nivel *dataset* siguiendo el mismo concepto de mejores valores para objetos más abultados:

Asimismo, a nivel clase (ver Tabla 4.5) apreciamos peores valores para el caso de multiclase y objetos menos voluminosos que para los casos de objeto-fondo seleccionando objetos abultados:

Caso	G. Accuracy	M. Accuracy	M. IoU	W. IoU	M. BFScore
Ciclista	0,99779	0,88201	0,81809	0,99595	0,92764
Peatón	0,99593	0,75976	0,73736	0,99219	0,88362
Coche	0,98818	0,93448	0,89245	0,97748	0,88115
Señal tráfico	0,99239	0,78456	0,74903	0,98570	0,84208
Multiclase	0,68061	0,49176	0,37000	0,53051	0,40069

Tabla 4.4: Representación de los resultados obtenidos para red *Resnet18* y función de coste *Dice* a nivel *dataset*.

Objeto	Caso	Accuracy	IoU	MeanBFScore
Ciclista	Clase	0,76505	0,63840	0,69183
	Multiclase	0,42387	0,31143	0,39790
Peatón	Clase	0,52014	0,47882	0,74584
	Multiclase	0,28983	0,22286	0,48228
Coche	Clase	0,87440	0,79730	0,79591
	Multiclase	0,51499	0,43734	0,49371
Señal tráfico	Clase	0,57090	0,50572	0,70162
	Multiclase	0,30731	0,23327	0,39195

Tabla 4.5: Representación de los resultados obtenidos para red *Resnet18*, función de coste *Dice* y caso de única clase frente multiclase.

En cuanto a lo comentado antes acerca de los resultados de multiclase, de manera visual podemos observar que esta función de coste no es buena para ese caso en la Figura 4.5:

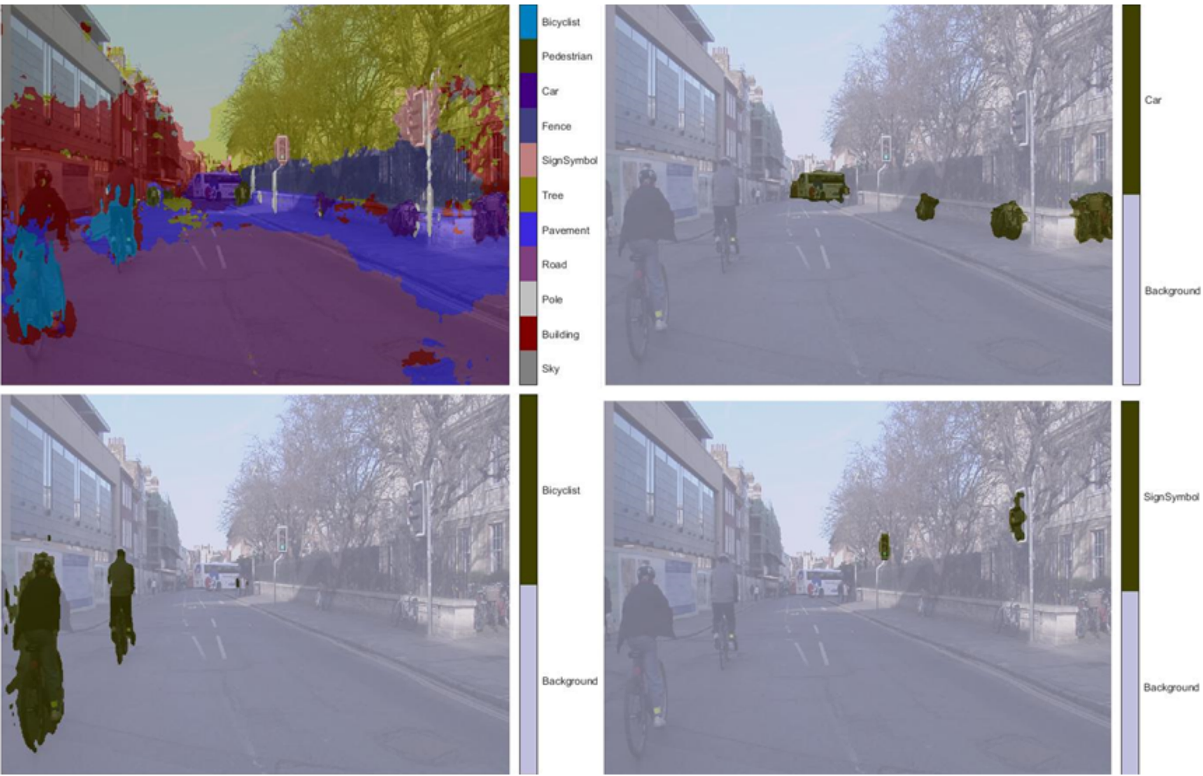

Figura 4.5: Ejemplo de *Dice* para multiclase, 'coche', 'ciclista' y 'señal de tráfico'.

