Trabajo Fin de Máster

Detección y segmentación de objetos en imágenes panorámicas

Object detection and segmentation on panoramic images

Autor

Alejandro de Nova Guerrero

Directores

José Jesús Guerrero Campo
Samuel Bruno Berenguel Baeta

Escuela de Ingeniería y Arquitectura

2021

Resumen

Las imágenes panorámicas permiten representar en una sola imagen la totalidad de la escena, pudiendo obtener el contexto completo de los objetos que se encuentran en ella. Esto es un gran aliciente para utilizarlas en tareas de reconocimiento de objetos en escenas de interior. Sin embargo, el uso de imágenes panorámicas presenta algunos inconvenientes que deben ser abordados, como la distorsión inherente a los modelos de proyección o la falta de conjuntos de datos masivos y etiquetados para utilizar técnicas de *deep learning*. Por estos motivos, actualmente existen pocos trabajos que utilizan imágenes panorámicas para realizar tareas de detección y segmentación semántica de objetos.

En este trabajo se ha desarrollado una red neuronal totalmente convolucional basada en la red neuronal *Panoramic BlitzNet* para detección y segmentación semántica de objetos en imágenes panorámicas de interior. El propósito del proyecto es el de programar *from scratch*, con código propio, la red neuronal utilizada en Panoramic BlitzNet. Esta red es una adaptación de BlitzNet, permitiendo usar imágenes panorámicas en lugar de imágenes convencionales. Además, cuenta con licencia GPL con uso libre sólo para investigación y un código desactualizado, implementado en Tensorflow 1.x.

La arquitectura de la red está basada en una estructura *encoder-decoder* con un extractor de características que permite codificar la imagen a distintas escalas para realizar la detección y segmentación de objetos de distinto tamaño. Ambas tareas comparten la mayor parte de la estructura de la red facilitando el entrenamiento conjunto del modelo. Además, se han sustituido las convoluciones estándar por convoluciones equirectangulares para trabajar con la distorsión que introducen las imágenes panorámicas de 360 grados.

El modelo ha sido implementado en Python utilizando principalmente la librería Tensorflow 2.x. El entrenamiento y evaluación ha sido realizado sobre el *dataset* SUN360, de uso libre, etiquetado en otros trabajos para permitir el entrenamiento de redes neuronales con imágenes panorámicas. Los resultados obtenidos son similares a los originales, cumpliendo uno de los objetivos de este proyecto: crear una implementación propia de una red neuronal para tareas de detección y segmentación semántica en imágenes panorámicas de interior.

Abstract

Panoramic images can represent the whole scene in just one image, allowing us to obtain the complete context from the objects represented in the image. This is a huge incentive to use them for object recognition tasks on indoor scene images. However, panoramic images have some drawbacks that must being approached, like the inherent distortion of the projection models or the lack of masive labeled datasets for panoramic images. For these reasons, there are only a few works using panoramic images for semantic segmentation and object detection tasks.

In this work we have developed a fully convolutional neural network based on Panoramic BlitzNet, which is a neural network for semantic segmentation and object detection tasks on indoor scene images. The main goal of this work is programming from scratch, with own code, the neural network used in Panoramic BlitzNet. This neural network is an adaptation from BlitzNet, allowing the use of panoramic images instead of conventional images. In addition, BlitzNet has a GPL license allowing free use for research work only. Moreover, its code is implemented in Tensorflow 1.x and it is outdated.

The network architecture is based on an encoder-decoder structure with a feature extractor that allows codifying the image to different scales to perform multi-scale detection and semantic segmentation. Both tasks share most of the structure, making easier the joint training. Besides, the standard convolutions used in BlitzNet have been changed for equirectangular convolutions in order to deal with the distortion on the 360 degrees panoramic images.

The model has been implemented in Python using as main library Tensorflow 2.x. The training and evaluation has been done using SUN360 dataset, free to use, labeled in other works to allow the use of panoramic images. The results obtained are quite similar to the original ones, satisfying one of the goals of this project: create an own implementation of a neural network for semantic segmentation and object detection tasks on indoor scene panoramic images.

Agradecimientos

Quiero agradecer a mis directores, Bruno y Josechu por su tiempo, dedicación y ayuda. Agradezco especialmente a Julia, que pese no ser directora se ha implicado en todo momento y ha hecho posible que este proyecto salga adelante.

Tabla de contenidos

Lista de figuras

Lista de tablas

Capítulo 1

Introducción

1.1. Motivación

Para los seres humanos el sentido de la vista es el más importante a la hora de obtener información del entorno o de imágenes. La visión por computador es el conjunto de herramientas y métodos que permiten obtener y procesar imágenes del mundo real para poder ser utilizadas por ordenadores. Dentro de este campo de estudio existen tareas muy diversas como reconstrucción 3D, detección de caras, detección de tumores, etc.

Entre estas aplicaciones, una de las más estudiadas a lo largo de los años ha sido el reconocimiento de objetos, que consiste en identificar y localizar objetos dentro de una imagen. El objetivo principal es hacer que una máquina sea capaz de emular una tarea que para los humanos es trivial: entender las características y propiedades de los objetos para ser capaz de distinguirlos correctamente. Entre las aplicaciones más comunes del reconocimiento de objetos se encuentran los sistemas de guiado y seguridad para vehículos autónomos o sistemas de seguimiento de objetos.

Si bien para los humanos es sencillo reconocer visualmente un objeto, esto conlleva grandes dificultades para una máquina. En imágenes, un mismo objeto puede presentar distintos tamaños, formas, orientaciones o incluso cambios en la iluminación, dando lugar a grandes cambios a nivel de píxel para un mismo objeto. Estas variaciones complican en gran medida que las máquinas puedan llegar a realizar tareas de reconocimiento. Sin embargo, la aparición de modelos basados en aprendizaje profundo o *deep learning*, en especial, las redes neuronales convolucionales revolucionaron el estado del arte en cuanto a reconocimiento de objetos por su capacidad de aprender de forma automática las características inherentes a los objetos de la escena.

Históricamente, la segmentación y detección de objetos con redes neuronales se ha realizado con imágenes de cámaras convencionales, cuyo campo de visión es reducido (~40º). Este tipo de

imágenes no permite conocer el contexto de un entorno tal y como lo hacen los humanos (el campo de visión del ojo humano es de aproximadamente 120º). Por este motivo, se empezó a estudiar recientemente el uso de imágenes panorámicas, las cuales permiten representar la totalidad de la escena en una única foto (360º). Sin embargo, debido a la distorsión que presentan este tipo de imágenes omnidireccionales, se requieren desarrollar nuevas técnicas que permitan su tratamiento. Además, por su novedad, aún no existen datasets de la misma dimensión que los de imágenes convencionales, lo que dificulta su explotación con redes neuronales.

1.2. Conceptos iniciales

Previamente a entrar en detalles del proyecto, se van a explicar algunos conceptos básicos relacionados con las redes neuronales y el reconocimiento de objetos, así como con las imágenes panorámicas, ya que son la base del funcionamiento del modelo propuesto.

1.2.1. Perceptrón y perceptrón multicapa

Antes de describir qué es y cómo funciona una red neuronal convolucional es necesario hablar sobre el concepto de perceptrón. Un perceptrón [1] es un algoritmo basado en el funcionamiento del cerebro humano, teniendo como objetivo aprender y tomar decisiones por sí mismo.

El perceptrón es la unidad básica de una red neuronal y consta de cuatro partes: datos de entrada, pesos asociados a cada entrada, un sesgo y una función de activación. El funcionamiento de este modelo, representado en la Figura 1.1, consiste en multiplicar cada dato de entrada por su peso asociado y sumar todos los términos junto con el sesgo, dando lugar a un valor que se pasa a través de la función de activación para obtener la salida del perceptrón (o), como se expone a continuación:

$$o = f(x) = f\left(b + \sum_{\forall i} w_i x_i\right) \qquad (1.1)$$

siendo w_i los pesos asociados a cada conexión neuronal, x_i las entradas, b el sesgo y f la función de activación.

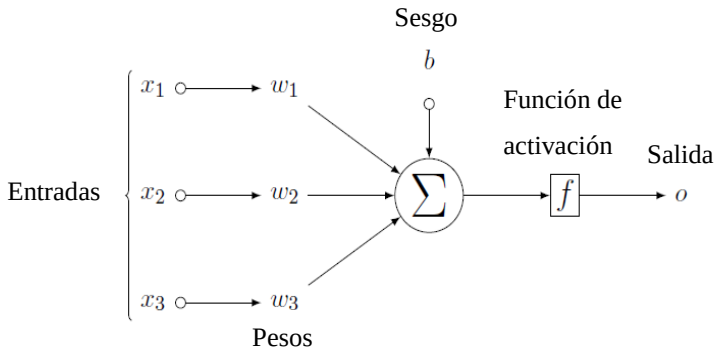


Figura 1.1: Esquema del funcionamiento de un perceptrón.

Si bien el perceptrón es la unidad básica de una red neuronal y a la vez la red neuronal más simple, una de las redes neuronales más comunes es el perceptrón multicapa [2] (MLP). Se trata de una red neuronal formada por al menos tres capas: una de entrada, una de salida y al menos una capa intermedia. Estas capas están formadas por perceptrones conectados con todos los perceptrones de la capa anterior y siguiente, por lo que este tipo de red se suele denominar completamente conectada o *fully connected*. En la Figura 1.2 se muestra un ejemplo de este tipo de red con 5 capas.

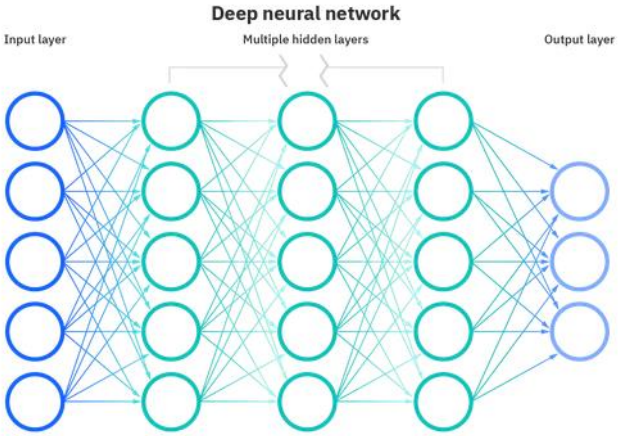


Figura 1.2: Ejemplo de una MLP de 5 capas. Imagen obtenida de [3].

El proceso de aprendizaje de esta red neuronal consiste en ajustar los pesos y sesgos de cada perceptrón, o neurona, con el objetivo de minimizar una función de coste. El proceso de minimización de esta función y la actualización de los parámetros de la red se realiza mediante la propagación de errores o *backpropagation*. Este algoritmo calcula el gradiente de la función de coste a través de sus derivadas parciales respecto de cada uno de los parámetros a optimizar en la red, permitiendo así variar los parámetros para minimizar la función de coste.

A continuación se van a introducir las funciones de activación presentes en el perceptrón. Existen diversos tipos de funciones de activación, tanto lineales como no lineales, aunque sólo se explicarán las más utilizadas: *ReLU*, *sigmoid* y *softmax*.

La función ReLU (*Rectified Linear Units*), representada en la Figura 1.3a, consiste en una función en la que la salida toma valor cero si la entrada es menor que cero. En caso contrario, la salida toma el valor de la entrada. Esta función es la más utilizada en capas intermedias de la red neuronal y se define como sigue:

$$ReLU(z) = \text{máx}(0, z) \tag{1.2}$$

donde z es el valor de la entrada a la función de activación.

La función sigmoid, representada en la Figura 1.3b, consiste en una función cuya salida está comprendida en el intervalo [0, 1], por lo que resulta especialmente útil en la capa final de una red neuronal para tareas de clasificación multi-etiqueta en los que las clases son no excluyentes, por lo que la suma de probabilidades de todas las clases no tiene por qué sumar 1. En estos modelos la salida de la red neuronal consiste en un vector de dimensión 1xn, siendo *n* el número de clases posibles a clasificar, por lo que con esta función se puede obtener una probabilidad entre 0 y 1 para cada una de las posibles clases. La función se define como sigue:

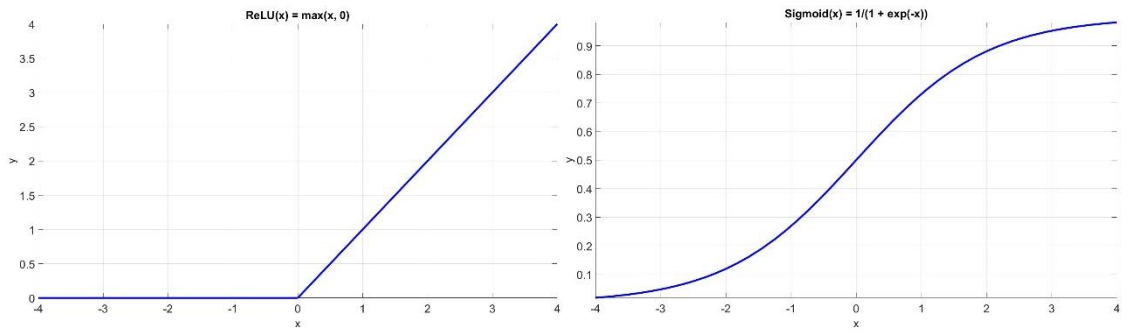
$$sigmoid(z) = \frac{1}{1 + e^{-z}} \tag{1.3}$$

donde z es el valor de la entrada a la función de activación.

La función softmax es similar a la función sigmoid, ya que está comprendida en el intervalo [0, 1], por lo que también es muy utilizada en labores de clasificación. Sin embargo, a diferencia de la función anterior, esta se utiliza en problemas de clasificación multi-clase en los que las diferentes clases son mutuamente excluyentes. Es decir, la suma de las probabilidades de todas las clases da como resultado 1. El cálculo de la función se realiza de la siguiente forma:

$$softmax(z_j) = \frac{e^{z_j}}{\sum_{i=1}^{K} e^{z_i}} \;\; para\; j = 1, \ldots, K \tag{1.4}$$

donde z_j es el valor de la entrada de la clase para la que se calcula el softmax y *K* el número total de clases.



(a) *(b)*

Figura 1.3: (a) Función ReLU. (b) Función sigmoid.

1.2.2. Redes neuronales convolucionales

Las redes neuronales convolucionales [4] (CNNs) siguen el mismo funcionamiento que las redes neuronales ya explicadas, utilizando pesos, sesgos y funciones de activación. Una de las diferencias es que la entrada de una CNN es o bien una imagen o bien una representación matricial de los datos de entrada.

Otra diferencia a destacar de este tipo de red neuronal es el uso de capas convolucionales, en las cuales se aplican convoluciones. Una convolución es una operación matemática que consiste en aplicar un filtro o *kernel* de convolución sobre un dato de entrada (matriz o imagen). Este filtro consiste en una matriz, normalmente de tamaño 3x3 o 5x5, con el centro como punto de anclaje. Este filtro se pasa por cada uno de los píxeles de la imagen para dar como resultado en el punto de anclaje la suma de cada valor del filtro por el valor del píxel sobre el que se sitúa el filtro. En la Figura 1.4 se observa un ejemplo de convolución sobre una matriz que puede ser extrapolable a una imagen.

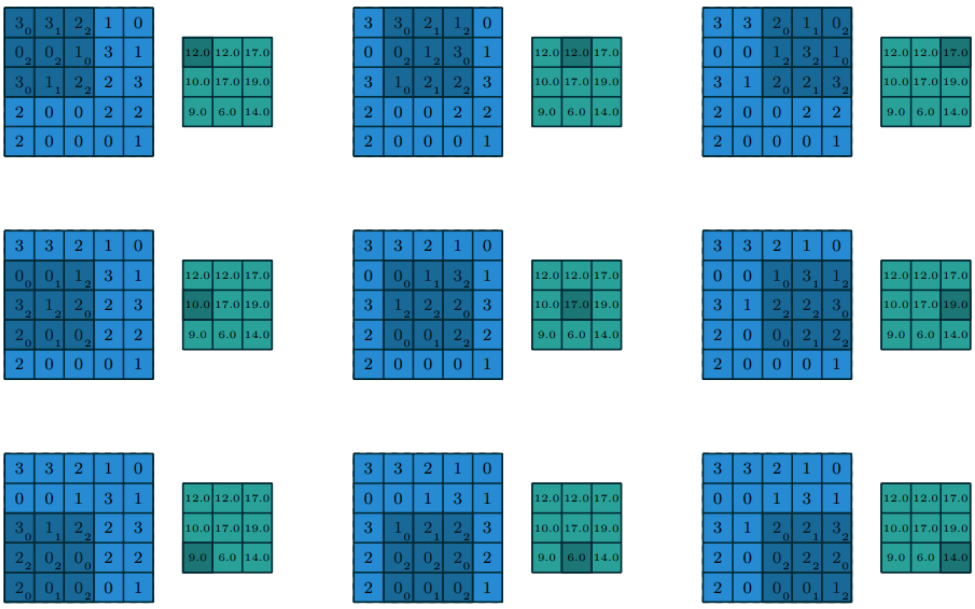


Figura 1.4: Ejemplo de una convolución aplicando un kernel 3x3 sobre una matriz de 5x5 píxeles. Imagen obtenida de [5].

Esta es la definición de una convolución sobre una entrada de 2 dimensiones. Sin embargo, tanto las imágenes de entrada como las imágenes intermedias de la red neuronal presentan una profundidad o número de canales mayor que 1 (3 en el caso de imágenes RGB). Por este motivo, en las capas convolucionales de una CNN se aplican filtros tridimensionales de una anchura y altura determinada, así como una profundidad igual a la de la imagen sobre la que se aplica. Se va trasladando el filtro a lo largo de la altura y anchura de la entrada obteniendo así un número de salidas equivalente a la profundidad de la imagen. Finalmente, se suman y se añade un sesgo

para obtener así un canal de la salida. De esta forma, aplicando una cantidad de filtros específica se obtiene una salida con un número de canales idéntico al número de filtros utilizados.

Las capas convolucionales dan como resultado un mapa de características que contiene información referente a diferentes propiedades de lo que aparece en la imagen, de forma que la red neuronal puede aprender características inherentes a diferentes objetos (contornos, texturas, patrones, partes de objetos…). El tamaño de la salida de cada capa depende del tamaño de la entrada, del número de filtros que se apliquen, del tamaño del filtro, del paso o *stride* del filtro y del relleno o *padding* que se aplique. Este último consiste en la posible adición de píxeles alrededor del borde de la imagen para evitar la pérdida de información en los bordes y la reducción espacial de la salida al aplicar un filtro. Por otro lado, también se utilizan otros tipos de capas como las de reducción o *pooling* que reducen el tamaño espacial del mapa de características obtenido tras la convolución. Existen distintos tipos de reducción aunque sólo se comentarán los más utilizados: *max pooling* y *average pooling*. El primero devuelve el valor máximo de los píxeles sobre los que se encuentra el kernel, mientras que el segundo devuelve la media de valores de los píxeles sobre los que se encuentra el kernel, tal y como se muestra en la Figura 1.5.

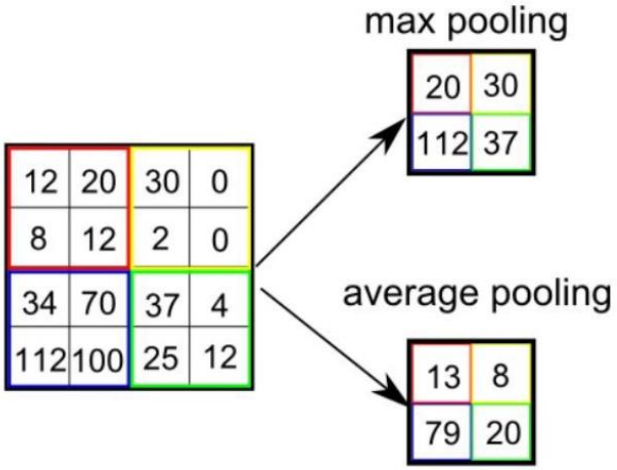


Figura 1.5: Ejemplo de una capa max pooling y una capa average pooling. Imagen obtenida de [6].

1.2.3. Reconocimiento de objetos

Una vez explicados los conceptos básicos de las redes neuronales se va a abordar una de las tareas principales de este trabajo, el reconocimiento de objetos. En la visión por computador existen distintas tareas relacionadas con el reconocimiento de objetos, como se muestra en la Figura 1.6.

La actividad más básica dentro del reconocimiento de objetos es la clasificación de imágenes. Esta tarea consiste en enumerar todas las clases de objetos que aparecen en una imagen o, si sólo hay un objeto por imagen, obtener la categoría a la que pertenece. Por otro lado, la localización

de objetos consiste en localizar cada uno de los objetos de la imagen a través de *bounding boxes*, que son recuadros que encierran el objeto que se está localizando. La detección de objetos mezcla estos dos conceptos, siendo su objetivo localizar y clasificar cada uno de los objetos que aparecen en una imagen. En cuanto a las tareas de segmentación, estas consisten en clasificar cada píxel de la imagen, por categoría, en la segmentación semántica, o por objeto, en la segmentación por instancias.

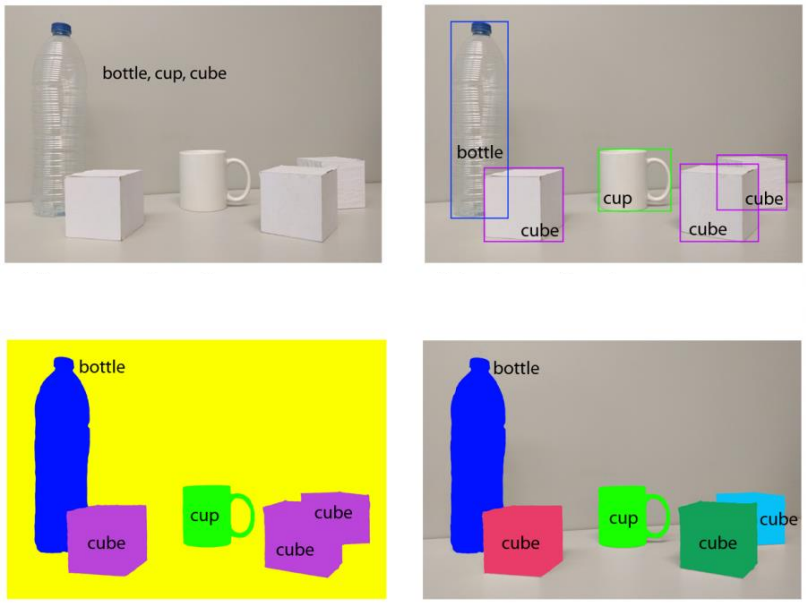
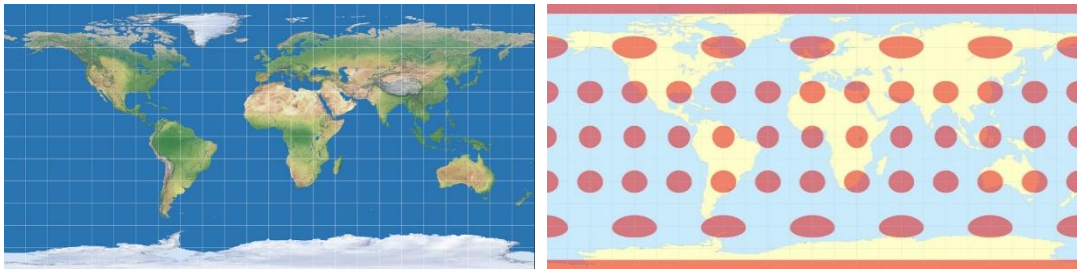


Figura 1.6: Aplicaciones del reconocimiento de objetos. De arriba a abajo y de izquierda a derecha: clasificación de imágenes, detección de objetos, segmentación semántica y segmentación por instancias. Imagen obtenida de [7].

1.2.4. Imágenes panorámicas y *EquiConvs*

Por último, se tratarán las imágenes panorámicas y sus propiedades básicas, ya que son el elemento de entrada al modelo propuesto. Las imágenes panorámicas equirectangulares son un modelo de proyección omnidireccional donde se proyecta el entorno sobre la superficie de una esfera, la cual luego es desplegada en la imagen final. Esto permite tener un campo de visión de 360º horizontal y 180º vertical en una única imagen. Sin embargo, este tipo de proyección presenta grandes distorsiones cerca de los polos de la esfera, como se puede observar en la Figura 1.7.



(a) *(b)*

Figura 1.7: (a) Proyección equirectangular en un mapamundi. Imagen obtenida de [8] *(b) Indicatriz de Tissot mostrando la distorsión en la proyección equirectangular. Imagen obtenida de* [8].

Por otro lado, si bien se ha estudiado el uso de convoluciones normales con panoramas, en algunos trabajos como en el de Fernández et. al. [9] se propuso el uso de convoluciones equirectangulares o EquiConvs. Este tipo de convoluciones presentan un kernel de tamaño y forma variable, el cual le permite adaptarse a la distorsión de la imagen según el punto sobre el que se sitúe, de acuerdo a lo que se muestra en la Figura 1.8.

Figura 1.8: Diferentes kernels de EquiConvs sobre un panorama 360º. Nótese que la forma del kernel en la proyección equirectangular varía según la posición para adaptarse a la distorsión, mientras que en la esfera aparece con forma cuadrada. Imagen obtenida de [9].

El kernel de la EquiConv se define como un parche sobre la esfera parametrizado por su tamaño angular α y su resolución r. Este kernel se rota y aplica a lo largo de la esfera y la posición central del kernel se define por sus coordenadas esféricas (Φ, θ), tal y como se observa en la Figura 1.9.

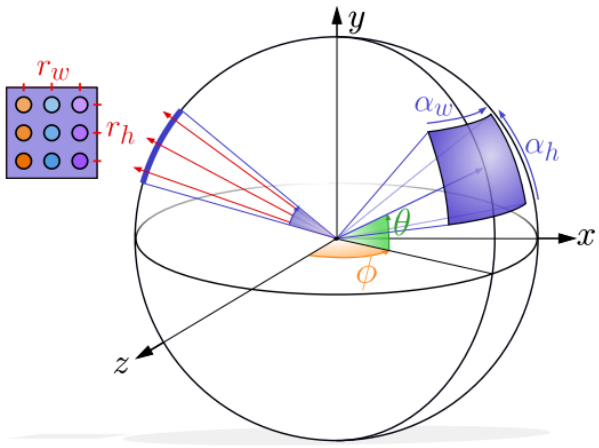


Figura 1.9: Parametrización de la EquiConv. Nótese que para kernel cuadrado $\alpha_w = \alpha_h = \alpha$ y $r_w = r_h = r$. Imagen obtenida de [9].

De acuerdo a [9], para crear el kernel de la EquiConv se deforma la forma del kernel para adaptarse a la proyección equirectangular. Para lograr esto se utilizan *offsets* fijos basados en la distorsión de la imagen equirectangular, los cuales varían en el eje vertical pero son constantes en el eje horizontal para cada posición en el eje vertical. Para obtener las coordenadas de los píxeles distorsionados a partir de los originales se definen en primer lugar las coordenadas de todos los elementos del kernel, el cual se rota después al punto de la esfera sobre el que ha de aplicarse. Estos puntos del kernel se definen de la siguiente forma:

$$\hat{p}_{ij} = \begin{bmatrix} \hat{x}_{ij} \\ \hat{y}_{ij} \\ \hat{z}_{ij} \end{bmatrix} = \begin{bmatrix} i \\ j \\ d \end{bmatrix} \tag{1.5}$$

donde i y j son enteros en el rango $\left[-\frac{r-1}{2}, \frac{r-1}{2}\right]$ y d es la distancia desde el centro de la esfera al kernel. Para cubrir todo el campo visual del kernel (α) se define este parámetro d como sigue:

$$d = \frac{r}{2\tan(\frac{\alpha}{2})} \tag{1.6}$$

A continuación se proyecta cada punto sobre la superficie de la esfera normalizando el vector $\hat{p}_{ij}$ y se rota el kernel para que el centro del mismo coincida con el punto sobre el que se aplica según la siguiente ecuación:

$$p_{ij} = \begin{bmatrix} x_{ij} \\ y_{ij} \\ z_{ij} \end{bmatrix} = R_y(\Phi_{0,0}) R_x(\theta_{0,0}) \frac{\hat{p}_{ij}}{|\hat{p}_{ij}|} \tag{1.7}$$

donde $R_a(\beta)$ es la matriz de rotación de un ángulo β en torno al eje a y $\Phi_{0,0}$ y $\theta_{0,0}$ son las coordenadas esféricas del centro del kernel, definidas como sigue:

$$\Phi_{0,0} = \left(u_{0,0} - \frac{W}{2}\right)\frac{2\pi}{W} \; ; \; \theta_{0,0} = -\left(v_{0,0} - \frac{H}{2}\right)\frac{\pi}{H} \tag{1.8}$$

siendo $(u_{0,0}, v_{0,0})$ el punto de la imagen equirectangular sobre el que se aplica el kernel y *W* y *H*, respectivamente, la anchura y altura de la imagen equirectangular en píxeles.

Finalmente se proyectan los elementos del kernel a la proyección equirectangular. Primero convirtiendo las coordenadas en la esfera a coordenadas esféricas:

$$\Phi_{ij} = arctan\left(\frac{x_{ij}}{z_{ij}}\right) \; ; \; \theta_{ij} = arcsen\left(y_{ij}\right) \tag{1.9}$$

Y después, convirtiéndolas al dominio 2D de la imagen equirectangular:

$$u_{ij} = \left(\frac{\Phi_{ij}}{2\pi} + \frac{1}{2}\right)W \; ; \; v_{ij} = \left(\frac{\theta_{ij}}{\pi} + \frac{1}{2}\right)H \tag{1.10}$$

En [9] no sólo se definió el concepto de EquiConv sino que también se demostró que su uso permite aprender patrones más generales reduciendo el sobreajuste y aumentando la robustez en cuanto a la traslación y rotación de la posición de la cámara, por lo que es ideal para panorámicas hechas por una persona con una cámara en mano. Este es uno de los motivos por los que este tipo de convolución es un gran avance en cuanto a tareas de detección de objetos y segmentación semántica en imágenes panorámicas mediante el uso de redes convolucionales.

1.3. Estado del arte

El reconocimiento de objetos ha sido uno de los campos de estudio más importantes de la visión por computador en cuanto a esfuerzo en investigación se refiere, habiendo varias décadas de trabajos relacionados. Sin embargo, en este caso sólo se va a hablar de la historia del reconocimiento de objetos desde que se comenzaron a utilizar técnicas de deep learning para esta tarea. En concreto, el trabajo más importante y que cambió completamente la percepción sobre el reconocimiento de objetos fue AlexNet [10], una red neuronal convolucional que incorporó una gran cantidad de propuestas como el uso de funciones de activación ReLU, múltiples GPUs, *data augmentation* y *dropout*. Este modelo consiguió ganar con una gran ventaja el ILSVRC [11] (*ImageNet Large Scale Visual Recognition Challenge*) en 2012, una de las competiciones más importantes a lo largo de la historia para comprobar la evolución de las técnicas de visión por computador en cuanto a reconocimiento de objetos se refiere. Esto supuso un cambio total en el modo de abordar la investigación en este ámbito, pues todas las técnicas tradicionales fueron sustituidas por el uso de redes neuronales convolucionales.

Durante los años posteriores, surgieron numerosos modelos para tareas de detección y segmentación de objetos, a cada cual con mejores resultados. En cuanto a la detección de objetos, se pueden distinguir dos grupos: detectores de dos etapas, que requieren primero generar las propuestas de bounding boxes para luego utilizar las características de estas propuestas para clasificar los objetos, y detectores de una etapa que realizan la extracción de propuestas y clasificación en una misma etapa. Dentro de los modelos más importantes basados en dos etapas se encuentran los siguientes:

R-CNN [12] surgió en 2013 y se trataba de una red neuronal convolucional a la que se alimentaba con aproximadamente 2000 propuestas de bounding boxes extraídas de la imagen mediante un algoritmo de búsqueda selectiva para llevar a cabo la detección de objetos.

En 2015 surgió *Fast R-CNN* [13], con un funcionamiento similar a su predecesora pero alimentando la red directamente con la imagen y extrayendo un mapa de características en el que identificar las diferentes propuestas de bounding boxes para luego realizar la clasificación, solucionando así los problemas de velocidad durante el entrenamiento e inferencia de R-CNN. Sin embargo, este modelo seguía siendo lento a la hora de realizar el proceso de inferencia debido al algoritmo de búsqueda selectiva para generar las propuestas, por lo que durante 2015 surgió una nueva variante, *Faster R-CNN* [14], que sustituyó este algoritmo por una red neuronal que generase las propuestas de bounding boxes, mejorando así el tiempo durante el proceso de inferencia.