4.4.3. *Tversky*

Respecto a la función de coste *Tversky*, ocurre lo mismo a nivel de conjunto de datos (ver Tabla 4.6), siendo mejores los valores para los objetos más grandes y, por tanto, la precisión es mayor.

Caso	G. Accuracy	M. Accuracy	M. IoU	W. IoU	M. BFScore
Ciclista	0,99683	0,80437	0,74595	0,99427	0,90017
Peatón	0,99607	0,76782	0,74565	0,99245	0,88921
Coche	0,98737	0,89953	0,87908	0,97534	0,86778
Señal tráfico	0,99174	0,71916	0,70590	0,98390	0,81164
Multiclase	0,92098	0,76361	0,69316	0,85633	0,75183

Tabla 4.6: Representación de los resultados obtenidos para red *Resnet18* y función de coste *Tversky* a nivel *dataset*.

Al igual que en las otras funciones de coste, la clase coche obtiene los resultados más óptimos mientras que la señal de tráfico, al ser el menos voluminoso, obtiene los más bajos (ver Tabla 4.7).

Objeto	Caso	Accuracy	IoU	MeanBFScore
Ciclista	Clase	0,68245	0,60068	0,65069
	Multiclase	0,80717	0,70836	0,76765
Peatón	Clase	0,53623	0,49524	0,75890
	Multiclase	0,56712	0,51872	0,78336
Coche	Clase	0,80124	0,77136	0,76969
	Multiclase	0,83182	0,78959	0,78830
Señal tráfico	Clase	0,43893	0,42010	0,63411
	Multiclase	0,44913	0,43047	0,62492

Tabla 4.7: Representación de los resultados obtenidos para red *Resnet18*, función de coste *Tversky* y caso de única clase frente multiclase.

Gráficamente observamos los resultados en la Figura 4.6:

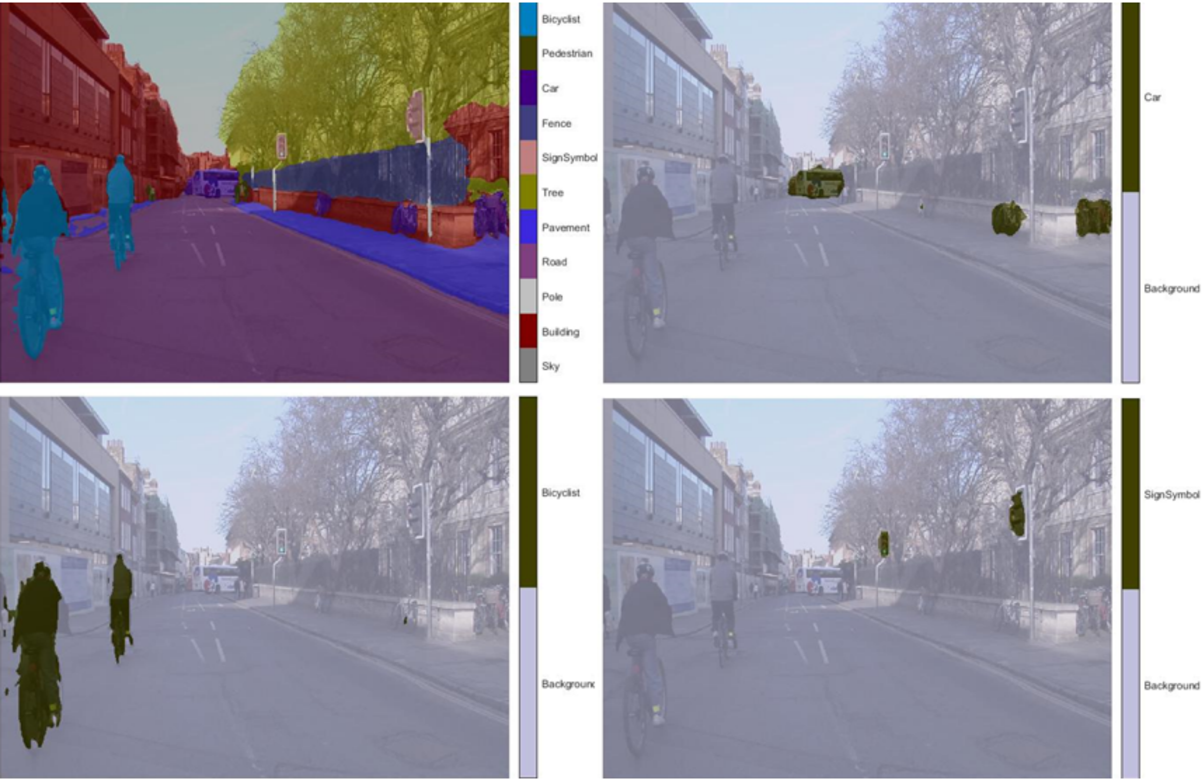

Figura 4.6: Ejemplo de *Tversky* para multiclase, 'coche', 'ciclista' y 'señal de tráfico'.

4.5. Comparativa resultados

Una vez realizados todos los experimentos propuestos, se aprecia como patrón que a medida que los objetos son más voluminosos, mejores resultados se obtienen al seleccionar únicamente ese objeto concreto respecto al fondo, por lo que serán más precisos esos objetos.

También, se aprecia que los valores para multiclase son bastantes buenos tanto para entropía cruzada como para *Tversky*, pero sin embargo *Dice* no funciona para esta casuística.

Finalmente, en la figura 4.7 podremos realizar una mejor comparación para la clase 'ciclista' respecto a como ha clasificado dependiendo de la función de coste seleccionada:

Figura 4.7: Comparación de clasificación de la clase 'ciclista' para cada función de coste (entropía cruzada, *Dice* y *Tversky* respectivamente)

Para entenderlo mejor, la silueta blanca corresponde a los píxeles etiquetados previamente como clase 'ciclista', las zonas moradas oscuras representan lo que es el fondo, las zonas rosa clarito se refieren a aquellos píxeles que debería haber clasificado como 'ciclista' pero no ha sido así (falso negativo) y las zonas verdes son aquellas donde se ha detectado la clase 'ciclista' pero no pertenece a esa clase (falso positivo).

Por tanto, eso quiere decir que elegiremos la función de coste más apropiada según que criterio queramos escoger, por ejemplo si nos interesa asegurarnos que acierta en determinada clase aplicaremos entropía cruzada ya que asegura píxeles adicionales a los reales, pero si por lo contrario no deseamos que ocurra eso, escogeremos la función de coste *Tversky* que es el otro extremo, siendo *Dice* un caso intermedio entre ambas.

5

Conclusiones y Trabajo Futuro

5.1. Conclusiones

En este Trabajo de Fin de Grado se ha tratado de realizar la implementación de la técnica de segmentación semántica para la clasificación de imágenes basado en un entorno de reconocimiento de objetos para conducción autónoma de vehículos.

El estudio del estado del arte ha resultado de gran utilidad puesto que gracias a su análisis, se ha obtenido una serie de conocimientos básicos que nos han permitido introducirnos en el *Deep Learning* con el contexto adecuado. Se ha podido explicar los componentes de las redes neuronales así como profundizar más en las funciones de coste que adquieren gran importancia en este proyecto.

También, gracias a las técnicas de *Transfer Learning* y *Fine Tuning*, hemos podido emplear las redes pre-entrenadas de forma eficiente para no realizar un entrenamiento desde cero, lo que supondría un coste altísimo temporal como computacionalmente.

Respecto a los resultados, como resumen podemos comentar que el hecho de aplicar una u otra función de coste puede hacer que el modelo sea más o menos estricto como hemos visto, a pesar de seguir la idea de ser más precisos para objetos cuánto mayor volumen tengan. Por ello, este hiperparámetro adquiere también cierta importancia al apreciar diferencias para misma imagen y mismo conjunto de datos pero distintas funciones de coste.

Visualmente, los resultados se aprecian más facilmente como hemos mostrado en el anterior capítulo para que apoyados de las tablas se entienda correctamente el trabajo evaluado, siendo buenos resultados en general aunque depende de la aplicación que queramos dar decidiremos si es preferible ser más o menos preciso.