Estos modelos sólo trabajaban con detección de objetos por lo que en 2017 apareció *Mask R-CNN* [15], una extensión de Faster R-CNN con una rama para segmentación por instancias paralela a la existente para detección de objetos.

Por otro lado, los modelos basados en una etapa con mayor importancia son los que se describen a continuación:

SSD [16] (*Single Shot MultiBox Detector*) surgió en 2016 para realizar detección de objetos en una sola etapa mediante el uso de mapas de características a distintas escalas (*multi-scale detection*), obtenidos a través de un extractor de características y un flujo de *downscaling*. Así como el uso de un sistema de propuestas de bounding boxes similar al utilizado en *MultiBox* [17], de forma que cada mapa de características se divide en una rejilla en la que para cada celda se definen varias bounding boxes con *aspect ratios* variados denominadas *prior boxes* que luego se utilizan para predecir las bounding boxes finales.

BlitzNet [18] apareció en 2017 con un modelo que realizaba de forma conjunta detección y segmentación semántica de objetos mediante una red neuronal con una sola rama que se dividía al final para realizar las dos tareas por separado, siendo la arquitectura similar a la utilizada en

SSD pero añadiendo bloques de deconvolución similares a los utilizados en *DSSD* [19] para mejorar la segmentación semántica y la detección de objetos.

Los modelos anteriores se basaban en el uso de imágenes convencionales con reducido campo de visión, por lo que surgieron algunos trabajos en los que se utilizaron panoramas como entrada del modelo. Aquí destaca Panoramic BlitzNet [20], una red neuronal basada en BlitzNet adaptada al uso de imágenes panorámicas, utilizando además EquiConvs.

1.4. Objetivos y alcance

El objetivo principal del proyecto es implementar en Python, desde cero, la red neuronal completamente convolucional utilizada en BlitzNet para detección y segmentación semántica de objetos, y adaptarla para su aplicación directa en imágenes panorámicas de interior siguiendo la propuesta de Panoramic BlitzNet. Además, la red neuronal BlitzNet presenta una licencia GPL, lo que permite su uso para investigación, pero dificulta posibles usos comerciales de Panoramic BlitzNet, y está basada en la librería para deep learning TensorFlow 1.x. Por ello, se pretende crear una implementación propia utilizando la última versión de TensorFlow, la cual fue actualizada en 2019 a su versión 2.x. En esta actualización se realizaron una gran cantidad de cambios, mejorando la experiencia de usuario y eficiencia de la librería. Por otro lado, con una programación desde cero, se pretende obtener un código optimizado para las tareas de detección y segmentación semántica de objetos con imágenes panorámicas. Además, una implementación propia permitirá poder distribuir el código de detección y segmentación semántica de objetos en imágenes panorámicas para fines de investigación o licenciarlo con fines comerciales sin tener que depender de código o licencias externas.

Para cumplir con los objetivos propuestos se han planteado una serie de objetivos divididos en tres fases:

1. Formación específica en deep learning mediante cursos online, lectura de artículos de investigación y mediante la realización de una pequeña red neuronal completa para clasificación de objetos utilizando Tensorflow 2.x.
2. Implementación de la red neuronal completamente convolucional:
    - Pre-procesamiento del dataset utilizando data augmentation.
    - Estudio e implementación de la arquitectura de la red neuronal utilizada en Panoramic BlitzNet.
    - Definición de métricas y funciones de coste para el entrenamiento del modelo.
3. Entrenamiento de la red neuronal y evaluación de resultados.

Capítulo 2

Modelo

En este capítulo se describe en detalle la arquitectura del modelo, el cual tiene como objetivo dos tareas: la detección y la segmentación semántica de objetos. Además, se propone una arquitectura que permite trabajar directamente con imágenes panorámicas. Por otro lado, el hecho de tener una sola red neuronal que realice tanto la tarea de detección como la de segmentación permite compartir pesos entre ambas, ayudando a mejorar los resultados obtenidos tal y como se demostró en Blitznet [18].

2.1. Arquitectura del modelo

La arquitectura escogida para la realización de la red neuronal está basada principalmente en la utilizada por BlitzNet, cuyo código implementado en Tensorflow 1.x y Python está publicado en GitHub con licencia GPL. Se han realizado varios cambios para trabajar directamente con imágenes panorámicas frente al uso de imágenes cuadradas en el trabajo original. Estos cambios son principalmente los siguientes: sustitución de convoluciones estándar por EquiConvs y cambios en las dimensiones de las variables del modelo para tener en cuenta el aspect ratio (relación anchura-altura) de las imágenes panorámicas frente al de las imágenes convencionales. Para reconstruir la arquitectura de la red, se ha creado un nuevo código haciendo uso de la librería Tensorflow 2.x, actualizando la implementación realizada en BlitzNet que utilizaba Tensorflow 1.x.

Como se muestra en la Figura 2.1, el modelo comienza con una imagen RGB de entrada de dimensiones HxWxD, siendo *H* la altura, *W* la anchura y *D* la profundidad o número de canales de la imagen. Esta imagen es procesada por un extractor de características o *feature extractor*, el cual consiste en una red neuronal convolucional formada por varios bloques que se encarga de generar mapas de características a partir de la imagen de entrada. Estos mapas de características

son imágenes de pequeña resolución que contienen la información relevante de la imagen original, como pueden ser bordes, color, orientación, textura, partes, etc. de los objetos cuyas características se quieren conocer para entrenar la red neuronal correctamente. Mediante esta extracción de características se reduce la resolución espacial de la imagen de entrada mientras se aumenta el número de canales, o mapas de características. De esta forma, se reduce el coste computacional, tratando una menor cantidad de información pero más relevante. Para este proyecto se utiliza *ResNet50* [21], un extractor de características formado por 50 capas, comúnmente utilizado en tareas de visión por computador debido a su elevado rendimiento.

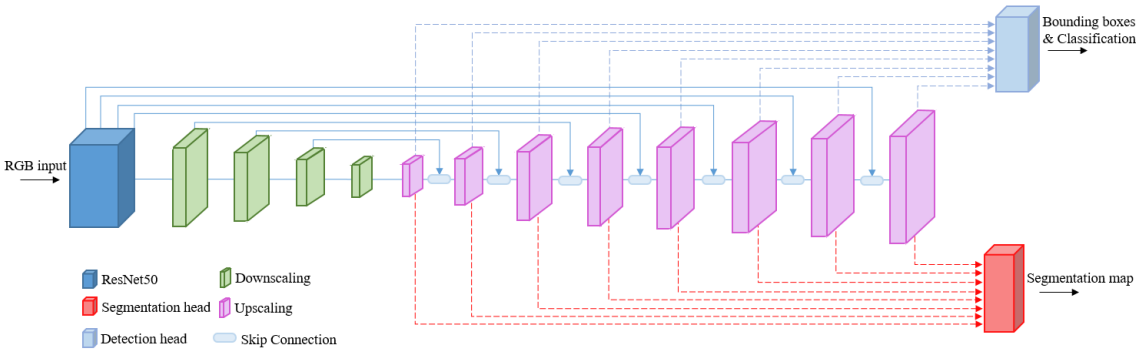


Figura 2.1: Detalle de la arquitectura del modelo propuesto.

La salida del extractor de características, de una resolución de 8x16x2048, presenta una gran reducción de tamaño frente a la imagen de entrada, de resolución 512x1024x3. Posteriormente, siguiendo la estrategia de *downscaling* propuesta en SSD [16], se continúa reduciendo la resolución de los mapas de características de forma iterativa hasta una resolución de 1x2x2048.

Tras esto, pensando especialmente en la segmentación semántica dónde se requiere clasificar a nivel de pixel y siguiendo con el modelo propuesto en DSSD [19], se realiza un aumento de la resolución espacial o *upscaling* de los mapas de características. Esto permite obtener información adicional a distintas escalas, mejorando tanto la detección como la segmentación de objetos, sobre todo en el caso de objetos de pequeño tamaño. Para este aumento de resolución se utilizan capas de deconvolución, que son opuestas a las capas de convolución, además de un bloque denominado *ResSkip Connection*, cuya estructura se puede observar en la Figura 2.2. Este bloque utiliza los mapas de segmentación tanto del downscaling como del upscaling, combinándolos de la siguiente forma: se re-escala el mapa de características que llega del flujo de upscaling al tamaño del mapa que viene del downscaling y se concatenan. Tras esto, se pasan los mapas concatenados por una serie de convoluciones de kernel 1x1, 3x3 y 1x1, respectivamente, para sumar finalmente el resultado obtenido tras la última convolución con el mapa de entrada que se había re-escalado siguiendo una conexión residual como las que se utilizan en ResNet50. Los beneficios de este bloque han sido demostrados en Blitznet [18] en el que explican que los buenos resultados pueden

deberse al uso de conexiones similares a las utilizadas en ResNet50, haciendo la arquitectura más homogénea.

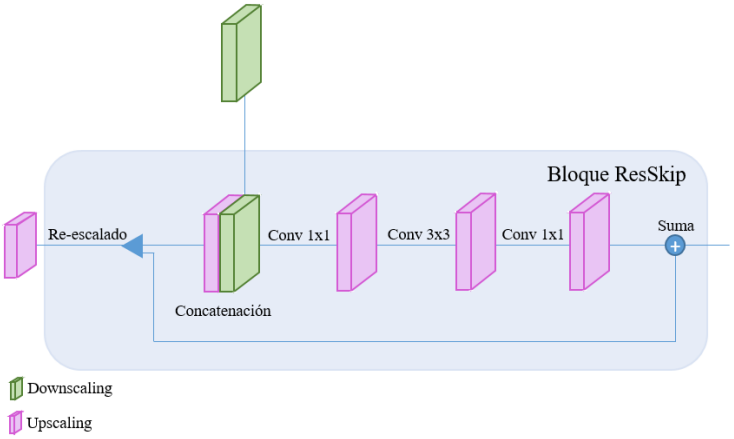


Figura 2.2: Detalle del bloque ResSkip.

Una vez se obtienen los mapas de características del flujo de upscaling se introduce cada uno de ellos como entrada de las ramas de detección y de segmentación. Estos mapas de características presentan distintas resoluciones espaciales, desde 128x256 hasta 1x2, existiendo 512 mapas para cada resolución. A partir de este punto se pasará a utilizar el concepto de mapa de características para referirse a todos los mapas existentes para cada una de las posibles resoluciones espaciales.

2.1.1. Rama de Detección

La detección de objetos se lleva a cabo utilizando todos los mapas de características del flujo de upscaling, en este caso ocho. El procedimiento consiste, para cada mapa de características, dividir el tamaño de la imagen de entrada (512x1024 en este caso) en una malla o rejilla de dimensiones idénticas a la del mapa correspondiente, de forma que aparezca un número total de celdas en la rejilla igual al número de píxeles del mapa de características. Sobre esta rejilla, tal y como se observa en la Figura 2.3, se generan una serie de bounding boxes de referencia, las cuales se denominan *anchor boxes* o prior boxes según la literatura.

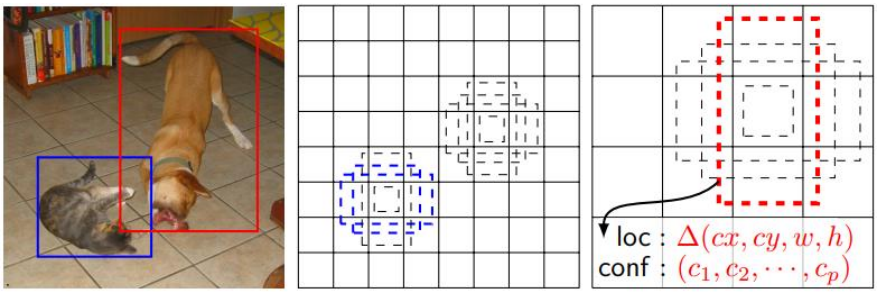


Figura 2.3: Ejemplo de una imagen con ground truth (izquierda), un mapa de características de 8x8 sobre el que se han representado algunas anchor boxes de distintos aspect ratios (centro) y un mapa de características de 4x4 sobre el que se han representado algunas anchor boxes de distintos aspect ratios (derecha). Imagen obtenida de [16].

Estas anchor boxes están definidas por 4 coordenadas: *cx*, *cy*, *w* y *h*, correspondientes al centro, anchura y altura de cada caja. El objetivo principal es que cada una de ellas tenga una posición fija en el mapa de características de forma que aprenda a detectar objetos que se encuentren en sus cercanías. Además, como los objetos a detectar pueden tener distintas formas, en cada posición se generan varias anchor boxes, cada una con un aspect ratio distinto. De esta forma, cada anchor box puede aprender a detectar objetos de una forma concreta en una posición de la imagen concreta. En este caso, y siguiendo con lo propuesto en Panoramic BlitzNet [20], se han utilizado 5 aspect ratios diferentes: 1, 2, 1/2, 3 y 1/3, mostrados en la Figura 2.4 sobre algunos objetos del dataset. Así, en cada celda de la rejilla se generan 10 anchor boxes cuyo centro se corresponde con el centro de la celda (*cx*, *cy*). Estas 10 anchor boxes se dividen en dos grupos de 5 anchor boxes cada grupo, las cuales están definidas por los 5 aspect ratios propuestos. A cada grupo se le asocia una escala con la que se calcula el tamaño de las anchor boxes de ese grupo, quedando definidas las coordenadas *w* y *h* de cada anchor box. Esto se hace para abarcar una mayor cantidad de posibles tamaños de objetos a detectar. Para cada mapa de características se utilizan 2 escalas diferentes, por lo que se acaban generando decenas de miles de anchor boxes con una gran diversidad de tamaños para conseguir detectar la mayor cantidad de objetos posible.

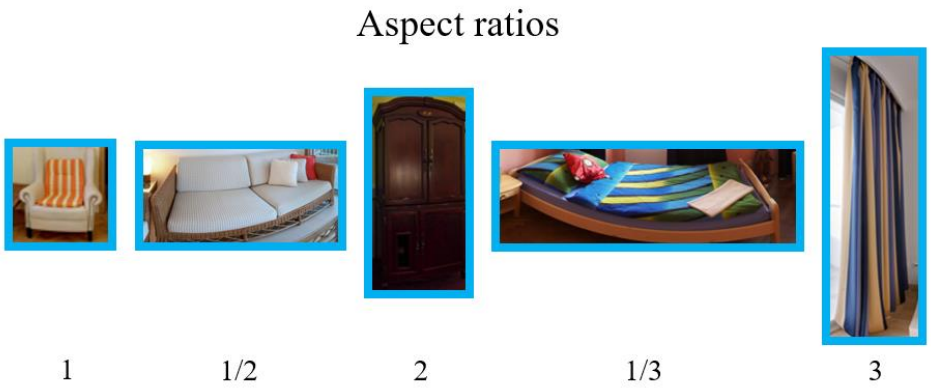


Figura 2.4: Aspect ratios utilizados en las anchor boxes propuestas para detección de objetos sobre algunos de los objetos que aparecen en las imágenes del dataset.

Para la elección de las 2 escalas a utilizar para cada mapa de características se tiene en cuenta que las escalas más pequeñas se utilizan en el caso de la rejilla de mayor dimensión (128x256 en este caso), ya que al dividir la imagen en más celdas se está buscando detectar los objetos más pequeños. Por otro lado, las escalas más grandes se utilizan en el caso de la rejilla de menor dimensión (1x2 en este caso), ya que con esta división de celdas se busca detectar los objetos más grandes que puedan ocupar toda la imagen. Eligiendo la escala mínima, $s_{mín} = 0.02$ y la escala máxima, $s_{máx} = 0.98$ se calculan el resto de escalas para las demás rejillas mediante un espaciado lineal entre $s_{mín}$ y $s_{máx}$. El cálculo de la anchura y altura de cada anchor box se calcula de la siguiente forma:

$$w_{k,i}^{a} = s_{k,i}\sqrt{a_r}\,, i \in [1,2], k \in [1,m]$$
$$h_{k,i}^{a} = \frac{s_{k,i}}{\sqrt{a_r}}, i \in [1,2], k \in [1,m] \quad (2.1)$$

siendo $w_{k,i}^{a}$ la anchura de la anchor box *a* usando la escala *i* asociada al mapa de características *k*, a_r el aspect ratio de la anchor box, $s_{k,i}$ la escala *i* asociada a cada mapa de características y *m* el número total de mapas de características.

Como resultado de este proceso se obtiene para cada mapa de características un número de anchor boxes igual a 10xHxW, siendo *H* y *W* la altura y la anchura del mapa de características, respectivamente. Además, hay que destacar que mientras las dimensiones de los mapas de características se mantengan siempre iguales (vienen definidos por la estructura de la red neuronal), así como los aspect ratios y las escalas que se utilizan, el número y las dimensiones de las anchor boxes generadas serán siempre iguales.

Por otro lado, es necesario recalcar que las anchor boxes sólo sirven como referencia para detectar objetos de forma similar y en posiciones similares. Sin embargo, la red neuronal predice una serie de offsets para cada anchor box, es decir, una serie de coordenadas relativas a las de cada anchor box que refinan la posición y tamaño de la caja inicial para adaptarse al objeto real que está detectando. Mediante el paso de cada mapa de características por una convolución se predicen 4 offsets (δ_{cx}, δ_{cy}, δ_w, δ_h) por cada anchor box generada, por lo que el número de filtros a aplicar en esta convolución es de 4x10 en este caso (4 coordenadas para cada anchor box por 10 anchor boxes generadas en cada celda). Estos offsets, junto con las coordenadas de las anchor boxes (*a*), permiten obtener las coordenadas reales de la bounding box predicha (*p*) a través de las siguientes ecuaciones.

$$p_{cx} = a_{cx} + \delta_{cx}a_w \;\; ; \;\; p_{cy} = a_{cy} + \delta_{cy}a_h$$
$$p_w = a_w exp^{\delta_w} \;\; ; \;\; p_h = a_h exp^{\delta_h} \quad (2.2)$$

Cabe destacar que tal y como están definidas las relaciones entre los offsets y las anchor boxes, una anchor box situada en un extremo del mapa de características podría aprender a detectar un objeto que se encuentra en el otro extremo de la imagen. Sin embargo, esto no es habitual y normalmente al aplicar los offsets el centro de cada anchor box se mantendrá dentro de la celda en la que se encontraba inicialmente.

Adicionalmente, mediante otra convolución seguida de una activación softmax, cuya entrada es cada uno de los mapas de características, se predicen las probabilidades para cada anchor box de pertenecer a cada una de las clases a detectar. En este caso, el número de clases posibles es 14 junto con el *background* de la imagen, el cual hay que tener en cuenta, por lo que el número de

filtros a aplicar en la convolución es 15x10 (15 clases distintas por 10 anchor boxes generadas en cada celda).

Por último, a la hora de realizar el entrenamiento del modelo hay que llevar a cabo un proceso de emparejamiento de las bounding boxes del *ground truth* con las anchor boxes. Es decir, hay que determinar qué anchor boxes corresponden a una bounding box de un objeto etiquetado en la imagen de entrada para entrenar la red adecuadamente. Para esto, se utiliza un parámetro denominado *Intersection Over Union* (IoU), el cual evalúa la superposición entre dos áreas. De esta forma, a partir del área de una bounding box del ground truth (*A)* y el área de una anchor box (*A'*) se define este parámetro como la relación entre el área de intersección y el área de unión.

$$IoU = \frac{A \bigcap A'}{A \bigcup A'} \tag{2.3}$$

Una vez se calcula el IoU entre cada una de las anchor boxes y cada una de las bounding boxes del ground truth se obtiene una matriz de tamaño *n*x*m*, siendo *n* el número de anchor boxes y *m* el número de bounding boxes del ground truth. A continuación, se realiza en primer lugar un emparejamiento de cada bounding box del ground truth con aquella anchor box que mayor IoU presente, de forma que cada bounding box del ground truth tenga una anchor box asociada. Además, es necesario emparejar el resto de anchor boxes para que aprendan a predecir algo. Esto se consigue emparejando cada anchor box con la bounding box del ground truth con mayor IoU. Además, se aplica un umbral, que en este caso es de 0.5, de forma que si el IoU es mayor, la anchor box está aprendiendo a predecir un objeto (emparejamiento positivo), mientras que si es menor está aprendiendo a predecir background (emparejamiento negativo). Esta estrategia es la utilizada en SSD [16] para conseguir simplificar el aprendizaje del modelo.

Si bien esta estrategia de emparejamiento es necesaria durante el entrenamiento de la red neuronal, durante la fase de inferencia en la que el modelo debe predecir bounding boxes por sí sola no se tiene en cuenta este emparejamiento sino que se utiliza un proceso denominado *Non-Maximum Suppression* (NMS). Este algoritmo parte de la lista de bounding boxes predichas por el modelo y la probabilidad de cada bounding box de pertenecer a cada una de las posibles clases. Este algoritmo iterativo elige en primer lugar la bounding box con mayor probabilidad de pertenecer a una clase y calcula su IoU con el resto de propuestas de esa clase. Esta bounding box con mayor probabilidad es elegida como una predicción correcta, de forma que todas las demás propuestas cuyo IoU respecto a esta bounding box sea superior a un umbral determinado (0.1 en este caso) son eliminadas, ya que se corresponden con bounding boxes que están prediciendo el mismo objeto que la elegida inicialmente pero con una menor probabilidad. Una vez realizado esto, se vuelve a elegir la siguiente bounding box con mayor probabilidad y se repite el proceso de cálculo de IoU y eliminación de propuestas hasta que se han elegido las mejores bounding

boxes y eliminado las demás. Este procedimiento es necesario porque tal y como está definido el modelo, las anchor boxes que se encuentren cercanas pueden haber aprendido a detectar el mismo tipo de objeto en una posición de la imagen similar. De esta forma, se eliminan todas aquellas bounding boxes que están detectando el mismo objeto quedándose la que mejor lo está detectando (mayor probabilidad).

En este caso, antes de aplicar el NMS a la salida de la red, se realiza un pre-procesado para eliminar parte de las bounding boxes propuestas, ya que la salida del modelo está formada por decenas de miles de propuestas. En primer lugar, para cada clase se eligen las 400 bounding boxes con mayor probabilidad que superen cierto umbral, cuyo valor será discutido a la hora de explicar los resultados obtenidos. A continuación, se aplica el NMS a estas bounding boxes dejando como máximo 50 de ellas (las que mayor probabilidad presenten). Finalmente, tras realizar esto con todas las clases se eligen las 200 bounding boxes con mayor probabilidad entre todas las clases, que serán las bounding boxes predichas finalmente para la imagen.

2.1.2. Rama de Segmentación

En cuanto a la segmentación semántica, la estrategia a seguir es bastante más simple que en el caso de la detección. Para esta tarea en primer lugar se aplica una convolución a cada uno de los mapas de características del flujo de upscaling, utilizando 64 filtros para conseguir una representación intermedia de cada uno de los mapas, de igual forma que en BlitzNet [18]. A continuación, se re-escala cada uno de ellos al tamaño de la última capa (128x256), mediante interpolación bilineal y se concatenan todos los mapas de características. Por último, se aplica una convolución con 15 filtros (14 clases y background) seguida de una función de activación tipo softmax obteniendo así un mapa de segmentación de 128x256 píxeles y 15 canales que representan, cada uno de ellos, la probabilidad de cada pixel de pertenecer a cada una de las clases.

2.2. Función de coste

Una de las propiedades más importantes de la arquitectura del modelo propuesta es que la mayoría de pesos se comparten entre las tareas de detección y segmentación, lo que facilita el aprendizaje de la red neuronal durante el entrenamiento. Para conseguir esto es muy importante definir cuidadosamente la función de coste a utilizar, ya que será la que permita la actualización de pesos y sesgos a lo largo de la red neuronal durante la propagación de errores. Para este modelo se ha utilizado una función de coste definida como la suma de la función de coste de detección y la función de coste de segmentación, pues al presentar valores en un mismo rango ambas contribuyen de igual forma a la función de coste total permitiendo a la red aprender ambas tareas.

2.2.1. Detección de objetos

La función de coste seleccionada para la tarea de detección es compleja y es la utilizada en BlitzNet [18]. El cálculo se realiza mediante la suma de la función de coste de clasificación y la función de coste de localización, las cuales se definirán más adelante. La función de coste de detección se calcula como sigue:

$$L_d = \frac{1}{N}(L_{class}(x, c) + L_{loc}(x, l, a, g) \tag{2.4}$$

donde L_{class} y L_{loc} son la función de coste de clasificación y de localización, respectivamente, y N el número de emparejamientos positivos durante el emparejamiento entre anchor boxes y ground truth. Si $N = 0$ el valor de la función de coste se fija en 0 de acuerdo a [18].

La función de coste de localización es del tipo *Smooth L1* entre los offsets predichos por el modelo (l) y los offsets relativos al ground truth respecto a la anchor box emparejada. Este cálculo entre los parámetros del ground truth (g) y los de las anchor boxes (a) da lugar a un offset relativo a la anchor box que se puede comparar directamente con el predicho por la red neuronal.

$$L_{loc}(x, l, a, g) = \sum_{i \in Pos}^{N} \sum_{m \in \{cx,cy,w,h\}} x_{ij}^{p} smooth_{L1}(l_i^m - \hat{g}_j^m)$$

$$\hat{g}_j^{cx} = \frac{(g_j^{cx} - a_i^{cx})}{a_i^w} \qquad \hat{g}_j^{cy} = \frac{(g_j^{cy} - a_i^{cy})}{a_i^h} \tag{2.5}$$

$$\hat{g}_j^w = \ln\left(\frac{g_j^w}{a_i^w}\right) \qquad \hat{g}_j^h = \ln\left(\frac{g_j^h}{a_i^h}\right)$$

Para calcular la función de coste de localización se utilizan varios parámetros: x_{ij}^p es un indicador binario (1 ó 0) para indicar si la anchor box *i* ha sido emparejada con la bounding box *j* del ground truth con categoría de objeto *p*; *l*, *a* y *g* son parámetros correspondientes a los offsets predichos por el modelo, las anchor boxes y las bounding boxes del ground truth, respectivamente. Estos parámetros son *cx*, *cy*, *w* y *h*, correspondientes a las coordenadas del centro de la caja, la anchura y la altura, respectivamente. Cabe destacar que debido al proceso de emparejamiento de bounding boxes (véase la Sección 2.1.1) puede ocurrir que varias anchor boxes estén emparejadas con la misma bounding box del ground truth, de forma que $\sum_i x_{ij}^p \geqslant 1$.

En cuanto a la función de coste de clasificación, esta se corresponde con una *cross entropy* aplicada a lo largo de todas las probabilidades (c) de pertenecer a cada clase para cada anchor box, ya sea emparejamiento positivo o negativo.

$$L_{class}(x, c) = -\sum_{i \in Pos}^{N} x_{ij}^{p} \ln\left(c_i^p\right) - \sum_{i \in Neg} \ln(c_i^0) \quad (2.6)$$

Hay que tener en cuenta que para los emparejamientos positivos se calcula la función de coste con las probabilidades de todas las clases posibles excepto el background, mientras que para los emparejamientos negativos sólo se tiene en cuenta la probabilidad de la clase background.

2.2.2. Segmentación semántica

En lo que se refiere a segmentación semántica, son varias funciones de coste las que se han utilizado a lo largo de los años en diferentes modelos. Sin embargo, la más común (y la que se ha utilizado en este proyecto) es la cross entropy entre la clase predicha y la clase real de cada píxel en la máscara de segmentación. Hay que tener en cuenta que la salida de la red neuronal presenta *M* mapas de segmentación, siendo *M* el número de clases a predecir. El cálculo de esta función de coste (L_s) se realiza de acuerdo a la siguiente ecuación:

$$L_s = -\sum_{c=0}^{M} x_{o,c} \ln(p_{o,c}) \quad (2.7)$$

donde $x_{o,c}$ es un parámetro de valor 1 ó 0 si la clase *c* de la observación *o* es correcta o no y $p_{o,c}$ la probabilidad de que la observación *o* sea de la clase *c*. Cabe destacar que cada observación se corresponde con cada uno de los píxeles del mapa de segmentación.

Capítulo 3

Dataset

La elección del dataset a utilizar para el entrenamiento del modelo, así como el tratamiento previo a realizar, son factores determinantes a la hora de obtener buenos resultados en las predicciones de la red neuronal. Es importante utilizar un dataset amplio con imágenes panorámicas correctamente etiquetadas, con bounding boxes para las labores de detección y máscaras de segmentación para la segmentación semántica de los objetos.

En este capítulo se detalla el dataset elegido así como las técnicas utilizadas para el data augmentation, que consiste en modificar aleatoriamente las imágenes para obtener unos datos de entrenamiento más diversos.

3.1. SUN360

SUN360 [22] (*Scene UNderstanding 360º panorama*) es un dataset de la Universidad de Princeton que contiene imágenes panorámicas de resolución 1024x512 píxeles con un campo visual de 360ºx180º. Este dataset contiene tanto imágenes de interiores como de exteriores aunque para este trabajo sólo se utilizarán las primeras. Por otro lado, en PanoContext [23] se realizó el etiquetado de bounding boxes para dos grupos de imágenes de interior, mientras que en Panoramic Blitznet [20] se completó este etiquetado con máscaras de segmentación semántica.

Para este proyecto se ha utilizado este dataset ya que es uno de los pocos existentes con imágenes panorámicas etiquetadas y también porque es el que se utilizó en Panoramic Blitznet, cuyo modelo se pretende programar desde cero. El dataset cuenta con un total de 666 imágenes divididas entre salas de estar (248 imágenes) y dormitorios (418 imágenes) con un total de 14 clases diferentes de objetos, detallados en la Tabla 3.1.

Categoría	Número de objetos
cuadro	1261
cama	540
mesa	1319
espejo	366
ventana	848
cortina	1059
silla	1377
lámpara	749
sofá	855
puerta	1473
armario	898
mesilla	550
tv	463
estantería	123
TOTAL	11881

Tabla 3.1: Número de objetos pertenecientes a cada una de las 14 clases posibles en el dataset SUN360.

Se puede apreciar como el número de objetos entre las distintas clases es muy diverso. Esto se conoce como desequilibrio de clases o *class imbalance*. Además, ocurre tanto en la tarea de detección como en la tarea de segmentación. Tras analizar las imágenes del dataset, calculando para cada clase el porcentaje de píxeles que aparecen, se puede apreciar en la Figura 3.1 como hay una gran diferencia entre los píxeles pertenecientes al background y los píxeles de las distintas clases a segmentar. Además, entre las diferentes clases también hay diferencias apreciables en la cantidad de píxeles.

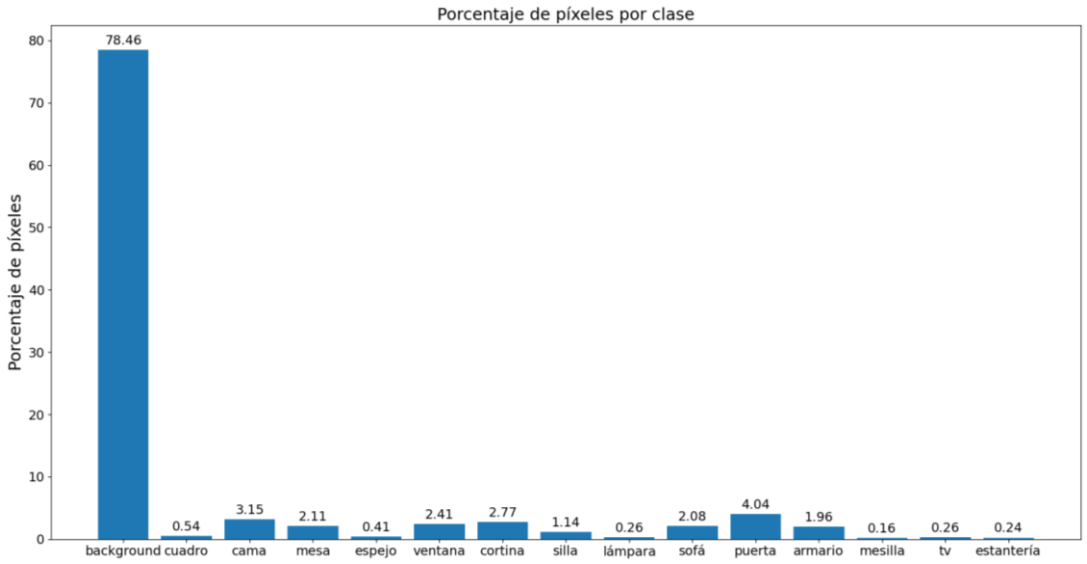


Figura 3.1: Distribución de píxeles por clase (en porcentaje) en el dataset SUN360.