Finalmente, tras haber realizado nuestro proyecto, podremos indicar ciertos trabajos futuros a implementar en la siguiente sección.

5.2. Trabajo Futuro

Una vez vistas las conclusiones relativas a nuestro proyecto, a continuación proponemos una serie de trabajos futuros con el fin de implementar mejoras:

- Durante este trabajo hemos analizado dos redes convolucionales bastantes básicas y comunes para entender su arquitectura y realizar un entrenamiento sencillo del modelo. Sin embargo, si queremos indagar y mejorar nuestro modelo, podemos aplicar redes más profundas tales como *Resnet34* o *Resnet50* si disponemos de capacidad para ello ya que implicaría una mayor precisión y no incrementaría tanto computacionalmente.
- También, sería interesante probar otros *dataset* con más cantidad de datos con el fin de ofrecer más variedad de imágenes e incluso de mayor complejidad, puesto que al fin y al cabo si se emplea para conducción autónoma habría que tener en cuenta muchísimas casuísticas y evitar golpes o accidentes.
- Relacionado con lo anterior, otra posibilidad sería la adición de mayores condiciones al emplear *Data Augmentation*, permitiendo no aumentar el conjunto de datos como tal pero si ofreciendo muchas más variedades.
- Finalmente, otra posible idea futura sería evaluar nuestro modelo en otro conjunto de datos de entorno totalmente distinto, tales como imágenes vía satélite para identificar las distintas zonas del terreno o incluso para casos muy futuristas como el descubrimiento de planetas tales como Marte una vez se disponga de un conjunto de datos considerable.

Bibliografía

[1] Mathworks. Segmentación semántica. https://es.mathworks.com/solutions/image-video-processing/semantic-segmentation.html. Accedido en Septiembre de 2020. 3

[2] Mark Everingham, Luc van Gool, Christopher K. I. Williams, John Winn, and Andrew Zisserman. The pascal visual object classes (voc) challenge. *International Journal of Computer Vision*, 88(2):303–338, June 2010. 4

[3] Carlos García Moreno. ¿qué es el deep learning y para qué sirve? https://www.indracompany.com/es/blogneo/deep-learning-sirve#. Accedido en Diciembre de 2020. 4

[4] Aurélien Géron. *Aprende Machine Learning con Scikit-Learn, Keras y TensorFlow: Conceptos, herramientas y técnicas para construir sistemas inteligentes*. Anaya Multimedia, 2020. 4, 5, 10, 12

[5] Iberdrola. Machine learning: definición, tipos y aplicaciones prácticas. https://www.iberdrola.com/innovacion/machine-learning-aprendizaje-automatico. Accedido en Noviembre de 2020. 5

[6] Gina Mejía. Técnicas de inteligencia artificial. https://www.slideshare.net/ginamejia4/tcnicas-de-inteligencia-artificial, Junio 2017. Accedido en Enero de 2021. 6

[7] Daniel Rodríguez. ¿cuál es la diferencia entre parámetro e hiperparámetro? https://www.analyticslane.com/2019/12/16/cual-es-la-diferencia-entre-parametro-e-hiperparametro/, Diciembre 2019. Accedido en Enero de 2021. 6

[8] Lidgi Gonzalez. Introducción a bias y varianza. https://aprendeia.com/bias-y-varianza-en-machine-learning/, Noviembre 2018. Accedido en Febrero de 2021. 7

[9] Alind Gupta. Regularization in machine learning. https://www.geeksforgeeks.org/regularization-in-machine-learning/, Agosto 2020. Accedido en Febrero de 2021. 7

[10] Alexander Buslaev, Vladimir I. Iglovikov, Eugene Khvedchenya, Alex Parinov, Mikhail Druzhinin, and Alexandr A. Kalinin. Albumentations: Fast and flexible image augmentations. *Information*, 11(2):125, Feb 2020. 8

[11] Vishal Kaushal. Ai/ml@cse: Autoaugment - learning augmentation strategies from data. https://www.cse.iitb.ac.in/~vkaushal/talk/auto-augment/, Julio 2019. Accedido en Noviembre de 2020. 8

[12] Aniththa Umamahesan. Ajuste de hiperparámetros de un modelo con azure machine learning. *docs.microsoft.com/es-es/azure/machine-learning*, 2021. 8