Los efectos de este class imbalance serán analizados más en profundidad durante la evaluación del modelo, pues puede afectar negativamente al desempeño del modelo en las clases con menor representación.

3.2. Conversión del dataset a TFDS

Una vez elegido el dataset a utilizar para el entrenamiento y validación del modelo, son necesarios varios pasos de pre-procesamiento.

Como se ha especificado anteriormente, la red neuronal ha de implementarse en Python utilizando Tensorflow 2.x, por lo que en primer lugar es necesario convertir las imágenes y la información de las etiquetas a datos que la red neuronal pueda procesar. Para conseguir esto se ha utilizado el módulo TFDS (*TensorFlow DataSets*) de Tensorflow 2.x que permite convertir las imágenes de entrada y las imágenes de las máscaras de segmentación a matrices de forma sencilla, así como obtener los datos de bounding boxes del fichero tipo JSON (*Java Script Object Notation*). En este fichero se almacenan de forma estructurada diferentes datos referentes a cada uno de los objetos de las imágenes, como por ejemplo, la clase a la que pertenece el objeto, las coordenadas de su bounding box o el número total de píxeles que ocupa en la imagen.

Para realizar esta tarea hay que programar un pequeño fichero que al ejecutarse guarda toda la información que se quiera en una serie de archivos de tipo TFRecord, propio de Tensorflow 2.x. Estos archivos se pueden cargar en cualquier momento mediante una línea de código para obtener todo el dataset en una estructura de datos que se puede manejar y modificar fácilmente y que puede ser alimentada a la red neuronal.

Pese a que este paso es sencillo, es necesario decidir la forma en la que se interpretará cada objeto, ya que al utilizarse imágenes panorámicas algunos objetos pueden aparecer cortados en los límites de la imagen, como se muestra en la Figura 3.2. Pese a que en el archivo JSON si un objeto se encuentra cortado aparece como dos instancias distintas, cada una con una bounding box distinta, se decidió almacenar cada objeto con dos posibles bounding boxes, de forma que si aparecía entero en la imagen tendría su propia bounding box junto con un vector nulo para indicar que no existe una segunda bounding box. Esto se hizo así debido a que Tensorflow 2.x no permitía guardar un tensor de longitud variable por lo que para cada objeto se necesitaban dos vectores indicando sus bounding boxes. Por otro lado, si el objeto aparece cortado se cuenta como un solo objeto y se almacenan sus dos bounding boxes.

Esta es una de las principales diferencias respecto al modelo original en el que si un objeto aparecía cortado en la imagen se consideraba como dos objetos distintos. Esto es habitual en

imágenes convencionales pero genera problemas al tratar con imágenes panorámicas de 360º, por lo que se propuso esta solución para evitarlos.

Figura 3.2: Ejemplo de una imagen del dataset donde una silla aparece cortada en los bordes de la imagen.

3.3. Data augmentation

Uno de los factores a tener en cuenta durante el entrenamiento de una red neuronal es el sobreajuste u *overfitting*: el modelo se ajusta excesivamente bien a los datos de entrenamiento de forma que la red no generaliza, impidiendo obtener resultados precisos con datos nuevos.

El objetivo principal de una red neuronal es obtener patrones a partir de datos de entrenamiento para poder predecir correctamente datos nuevos, por lo que se debe evitar el sobreajuste. Aunque existen distintos métodos para solucionar este problema, este apartado se centrará en el data augmentation, el cual consiste en aplicar transformaciones aleatorias a los datos de entrenamiento para conseguir así aumentar la diversidad del conjunto de entrenamiento.

En el caso de SUN360, se trata de un dataset con una cantidad de imágenes pequeña en comparación con otros datasets, por lo que es muy necesario realizar un buen data augmentation para evitar el gran sobreajuste que habría inicialmente. Para ello se han aplicado distintas funciones a los datos de entrenamiento, las cuáles se describirán en los apartados posteriores y se demostrará su funcionamiento sobre la Figura 3.3.

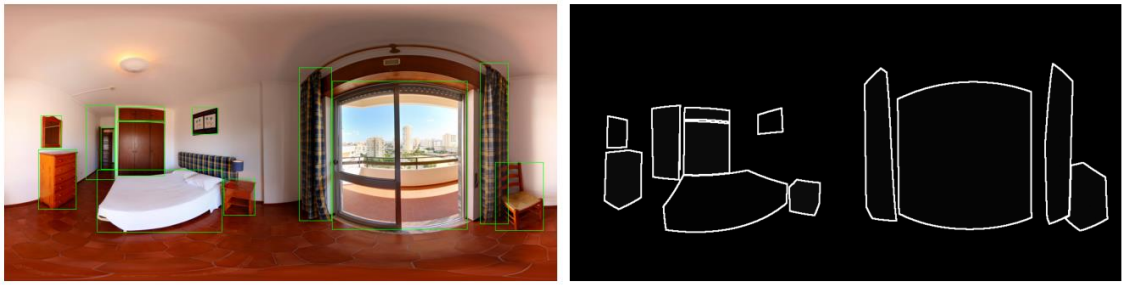


Figura 3.3: Figura inicial sobre la que se demostrará el funcionamiento de las distintas funciones de data augmentation.

3.3.1. Volteo horizontal

Una de las transformaciones más habituales es el volteo de las imágenes, de forma que se obtiene una imagen reflejada horizontalmente como si de un espejo se tratase. Esta función es muy sencilla de realizar y se aplica tanto a las imágenes de entrada como a las máscaras de segmentación y las coordenadas de las bounding boxes de cada objeto. En la Figura 3.4 se puede observar un ejemplo de aplicación de esta transformación.

Figura 3.4: Aplicación de volteo horizontal sobre la Figura 3.3.

3.3.2. Rotación horizontal

Por otro lado, el dataset utilizado presenta imágenes con unas condiciones de posición de cámara muy similares. Por este motivo, se ha realizado una función que rota horizontalmente la imagen a lo largo de los 360º de la esfera, de forma que se cubren las diferentes posiciones de la cámara en la misma, como se ha representado en la Figura 3.5.

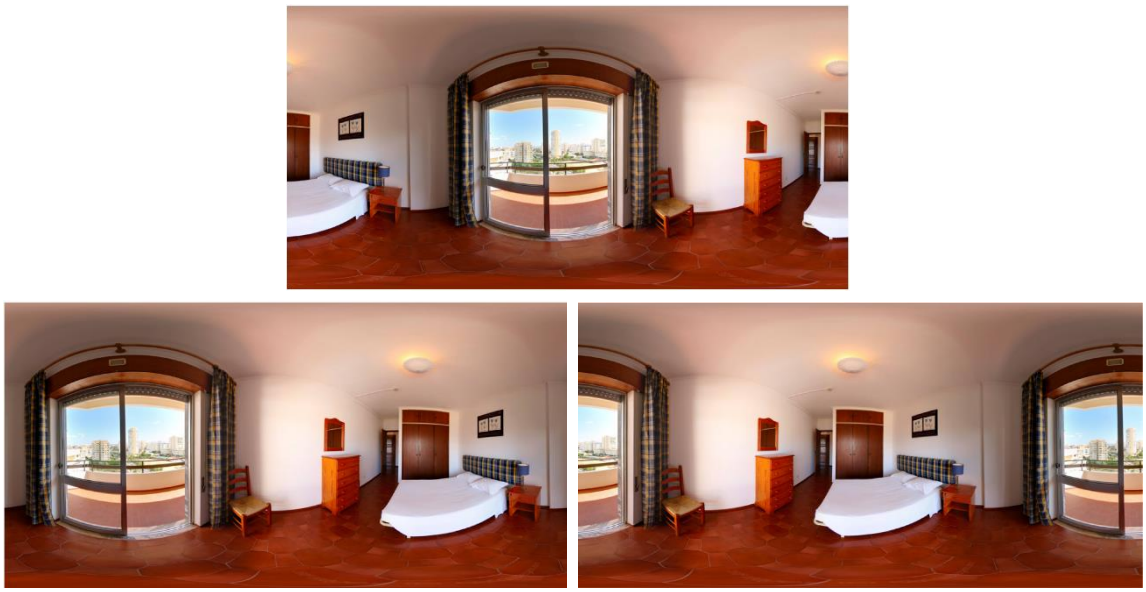


Figura 3.5: Ejemplos de rotaciones sobre la Figura 3.3: 90º (arriba), 180º (izquierda), 270º (derecha).

Para realizar esta función hay que tener en cuenta que al rotar aleatoriamente la imagen, objetos que aparecían cortados originalmente pueden aparecer enteros o viceversa, así como

mantenerse cortados o enteros. Debido a la decisión inicial de almacenar para cada objeto sus dos posibles bounding boxes hay que tener cuidado a la hora de realizar los cálculos de la rotación ya que hay diversas posibilidades, las cuales se han tenido que tener en cuenta a la hora de implementar esta función. En la Figura 3.6 se muestra un ejemplo completo de la aplicación de esta función.

Figura 3.6: Ejemplo de rotación sobre la Figura 3.3 tanto a la imagen original como a las bounding boxes y el mapa de segmentación.

3.3.3. Alteración cromática

Como ya se ha comentado anteriormente, los objetos pueden aparecer en condiciones de iluminación muy variadas. Por este motivo es muy importante que durante el entrenamiento se utilicen imágenes con condiciones de iluminación muy diversas. Sin embargo, debido a la reducida cantidad de imágenes que presenta el dataset no es posible conseguir estas condiciones, por lo que se han planteado transformaciones aleatorias que afectan a algunas propiedades de las imágenes de entrada. En este caso, se ha modificado aleatoriamente el matiz (*hue*), la saturación, el brillo y el contraste de las imágenes para obtener la mayor variedad en cuanto a condiciones de luz se refiere. En la Figura 3.7 se ha representado un ejemplo de aplicación de esta función.

Figura 3.7: Ejemplo de alteración cromática sobre la Figura 3.3.

Capítulo 4

Métricas de evaluación

Otro concepto clave a la hora de trabajar con redes neuronales es el de la métrica de evaluación, a través de la cual se mide el desempeño del modelo al predecir resultados. En este capítulo se explican algunas de las métricas utilizadas habitualmente en este tipo de redes neuronales, así como las métricas elegidas para este trabajo.

4.1. Detección de objetos

En el caso de la detección de objetos, en este trabajo se ha utilizado el *Mean Average Precision* (mAP). Aunque antes de explicar en qué consiste esta métrica es necesario explicar otras métricas utilizadas en ocasiones para evaluar la detección de objetos y que sirven como base para calcular el mAP.

En primer lugar, hay que nombrar el IoU, el cual se utiliza en algunos trabajos como métrica de evaluación. El primer paso es elegir un umbral concreto para filtrar las predicciones. Este umbral se selecciona en función de lo restrictivo que se quiera ser con la detección. Normalmente se utilizan valores de 0.5 o 0.75 en la mayoría de trabajos aunque se pueden ver valores entre 0.3 y 0.95, no obstante, en este trabajo se utilizará un valor de 0.5. Una vez se calcula la matriz de IoU entre el ground truth y las predicciones, con el umbral a utilizar elegido, se pueden definir los siguientes conceptos:

- Verdadero positivo (TP): Predicción correcta (IoU > umbral).
- Falso positivo (FP): Predicción incorrecta (IoU < umbral).
- Falso negativo (FN): Objeto del ground truth no detectado (IoU = 0 para toda bounding box del ground truth).

A partir de estos parámetros se definen dos conceptos muy importantes en la detección de objetos: precisión o *precision* y exhaustividad o *recall* (a partir de ahora se referirá a ambos términos por su nomenclatura inglesa), definidos a continuación:

$$Precision = \frac{TP}{TP + FP}$$
$$Recall = \frac{TP}{TP + FN} \quad (4.1)$$

La precision representa la capacidad del modelo de detectar correctamente los objetos, es decir, la relación entre las predicciones correctas y el total de predicciones. El recall, por otro lado, representa la exhaustividad con la que el modelo es capaz de detectar los objetos, lo que complementa la precision. Si bien ambos parámetros son importantes en este tipo de aplicaciones, el aumento de una de las métricas da lugar a una disminución de la otra, por lo que se debe buscar un equilibrio entre ambos o bien darle más importancia a una de las dos en función de lo que requiera la tarea a realizar.

Estos parámetros son esenciales en otra de las métricas más utilizadas en detección de objetos: *Average Precision* (AP), que representa la media de los valores de precision calculados a todos los niveles de recall posibles. Este valor se obtiene a partir de la curva precision-recall, la cual representa la evolución de la precision a lo largo de los distintos niveles de recall, siendo el AP el valor del área bajo la curva. Sin embargo, tal y como se muestra en la Figura 4.1, los valores de precision siguen una evolución en zigzag haciendo costoso el cálculo del área bajo la curva. Por ello, es habitual utilizar una precision interpolada (p_{interp}) que se define matemáticamente como sigue:

$$p_{interp}(r) = \max_{r' \geqslant r} p(r') \quad (4.2)$$

donde *r* es el nivel de recall que se está evaluando, *r'* todos los niveles de recall mayores que el nivel actual y *p* el valor de precision para un nivel de recall. La precision interpolada para un nivel de recall es, por tanto, el máximo valor de precision para cualquier nivel de recall mayor que el que se evalúa.

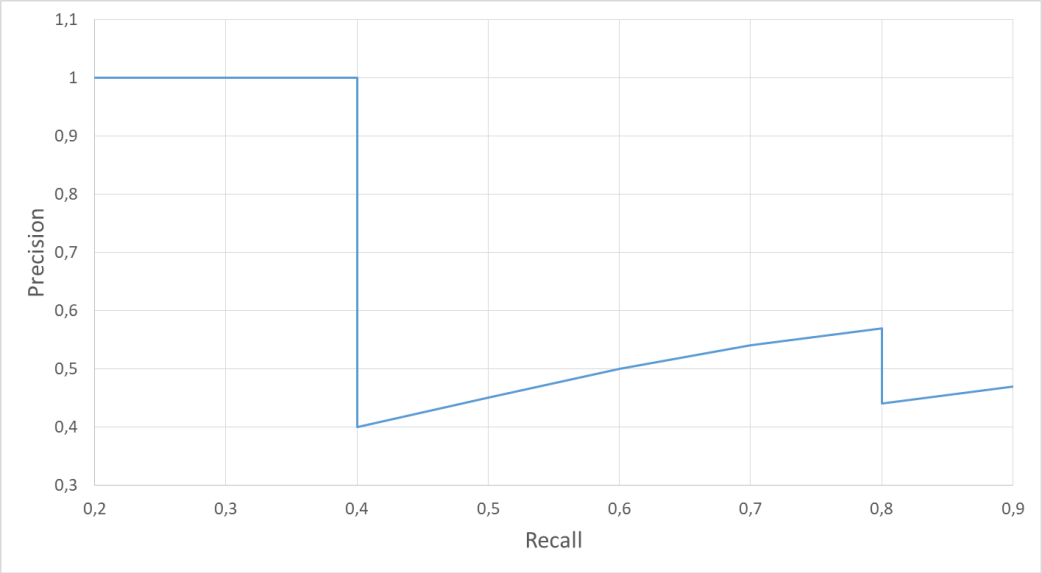


Figura 4.1: Ejemplo de una curva precision-recall.

Por último, el valor de AP se puede calcular como la media de todos los valores de la precision interpolada a lo largo de los *N* niveles de recall, como se define a continuación:

$$AP = \frac{1}{N}\sum_{i=1}^{N} p_{interp}(r_i) \tag{4.3}$$

Sin embargo, debido a la presencia de varias clases de objetos surge la necesidad de utilizar como métrica de evaluación el mAP que refleja la media de AP a lo largo de las distintas clases, de la forma que se define a continuación:

$$mAP = \frac{1}{M}\sum_{i=1}^{M} AP_i \tag{4.4}$$

donde *M* es el número total de clases y AP_i el *average precisión* para la clase *i*.

4.2. Segmentación semántica

Para las tareas de segmentación semántica la métrica de evaluación más utilizada es el IoU ya que permite relacionar directamente el área de píxeles predichos con el área de píxeles del ground truth. Sin embargo, es habitual que en las aplicaciones en las que se requiere segmentación semántica existan diversas clases de objetos a segmentar. Por este motivo, se utiliza el *Mean Intersection Over Union* (mIoU), que se corresponde con la media de IoU a lo largo de las distintas clases de objetos que se quieren segmentar, expresándose matemáticamente de la siguiente forma:

$$mIoU = \frac{1}{M}\sum_{i=0}^{M} \frac{A_i \cap A'_i}{A_i \cup A'_i} \tag{4.5}$$

siendo M el número de clases (el background se considera como la clase 0), A_i el área formada por todos los píxeles pertenecientes a la clase i en el mapa de segmentación del ground truth, es decir, el número total de píxeles de una clase y A'_i el área formada por todos los píxeles pertenecientes a la clase i en el mapa de segmentación predicho.

Para realizar el cálculo de esta métrica se parte de la matriz de confusión. Como se observa en la Figura 4.2, está formada por M filas y M columnas, de forma que cada fila representa una clase del ground truth y cada columna representa una clase predicha. En esta tabla se refleja el número de predicciones correctas para cada clase, que se corresponde con los valores de la diagonal de la matriz, así como el número de predicciones incorrectas, que se corresponden con los valores fuera de la diagonal. El IoU para cada clase se calcula de la siguiente forma:

$$IoU = \frac{TP}{FP + FN + TP} \tag{4.6}$$

Los verdaderos positivos (TP) se corresponden con el valor de la diagonal para cada clase, los falsos positivos (FP) son la suma de valores a lo largo de la fila para cada clase sin tener en cuenta el valor de la diagonal y los falsos negativos (FN) son la suma de valores a lo largo de la columna para cada clase sin tener en cuenta el valor de la diagonal. De esta forma se puede calcular el mIoU como la media de todos los valores de IoU calculados para todas las clases.

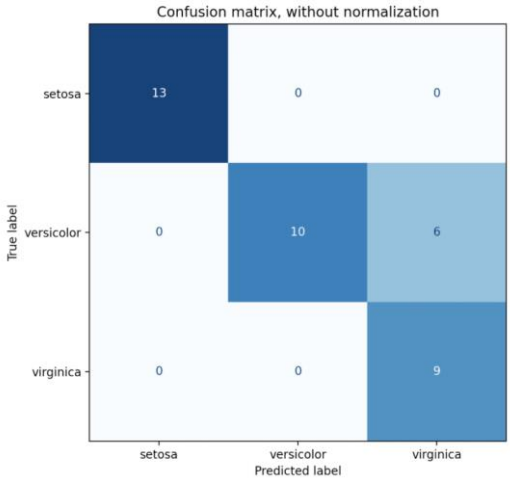


Figura 4.2: Ejemplo de una matriz de confusión con diversos tipos de flores. Imagen obtenida de [24].

Capítulo 5

Entrenamiento

El paso más importante para obtener unos resultados de calidad a la hora de utilizar el modelo para inferencia a partir de imágenes nuevas es realizar un correcto entrenamiento. Este viene definido por ciertos parámetros denominados hiperparámetros, los cuales se deben ajustar manualmente hasta optimizar los resultados obtenidos. En este caso, a priori no parecía necesario ajustar estos parámetros, ya que se utiliza la configuración de entrenamiento de Panoramic BlitzNet. Sin embargo, el hecho de implementar todo el modelo con código nuevo y en una versión de Tensorflow muy distinta de la usada en la red original de BlitzNet ha dado lugar a muchos problemas a la hora de obtener un entrenamiento adecuado.

En este capítulo se abordan las dificultades encontradas durante el proceso de entrenamiento, así como las pruebas realizadas y la forma en la que se ha realizado el entrenamiento final de la red neuronal.

5.1. Hiperparámetros y bases del entrenamiento de una CNN

Antes de explicar los experimentos realizados es necesario tener en cuenta ciertos conceptos relativos al entrenamiento de las redes neuronales convolucionales. En primer lugar se va a hablar de los hiperparámetros, los cuales son variables que determinan parte de la estructura del modelo y cómo se va a realizar el entrenamiento de la red neuronal. Si bien existen multitud de parámetros sólo se comentarán los más relevantes para el trabajo realizado.

Como ya se ha explicado (véase la Sección 1.2.1), los kernels utilizados a través de las capas convolucionales que conforman el modelo presentan unos pesos determinados que se van actualizando. Respecto a estos pesos aparecen dos hiperparámetros: el inicializador y el regularizador del kernel. El primero define como se van a elegir los primeros pesos del kernel al comienzo del entrenamiento, mientras que el segundo determina si se va a penalizar, así como la

forma de penalizar una capa del modelo. Esta penalización se suma a la función de coste durante la optimización del modelo durante el entrenamiento. La regularización se utiliza para reducir la posibilidad de sobreajuste al tener una red muy compleja con pocos datos de entrada, como es el caso de este trabajo.

Por otro lado, el proceso de entrenamiento consiste en pasar todas las imágenes del conjunto de entrenamiento a través del modelo para optimizar los pesos a través de la función de coste. El proceso de pasar todas las imágenes se denomina época, siendo el número de épocas el hiperparámetro que determina cuantas veces se van a pasar todos los datos de entrada al modelo y que se debe elegir de forma que el entrenamiento llegue a converger. Durante una época los datos de entrada no se pasan a la vez por el modelo sino que se elige un tamaño de lote o *batch size* que se corresponde con el número de datos de entrada que se pasan cada vez. Por ejemplo, si se elige un batch size de valor 2, cada vez se pasan dos imágenes del conjunto de entrenamiento por la red, se calcula la función de coste y se actualizan los pesos y sesgos del modelo. A continuación, se pasan otras dos imágenes y se repite el proceso hasta que se completa una época. Este hiperparámetro es importante ya que representa cuantas veces se van a actualizar los pesos de la red en cada época, afectando a la velocidad de convergencia del modelo, así como a la memoria de la GPU requerida para almacenar toda la información relativa a los datos de entrada que se introduzcan en cada batch, por lo que suele ser un factor limitante a la hora de elegir el batch size.

Por último, hay un hiperparámetro relativo a la optimización de la función de coste: la tasa de aprendizaje o *learning rate*. Esta tasa se corresponde con el paso con el que se actualiza cada uno de las variables del modelo durante la optimización de la función de coste. A mayor learning rate más rápido converge el entrenamiento hasta estancarse por ser el paso demasiado grande para seguir disminuyendo el valor de la función de coste, mientras que con un menor learning rate la convergencia es más lenta pero se llega a alcanzar el mínimo de la función de coste.

5.2. Proceso de depuración del modelo

Toda implementación de código lleva consigo un proceso de depuración o *debugging* para comprobar el correcto funcionamiento del mismo. Con este modelo implementado desde cero ocurre lo mismo. Sin embargo, si bien la gran mayoría de funciones del código son fácilmente depurables, todo lo referente al modelo y lo que ocurre desde que entra una imagen hasta que se obtiene la salida es complicado de comprobar debido a la alta complejidad de la red neuronal.

Pese a poder consultar el código original de BlitzNet, este es bastante confuso y presenta multitud de código destinado a otros datasets y tareas considerados. A esto hay que añadir que

esta red neuronal fue implementada para el uso con imágenes convencionales. Además, hay una gran diferencia entre la forma de implementar el modelo y el entrenamiento entre las versiones de Tensorflow 1.x y Tensorflow 2.x, por lo que leer y entender el código original ha resultado ciertamente complicado.

Con el modelo, las funciones de coste y las métricas de evaluación implementadas, se realizó un primer entrenamiento. Pese a que todo se ejecutaba correctamente, los resultados obtenidos durante el entrenamiento eran claramente erróneos, siendo el más claro que el mapa de segmentación obtenido era una imagen negra, es decir, el modelo sólo estaba prediciendo background.

Siendo que la red neuronal no estaba funcionando correctamente se realizaron sucesivas revisiones de todo el código, con especial atención al modelo y las funciones de coste y métricas, comparando a la vez con el código original de BlitzNet. Con esto se encontraron pequeños fallos de código, entre ellos: la falta de función de activación ReLU al final de alguna capa convolucional, un mal tratamiento de los datos del ground truth que hacían que el mapa de segmentación del ground truth fuera una imagen negra, el uso inadecuado de padding en algunas convoluciones o alguna ecuación con pequeñas erratas. Sin embargo, tras solucionar algunos errores del código, el entrenamiento seguía sin funcionar correctamente. Por este motivo se volvió a revisar en detalle el código del modelo comparando con el de BlitzNet, encontrando así los problemas que impedían a la red neuronal entrenar correctamente: el inicializador y el regularizador del kernel y la forma de entrada de las imágenes al modelo.

Por defecto, las convoluciones en Tensorflow 1.x y Tensorflow 2.x no utilizan regularizador, mientras que como inicializador utilizan el inicializador de Xavier uniforme [25]. Si bien en el cuerpo principal de la arquitectura utilizada en BlitzNet se utilizan estos parámetros, en las ramas de segmentación y detección de objetos se utilizan otros inicializadores y regularizadores, dando lugar a que el modelo implementado no entrenara correctamente. En el caso de la rama de detección se utiliza como inicializador uno denominado *variance scaling*, mientras que en la de segmentación se utiliza Xavier uniforme. Por otro lado, en ambas ramas se utiliza el regularizador L2. Esta regularización añade a la función de coste un término consistente en un parámetro λ de valor pequeño (0.0005 en este caso) multiplicando a la suma del cuadrado de todos los pesos de la capa.

En cuanto a la forma de introducir los datos al modelo, lo habitual en las redes neuronales es introducir las imágenes con valores normalizados en el rango [0, 1] para acelerar el proceso de entrenamiento y convergencia al trabajar con valores numéricos más pequeños. Esto se debe a que durante el paso de una imagen por la red neuronal las salidas de las capas se calculan mediante el producto entre los pesos y las entradas, por lo que valores más altos requieren mayor tiempo

de cómputo y mayor gasto de memoria. Sin embargo, el trabajar con valores pequeños puede ocasionar el problema de desvanecimiento de gradiente durante la propagación de errores para actualizar los pesos. Este problema consiste en que durante la propagación de errores se vayan acumulando valores de derivadas parciales cada vez más pequeños debido a la regla de la cadena. Esto impide que los pesos se actualicen correctamente y, por tanto, hace que la red neuronal sea incapaz de aprender. Además, en el caso de la segmentación, el background es la clase predominante por lo que tiene sentido que la salida de segmentación fuese una imagen negra, ya que al no actualizarse los pesos adecuadamente el modelo se estancaba prediciendo la clase predominante.

Una vez solucionados estos errores se consiguió un funcionamiento adecuado de la red neuronal en cuanto a la rama de segmentación se refiere. Sin embargo, seguían existiendo problemas en la rama de detección de objetos, en especial utilizando un *batch size* mayor que 1. El problema encontrado se debía a la falta de coherencia en las dimensiones de las variables intermedias del modelo, dando lugar a que durante el emparejamiento de bounding boxes se mezclaran las bounding boxes del ground truth de unas imágenes del batch con otras, consiguiendo así una oscilación de la función de coste, una falta de convergencia y unos resultados pobres.

5.3. Entrenamiento del modelo

Como ya se ha explicado en este capítulo, la elección de hiperparámetros es un factor determinante a la hora de conseguir un buen entrenamiento, sin embargo, la gran mayoría de ellos no han hecho falta ajustarlos a mano ya que se conocen del trabajo de BlitzNet.

La estrategia seguida para el entrenamiento ha sido similar a la utilizada en BlitzNet, utilizando un learning rate inicial de 10^{-4}, el cual se disminuye dos veces a lo largo del entrenamiento, primero a 10^{-5} y finalmente a 10^{-6}. Como optimizador de la función de coste se ha utilizado el algoritmo de optimización estocástico Adam [26]. Además, se ha inicializado el bloque ResNet50 del modelo con los pesos de ImageNet, pues de acuerdo a las pruebas realizadas en [20] se obtiene una mayor velocidad de convergencia y mejores resultados que entrenando la red neuronal desde cero.

Por otro lado, debido a que el entrenamiento se ha hecho con un dataset distinto al utilizado en BlitzNet y con una estrategia distinta a la usada en Panoramic BlitzNet (en ese caso se realiza un pre-entrenamiento con otro dataset antes de entrenar con SUN360), se han hecho pruebas de entrenamiento utilizando un conjunto de validación. Este conjunto de validación se ha elegido como un 15% de los datos del conjunto de entrenamiento y no se utiliza para entrenar la red

neuronal, sino que se utiliza al final de cada época para comprobar el funcionamiento del modelo con datos nuevos y así poder comprobar el sobreajuste existente a lo largo del entrenamiento. Además, este entrenamiento de prueba se ha realizado para conocer las épocas en las que es necesario disminuir el learning rate debido a un estancamiento del valor de la función de coste. Una vez se han ajustado estos hiperparámetros se ha realizado un entrenamiento final con el 100% de los datos del conjunto de entrenamiento, sin utilizar validación. En cuanto a la GPU utilizada, ha sido una RTX 2080Ti y se ha utilizado un batch size igual a 2 por limitaciones con la memoria de la tarjeta gráfica. Como se puede observar en la Tabla 5.1, la convergencia del entrenamiento es rápida, tan sólo 200 épocas, que para la GPU utilizada son aproximadamente 8 horas de entrenamiento.

mAP	*mIoU*	Convergencia
0.923	0.811	200 épocas

Tabla 5.1: Resultados de entrenamiento obtenidos con el conjunto de entrenamiento de SUN360

Capítulo 6

Evaluación

En este capítulo se abordan los experimentos realizados para comprobar el comportamiento del modelo en distintos casos. Además, se discute sobre los resultados obtenidos en los distintos experimentos realizados.

6.1. Experimentos

En este caso se han realizado tres experimentos diferentes: El primero de ellos muestra la variación del desempeño del modelo al modificar el umbral de nivel de confianza con el que se filtran las predicciones antes del NMS. El segundo de ellos estudia los resultados obtenidos modificando las funciones de coste para lidiar con el class imbalance. Finalmente, el tercer experimento consiste en una comparación entre el modelo implementado y el modelo de Panoramic BlitzNet.

6.1.1. Umbral de nivel de confianza

Como se ha explicado en la Sección 2.1.1, durante el filtrado de predicciones obtenidas con el modelo trabajando en inferencia, se realiza un primer filtrado de las predicciones antes de aplicar el NMS para obtener las predicciones finales. Este filtrado se realiza mediante un umbral de nivel de confianza, de forma que para cada una de las predicciones de cada clase si la probabilidad predicha es mayor que el umbral esa predicción se considera válida. Sin embargo, este umbral puede modificar los valores de precision y recall obtenidos, y por tanto, los valores de AP y mAP a la hora de evaluar el modelo. Elegir el valor de este umbral conlleva un compromiso entre precision y recall, ya que con valores altos las detecciones serán en su mayoría correctas pero muchos objetos no serán detectados, aumentando el número de falsos negativos. Por otro lado, un valor bajo de este umbral hará que muchos objetos sean detectados pero aumentará el número de

falsos positivos. En la Figura 6.1 se ha representado la evolución de la precision y el recall al variar el umbral de nivel de confianza utilizado.

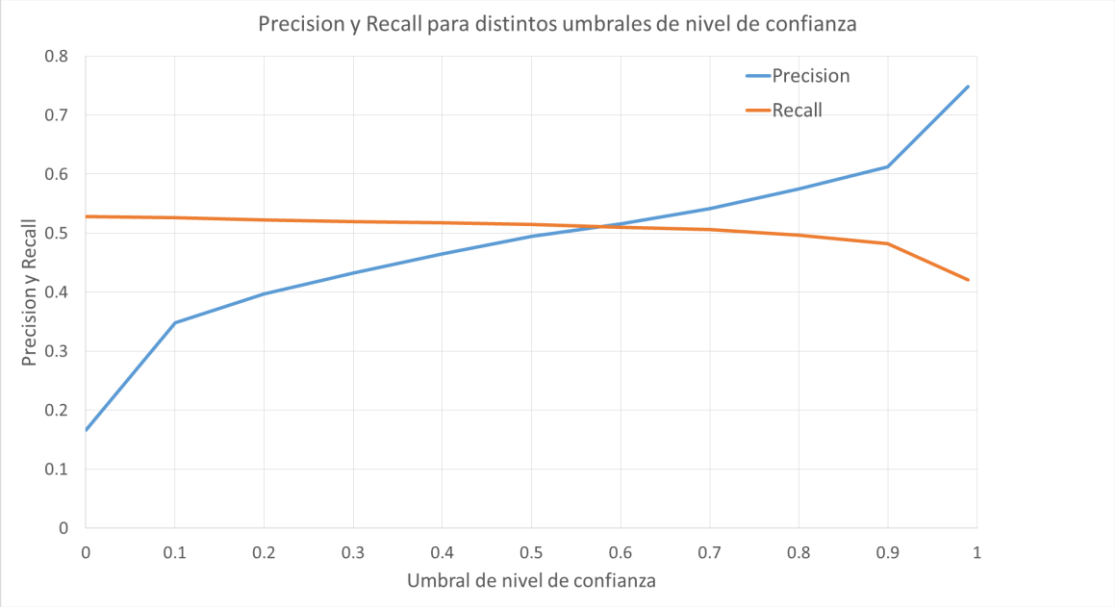


Figura 6.1: Evolución de precision y recall al variar el umbral de nivel de confianza para aceptar una predicción como válida.