[13] Jaime Durán. Todo lo que necesitas saber sobre el descenso del gradiente aplicado a redes neuronales. https://medium.com/metadatos/todo-lo-que-necesitas-saber-sobre-el-descenso-del-gradiente-aplicado-a-redes-neuronales-19bdbb706a78, Septiembre 2019. Accedido en Febrero de 2021. 9, 13

[14] InteractiveChaos. Mini-batch gradient descent. https://interactivechaos.com/es/manual/tutorial-de-machine-learning/mini-batch-gradient-descent. Accedido en Enero de 2021. 9

[15] Miguel Aguirre, Zulay Franco, and Antonio Pateti. Diseño y simulación de una red neuronal en vhdl y su aplicación en filtrado de un electrocardiograma. *Universidad, Ciencia y Tecnología*, 14:261 – 268, 12 2010. 11

[16] Vicente Rodríguez. Conceptos básicos sobre redes neuronales. https://vincentblog.xyz/posts/conceptos-basicos-sobre-redes-neuronales, Octubre 2018. Accedido en Enero de 2021. 11, 15

[17] Jason Brownlee. Loss and loss functions for training deep learning neural networks. https://machinelearningmastery.com/loss-and-loss-functions-for-training-deep-learning-neural-networks/, Enero 2019. Accedido en Febrero de 2021. 12

[18] Jeremy Jordan. An overview of semantic image segmentation. https://www.jeremyjordan.me/semantic-segmentation/, Mayo 2018. Accedido en Febrero de 2021. 12

[19] Deniz Erdogmus Salehi, Seyed Sadegh Mohseni and Ali Gholipour. Tversky loss function for image segmentation using 3d fully convolutional deep networks. In *International Workshop on Machine Learning in Medical Imaging*, pages 379–387. Springer International Publishing, 2017. 13

[20] Juan Ignacio Bagnato. *Aprende Machine Learning en Español: Teoría + Práctica Python*. N/A, 2020. 14, 16

[21] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning (Adaptive Computation and Machine Learning series)*, chapter 'Convolutional Networks'. The MIT Press, 2016. 14

[22] Jesús Vieco. Red convolucional con pytorch. https://cleverpy.com/red-convolucional-pytorch/. Accedido en Marzo de 2021. 15

[23] Michal Zejmo, Marek Kowal, Józef Korbicz, and Roman Monczak. *Advanced Solutions in Diagnostics and Fault Tolerant Control*, chapter 'Nuclei Recognition Using Convolutional Neural Network and Hough Transform', pages 316–327. Springer, 2018. 15

[24] Benny Prijono. Student notes: Convolutional neural networks (cnn) introduction. https://indoml.com/2018/03/07/student-notes-convolutional-neural-networks-cnn-introduction/, Marzo 2018. Accedido en Marzo de 2021. 16

[25] https://cdn-images-1.medium.com/max/1600/1*uAeANQIOQPqWZnnuH-VEyw.jpeg. Accedido en Marzo de 2021. 16

[26] Gabriel J. Brostow, Jamie Shotton, Julien Fauqueur, and Roberto Cipolla. Segmentation and recognition using structure from motion point clouds. In *ECCV (1)*, pages 44–57, 2008. 18

[27] Adrian Rosebrock. *Deep Learning for Computer Vision with Python*. PyImageSearch, 2019. 18

[28] Ramin Nabati. Adv. pytorch: Modifying the last layer. `https://raminnabati.com/2020/06/adv.-pytorch-modifying-the-last-layer/`, Junio 2020. Accedido en Enero de 2021. 19

[29] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015. 19, 20

[30] Xiangyu Zhang, Jianhua Zou, Kaiming He, and Jian Sun. Accelerating very deep convolutional networks for classification and detection. *CoRR*, abs/1505.06798, 2015. 20, 21

[31] Mathworks. Semantic segmentation using deep learning. `https://es.mathworks.com/help/deeplearning/ug/semantic-segmentation-using-deep-learning.html`. Accedido en Junio de 2020. 21

[32] Steve Eddins. Semantic segmentation using deep learning. `https://blogs.mathworks.com/deep-learning/2018/06/08/semantic-segmentation-using-deep-learning/`, Junio 2018. Accedido en Junio de 2020. 21

[33] Tilo Burghardt, Dima Damen, Walterio W. Mayol-Cuevas, and Majid Mirmehdi, editors. *British Machine Vision Conference, BMVC 2013, Bristol, UK, September 9-13, 2013*. BMVA Press, 2013. 25, 26

[34] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. Imagenet large scale visual recognition challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015. 27