Se puede observar que el recall apenas varía, mientras que la precision aumenta conforme aumenta el umbral utilizado. Sin embargo, puesto que el valor de recall no decae mucho hasta umbrales muy altos se ha decidido utilizar un umbral de 0.8.

6.1.2. Funciones de coste ponderadas

Al explicar el dataset utilizado para entrenar y evaluar la red neuronal implementada (véase la Sección 3.1) se ha mostrado como existe un desequilibrio de clases tanto en detección como en segmentación semántica, siendo más pronunciado este desequilibrio en la tarea de segmentación. Este desequilibrio puede ocasionar que algunas clases con menos representación en el dataset presenten peores resultados a la hora de realizar la inferencia con el modelo, por lo que se han realizado pruebas mediante la adición de pesos en las funciones de coste de detección y de segmentación.

A la hora de implementar los pesos, se han modificado las funciones de coste de clasificación (L_{class}), en el caso de la detección, y la de segmentación (L_s). La función de coste de clasificación queda de la siguiente forma:

$$L_{class}(x,c) = -\sum_{i \in Pos}^{N} w_{c,d} x_{ij}^{p} \ln\left(c_i^p\right) - \sum_{i \in Neg} \ln(c_i^0) \qquad (6.1)$$

Esta ecuación es similar a la Eq. 2.6 pero añadiendo el término $w_{c,d}$, el cual es el peso asociado a cada clase *c* para la tarea de detección. El peso para cada clase se calcula como el cociente entre el número total de objetos en el dataset y el producto del número de clases por el número de objetos de cada clase, como se describe a continuación:

$$w_{c,d} = \frac{n_o}{M n_{o,c}} \tag{6.2}$$

donde n_o es el número total de objetos, *M* el número de clases y $n_{o,c}$ el número de objetos para la clase *c*. Hay que destacar que el background tiene un peso de valor 1, ya que no se realizan detecciones positivas del mismo. Sin embargo, a la hora de calcular los pesos se ha tenido en cuenta que $M = 15$, ya que para el background también se calcula una probabilidad para cada predicción.

En el caso de la segmentación semántica, la función de coste ponderada queda como sigue:

$$L_s = -\sum_{c=0}^{M} w_{c,s} x_{o,c} \ln(p_{o,c}) \tag{6.3}$$

La función es idéntica a la Eq. 2.7 pero añadiendo el término $w_{c,s}$, que es el peso asociado a cada clase *c* para la tarea de segmentación. Los pesos se calculan de forma similar que en el caso de detección pero utilizando el número de píxeles totales y el número de píxeles de cada clase, de la siguiente forma:

$$w_{c,s} = \frac{n_p}{M n_{p,c}} \tag{6.4}$$

donde n_p es el número total de píxeles, *M* el número de clases y $n_{p,c}$ el número de píxeles para la clase *c*.

En la Tabla 6.1 se han representado los pesos para cada clase en las tareas de detección de objetos y segmentación semántica, respectivamente.

	Detección	Segmentación
background	1.0	0.085
cuadro	0.647	12.336
cama	1.435	2.114
mesa	0.587	3.160
espejo	2.248	16.111
ventana	0.931	2.768
cortina	0.754	2.411
silla	0.578	5.870
lámpara	1.077	25.316
sofá	0.915	3.201
puerta	0.537	1.651
armario	0.841	3.396
mesilla	1.508	41.038
tv	1.718	25.277
estantería	6.216	27.338

Tabla 6.1: Pesos asociados a cada una de las clases para las tareas de detección y segmentación semántica.

Con los pesos calculados para ambas tareas se han realizado experimentos con varias combinaciones: sin utilizar pesos, utilizando pesos sólo en la tarea de segmentación semántica, utilizando pesos sólo en la tarea de detección y usando pesos en ambas tareas. Los resultados obtenidos se han representado en la Tabla 6.2, mientras que en el Anexo A se puede encontrar información más detallada de los resultados obtenidos en este experimento divididos por clase.

Segmentación	Detección	*mAP*	*mIoU*
NO	NO	0.842	0.481
NO	SÍ	0.837	0.482
SÍ	NO	0.838	0.445
SÍ	SÍ	0.849	0.447

Tabla 6.2: Efecto de utilizar pesos durante el entrenamiento: En todos los casos la inicialización y el entrenamiento han sido idénticos, pero se han utilizado pesos en segmentación, detección, ambas o ninguna. Los resultados se han evaluado con un umbral de nivel de confianza de 0.8.

6.1.3. Comparación con Panoramic BlitzNet

Aunque no es un objetivo del trabajo obtener mejores resultados que los de la red neuronal original, es necesario obtener unos resultados suficientemente buenos como para tener una base sobre la que se pueda seguir trabajando. Por este motivo, en las Tablas 6.3 y 6.4 se han expuesto comparativas entre los resultados cuantitativos obtenidos en Panoramic BlitzNet y los obtenidos con la red neuronal implementada en este trabajo. Cabe destacar que esta comparativa no es del todo justa ya que en Panoramic BlitzNet la estrategia de entrenamiento es diferente a la utilizada en este trabajo. Aun así, estos resultados sirven para tener una idea del funcionamiento de la red

neuronal implementada. Por otro lado, en la Figura 6.2 se han representado resultados cualitativos comparando ambas implementaciones. Además, en el Anexo A se pueden encontrar más resultados cualitativos de la implementación realizada.

DETECCIÓN (AP)	Implementación propia	PanoBlitzNet
cuadro	**0.899**	0.839
cama	**0.965**	0.953
mesa	**0.905**	0.821
espejo	**0.891**	0.762
ventana	**0.808**	0.709
cortina	**0.827**	0.759
silla	**0.888**	0.809
lámpara	**0.732**	0.410
sofá	**0.945**	0.854
puerta	**0.813**	0.725
armario	**0.631**	0.556
mesilla	**0.918**	0.914
tv	**0.963**	0.933
estantería	**0.603**	0.402
mAP	**0.842**	0.746

Tabla 6.3: Comparativa de resultados en la tarea de detección (mAP) entre la implementación propia y Panoramic BlitzNet. La implementación propia ha sido entrenada sin utilizar pesos en las funciones de coste y evaluada con un umbral de nivel de confianza de 0.8.

SEGMENTACIÓN (IoU)	Implementación propia	PanoBlitzNet
background	0.906	**0.913**
cuadro	**0.632**	0.612
cama	**0.680**	0.621
mesa	0.404	**0.723**
espejo	0.378	**0.414**
ventana	0.515	**0.534**
cortina	**0.578**	0.537
silla	0.302	**0.552**
lámpara	0.259	**0.265**
sofá	**0.579**	0.329
puerta	0.461	**0.638**
armario	0.306	**0.511**
mesilla	**0.577**	0.366
tv	**0.569**	0.523
estantería	0.072	**0.619**
mIoU	0.481	**0.544**

Tabla 6.4: Comparativa de resultados en la tarea de segmentación semántica (mIoU) entre la implementación propia y Panoramic BlitzNet. La implementación propia ha sido entrenada sin utilizar pesos en las funciones de coste y evaluada con un umbral de nivel de confianza de 0.8.

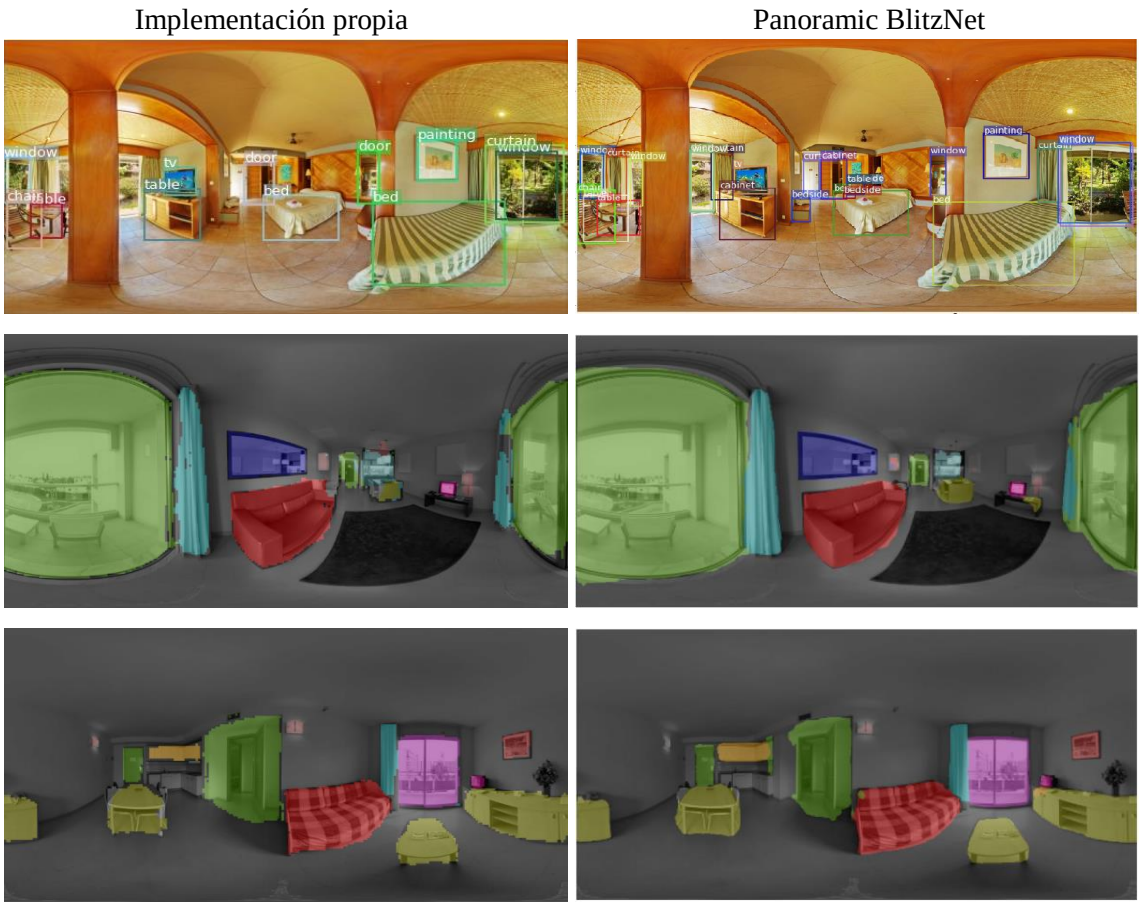


Figura 6.2: Comparativa de resultados cualitativos entre la implementación propia y Panoramic BlitzNet (imágenes obtenidas de [20]*). La implementación propia ha sido entrenada sin utilizar pesos en las funciones de coste y evaluada con un umbral de nivel de confianza de 0.8.*

6.2. Discusión de resultados

En los resultados expuestos se puede ver claramente como la tarea de segmentación semántica presenta peores resultados que la de detección. Esto puede deberse a que para la segmentación semántica la red neuronal debe aprender patrones a nivel de píxel, mientras que la detección no se realiza a nivel de píxel, de forma que es más difícil para el modelo aprender a segmentar. Por otro lado, durante el análisis del dataset se han detectado imágenes mal etiquetadas en cuanto a segmentación se refiere, lo cual puede haber afectado al desempeño del modelo en la tarea de segmentación. Además, el hecho de que el dataset utilizado presente una cantidad pequeña de imágenes puede ser un limitante para el modelo a la hora de aprender patrones tanto para segmentación semántica como para detección.

De los experimentos realizados para lidiar con el class imbalance, se ha demostrado que el hecho de ponderar las funciones de coste no mejora los resultados para el dataset utilizado. En el caso de la segmentación semántica, la utilización de pesos hace que los resultados de mIoU empeoren. Probablemente es debido al mal etiquetado de algunas máscaras de segmentación.

Aumentar el peso asociado a una clase mal etiquetada produce que se incentive una mala segmentación, empeorando así los resultados. En cuanto a la detección, al añadir pesos el mAP apenas varía. No obstante, la distribución del AP a lo largo de las distintas clases sí que varía. Este comportamiento es debido al alto desequilibrio entre las clases de objetos. Sin la utilización de pesos, la red favorece la detección de los objetos con mayor representación en el dataset. Sin embargo, al añadir los pesos, esta tendencia se invierte, favoreciendo aquellos con un peso mayor. De esta forma, para algunas clases mejoran los resultados mientras que para otras empeoran. Además, hay que sumar el hecho de que en el dataset hay alguna bounding box mal etiquetada e imágenes en las que sólo hay dos o tres objetos etiquetados. La clave de usar imágenes panorámicas es la posibilidad de conocer el contexto completo de la escena, por lo que al utilizar imágenes prácticamente vacías de objetos no se le está enseñando correctamente a la red neuronal, afectando negativamente a los resultados obtenidos. Por último, hay que destacar el hecho de que utilizar pesos en una única tarea provoca cambios en los resultados obtenidos en la otra tarea. La estructura de la red neuronal se comparte en gran medida entre ambas tareas como se ha explicado en la Sección 2.1, por lo que la mayoría de pesos se comparten entre las ramas de detección y segmentación semántica. Esta es una de las grandes ventajas del modelo propuesto y lo que hace que modificar la función de coste de una rama afecte también a la otra.

Se puede concluir, por tanto, que con el dataset utilizado es mejor no utilizar pesos en las funciones de coste. Aunque con otros datasets con más imágenes y datos mejor etiquetados posiblemente mejoraría el desempeño de la red neuronal utilizando funciones de coste ponderadas.

Analizando en profundidad los resultados para la segmentación semántica se puede ver como hay varias clases muy bien segmentadas con un IoU>0.45, destacando especialmente el background, ya que la mayor parte del dataset es de esta clase. Esto ayuda a que los objetos segmentados tengan bordes bastante bien definidos y no se expandan demasiado por zonas que son background. Sin embargo, hay clases con una peor representación, aunque aceptable, con un IoU>0.3 y dos clases con muy bajo IoU: las lámparas y las estanterías. En ambos casos hay muy pocos píxeles en el conjunto de datos de entrenamiento. En el caso de las lámparas se añade, además, el hecho de que por lo general ocupan muy pocos píxeles en la imagen, por lo que aprender patrones asociados a ellas puede ser difícil. En cuanto a las estanterías, es la clase con menor presencia en el conjunto de datos de entrenamiento, lo cual se añade a la baja representación de píxeles, dando lugar a un IoU muy bajo (0.072). Con lo comentado, una posible solución para mejorar la segmentación semántica sería buscar otros datasets etiquetados con imágenes panorámicas, teniendo así más imágenes para poder aprender mejor los patrones de las distintas clases, así como volver a generar máscaras de segmentación sin errores para el dataset utilizado, aunque esto conllevaría un mayor esfuerzo.

En cuanto a la tarea de detección hay algunas clases con un elevado AP pese a no ser las clases con más representación en el dataset, como pueden ser las camas, los sofás, las televisiones, los cuadros o los espejos. Sorprende entre estas tres últimas clases que sea tan buena la detección ya que tienen formas y tamaños similares de forma que es fácil confundirse. Destaca también el elevado AP de las estanterías, las cuales, pese a ser la clase con menor representación en el dataset son detectadas con gran acierto, siendo totalmente contrario a la segmentación semántica donde el desempeño es muy pobre.

En cuanto a la comparación de resultados con Panoramic BlitzNet, si bien no son resultados del todo comparables debido a la diferencia de entrenamientos entre ambos trabajos, se demuestra que la implementación realizada es más que válida para las tareas que se requerían. En cuanto a la detección de objetos, el modelo implementado es mejor en todas las clases respecto a Panoramic BlitzNet. Sin embargo, esto se debe a que en esta implementación se le ha dado algo más de importancia a obtener valores altos de precision en detrimento del recall. Por cómo está definido el mAP, a mayor valor de precision mayor valor de mAP se obtiene aunque el recall disminuya. Esto es lo que ocurre en este caso: en Panoramic BlitzNet se da importancia a la precision pero mantienen un valor de recall más alto que en la implementación realizada en este trabajo, por lo que el AP para cada clase es menor. En el caso de la segmentación semántica, en general Panoramic BlitzNet presenta mejores resultados que la implementación realizada aunque hay algunas clases en las que el modelo implementado es superior. El hecho de los resultados de Panoramic BlitzNet sean mejores se debe a que en ese trabajo se realiza un pre-entrenamiento con otro dataset antes de entrenar el modelo con SUN360. Por este motivo, es probable que Panoramic BlitzNet haya aprendido a generalizar mejor la mayoría de clases. Sin embargo, puede ocurrir que en la implementación realizada en este trabajo el modelo se haya centrado más en clases con mayor presencia en SUN360 mientras que en Panoramic BlitzNet la red se haya centrado en clases con mayor representación en el dataset utilizado para el pre-entrenamiento. Esta podría ser una explicación al hecho de que la implementación realizada segmente mejor algunas clases.

Para concluir, se puede afirmar que los resultados obtenidos son satisfactorios, de forma que puedan servir como base para seguir investigando y mejorando el desempeño de las redes neuronales convolucionales en tareas de reconocimiento de objetos en imágenes panorámicas.

Capítulo 7

Conclusiones

El desarrollo de la red neuronal planteada para detección y segmentación semántica de objetos ha sido complicado por varios motivos: En primer lugar, la falta de experiencia y conocimientos en el campo de deep learning, de forma que ha habido un gran aprendizaje inicial antes de comenzar la implementación. Además, a lo largo del proyecto el aprendizaje ha sido gradual, entendiendo y aprendiendo cada vez más sobre los conceptos relacionados con las redes neuronales. En segundo lugar, el hecho de realizar la implementación en Tensorflow 2.x, frente a la implementación original de BlitzNet [18], en Tensorflow 1.x, ha dado lugar a muchos problemas a la hora de entender y trasladar el modelo original al código nuevo.

Durante el desarrollo del modelo se han realizado, además, algunos experimentos que han mostrado cómo los resultados obtenidos pueden variar modificando ligeramente algunos parámetros. Se ha analizado la importancia de elegir correctamente el umbral de nivel de confianza con el que se filtran las predicciones para conseguir unos resultados óptimos en función de lo que se pretenda conseguir con la red neuronal. Además, se ha analizado el efecto del class imbalance del dataset utilizado (SUN360) y las posibles formas de lidiar con él mediante cambios en las funciones de coste, concluyendo que debido a las propiedades de este dataset es mejor no utilizar pesos en las funciones de coste para conseguir mejores resultados.

Pese a los inconvenientes encontrados a lo largo del desarrollo del proyecto, se han conseguido resolver todos los problemas para cumplir el objetivo de obtener una implementación propia de una red neuronal que realice tareas de detección y segmentación semántica de objetos en imágenes panorámicas de interior. Puesto que los resultados obtenidos son satisfactorios, esta implementación podrá servir como base para seguir trabajando en este campo sin depender de códigos externos con licencias restrictivas.

7.1. Trabajo futuro

En cuanto al trabajo futuro, en primer lugar se podría revisar el etiquetado del dataset utilizado, especialmente de las etiquetas de segmentación semántica para arreglar errores y conseguir mejores resultados. También se podría hacer un estudio de posibles datasets etiquetados para imágenes panorámicas de interior que hayan surgido en los últimos años, pudiendo tener así más imágenes para entrenar el modelo y por tanto, mejorar el aprendizaje de patrones más generales para todas las clases de objetos a reconocer.

Por otro lado, sería interesante implementar el aprendizaje del modelo para realizar tareas de segmentación por instancias, pudiendo reconocer así cada objeto por separado en vez de por clase. Esto sería posible adaptando la red neuronal para poder trabajar con este tipo de segmentación, pues en el dataset utilizado existen etiquetas para realizar segmentación por instancias, por lo que se podría entrenar y evaluar el modelo.

Bibliografía

[1] F. Rosenblatt, “The perceptron: A probabilistic model for information storage and organization in the brain,” *Psychol. Rev.*, vol. 65, no. 6, pp. 386–408, 1958, doi: 10.1037/h0042519.

[2] S. Abirami and P. Chitra, “Energy-efficient edge based real-time healthcare support system,” *Adv. Comput.*, vol. 117, no. 1, pp. 339–368, Jan. 2020, doi: 10.1016/BS.ADCOM.2019.09.007.

[3] “What are Neural Networks? | IBM.” https://www.ibm.com/cloud/learn/neural-networks (accessed Oct. 27, 2021).

[4] K. O’Shea and R. Nash, “An Introduction to Convolutional Neural Networks,” 2015, Accessed: Oct. 17, 2021. [Online]. Available: http://arxiv.org/abs/1511.08458.

[5] V. Dumoulin and F. Visin, “A guide to convolution arithmetic for deep learning,” pp. 1–31, 2016, [Online]. Available: http://arxiv.org/abs/1603.07285.

[6] “A Comprehensive Guide to Convolutional Neural Networks — the ELI5 way | by Sumit Saha | Towards Data Science.” https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53 (accessed Oct. 27, 2021).

[7] A. Garcia-Garcia, S. Orts-Escolano, S. O. Oprea, V. Villena-Martinez, and J. Garcia-Rodriguez, “A Review on Deep Learning Techniques Applied to Semantic Segmentation.”

[8] “Compare Map Projections.” https://map-projections.net/ (accessed Oct. 27, 2021).

[9] C. Fernandez-Labrador, J. M. Facil, A. Perez-Yus, C. Demonceaux, J. Civera, and J. J. Guerrero, “Corners for Layout: End-to-End Layout Recovery from 360 Images,” *IEEE Robot. Autom. Lett.*, vol. 5, no. 2, pp. 1255–1262, 2020, doi: 10.1109/LRA.2020.2967274.

[10] T. F. Gonzalez, “ImageNet Classification with Deep Convolutional Neural Networks,” *Handb. Approx. Algorithms Metaheuristics*, pp. 1–1432, 2007, doi: 10.1201/9781420010749.

[11] O. Russakovsky *et al.*, “ImageNet Large Scale Visual Recognition Challenge,” *Int. J. Comput. Vis.*, vol. 115, no. 3, pp. 211–252, 2015, doi: 10.1007/s11263-015-0816-y.

[12] R. Girshick, J. Donahue, T. Darrell, and J. Malik, “Rich feature hierarchies for accurate object detection and semantic segmentation,” *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.*, pp. 580–587, 2014, doi: 10.1109/CVPR.2014.81.

[13] R. Girshick, "Fast R-CNN," *Proc. IEEE Int. Conf. Comput. Vis.*, vol. 2015 Inter, pp. 1440–1448, 2015, doi: 10.1109/ICCV.2015.169.

[14] S. Ren, K. He, R. Girshick, and J. Sun, "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 39, no. 6, pp. 1137–1149, 2017, doi: 10.1109/TPAMI.2016.2577031.

[15] K. He, G. Gkioxari, P. Dollár, and R. Girshick, "Mask R-CNN," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 42, no. 2, pp. 386–397, 2020, doi: 10.1109/TPAMI.2018.2844175.

[16] W. Liu *et al.*, "SSD: Single shot multibox detector," *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 9905 LNCS, pp. 21–37, 2016, doi: 10.1007/978-3-319-46448-0_2.

[17] C. Szegedy, S. Reed, D. Erhan, D. Anguelov, and S. Ioffe, "Scalable, High-Quality Object Detection," 2014, [Online]. Available: http://arxiv.org/abs/1412.1441.

[18] N. Dvornik, K. Shmelkov, J. Mairal, and C. Schmid, "BlitzNet: A Real-Time Deep Network for Scene Understanding," *Proc. IEEE Int. Conf. Comput. Vis.*, vol. 2017-Octob, pp. 4174–4182, 2017, doi: 10.1109/ICCV.2017.447.

[19] C.-Y. Fu, W. Liu, A. Ranga, A. Tyagi, and A. C. Berg, "DSSD : Deconvolutional Single Shot Detector," 2017, [Online]. Available: http://arxiv.org/abs/1701.06659.

[20] J. Guerrero-Viu, C. Fernandez-Labrador, C. Demonceaux, and J. J. Guerrero, "What's in my Room? Object Recognition on Indoor Panoramic Images," *Proc. - IEEE Int. Conf. Robot. Autom.*, pp. 567–573, 2020, doi: 10.1109/ICRA40945.2020.9197335.

[21] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.*, vol. 2016-Decem, pp. 770–778, 2016, doi: 10.1109/CVPR.2016.90.

[22] J. Xiao, K. A. Ehinger, A. Oliva, and A. Torralba, "Recognizing scene viewpoint using panoramic place representation," *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.*, pp. 2695–2702, 2012, doi: 10.1109/CVPR.2012.6247991.

[23] Y. Zhang, S. Song, P. Tan, and J. Xiao, "PanoContext: A whole-room 3D context model for panoramic scene understanding," *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 8694 LNCS, no. PART 6, pp. 668–686, 2014, doi: 10.1007/978-3-319-10599-4_43.

[24] "Multi-class Classification: Extracting Performance Metrics From The Confusion Matrix | by Serafeim Loukas | Towards Data Science." https://towardsdatascience.com/multi-class-classification-extracting-performance-metrics-from-the-confusion-matrix-b379b427a872 (accessed Oct. 27, 2021).

[25] M. Van den Bergh, X. Boix, G. Roig, and L. Van Gool, "SEEDS: Superpixels Extracted Via Energy-Driven Sampling," *Int. J. Comput. Vis.*, 2015, doi: 10.1007/s11263-014-0744-2.

[26] D. P. Kingma and J. L. Ba, "Adam: A method for stochastic optimization," *3rd Int. Conf. Learn. Represent. ICLR 2015 - Conf. Track Proc.*, pp. 1–15, 2015.

Anexos

Anexo A

Ampliación de resultados

En este Anexo se exponen resultados cuantitativos y cualitativos que amplían los resultados presentados en el Capítulo 6.

Experimentos

	Sin pesos		Pesos en Detección		Pesos en Segmentación		Pesos en ambas tareas	
	AP	IoU	AP	IoU	AP	IoU	AP	IoU
background	-	0.906	-	0.904	-	0.873	-	0.875
cuadro	0.899	0.632	0.922	0.645	0.917	0.525	0.908	0.523
cama	0.965	0.68	0.963	0.675	0.968	0.666	0.97	0.656
mesa	0.905	0.404	0.895	0.413	0.907	0.412	0.928	0.415
espejo	0.891	0.378	0.932	0.423	0.986	0.384	0.962	0.368
ventana	0.808	0.515	0.788	0.591	0.765	0.58	0.818	0.55
cortina	0.827	0.578	0.841	0.589	0.796	0.518	0.828	0.518
silla	0.888	0.302	0.899	0.303	0.918	0.277	0.896	0.277
lámpara	0.732	0.259	0.643	0.217	0.682	0.207	0.678	0.218
sofá	0.945	0.579	0.94	0.555	0.942	0.525	0.936	0.521
puerta	0.813	0.461	0.859	0.46	0.858	0.442	0.864	0.458
armario	0.631	0.306	0.646	0.294	0.662	0.315	0.635	0.318
mesilla	0.918	0.577	0.925	0.527	0.901	0.392	0.91	0.406
tv	0.963	0.569	0.961	0.594	0.962	0.496	0.956	0.504
estantería	0.603	0.072	0.503	0.047	0.474	0.061	0.598	0.099
Media	0.842	0.481	0.837	0.482	0.838	0.445	0.849	0.447

Tabla A.1: Efecto de utilizar pesos durante el entrenamiento: Resultados detallados (ampliando la Tabla 6.2) divididos por clases para diferentes usos de pesos en las funciones de coste durante el entrenamiento del modelo. En todos los casos se ha evaluado el modelo con un umbral de nivel de confianza de 0.8.

Resultados cualitativos

Figura A.1: Resultados cualitativos obtenidos con el modelo implementado evaluado sobre el conjunto de test de SUN360. El modelo ha sido entrenado sin usar pesos en las funciones de coste y evaluado con un umbral de nivel de confianza de 0.8.

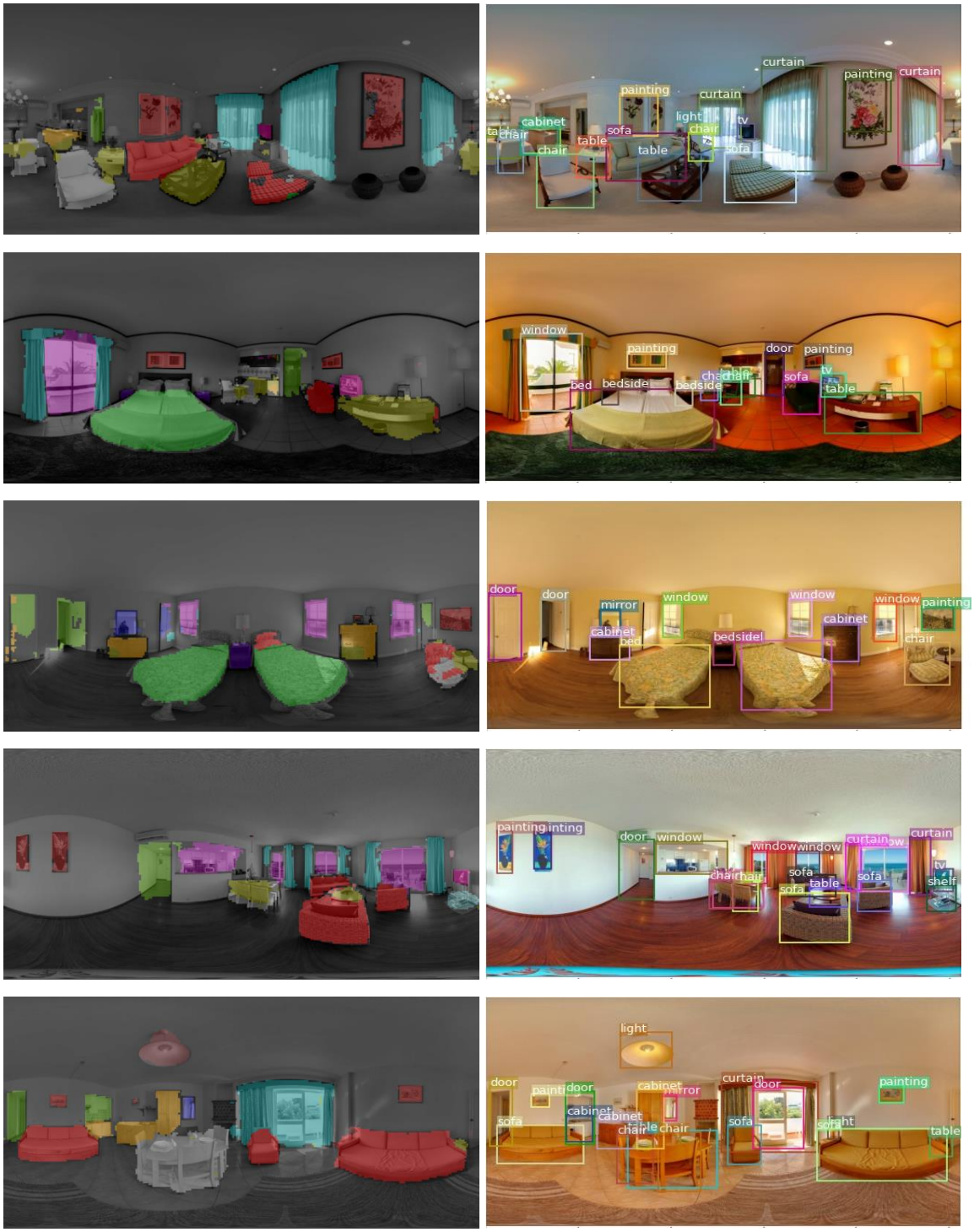


Figura A.2: Más resultados cualitativos obtenidos con el modelo implementado evaluado sobre el conjunto de test de SUN360. El modelo ha sido entrenado sin usar pesos en las funciones de coste y evaluado con un umbral de nivel de confianza de 0.8.

Anexo B

Gestión del proyecto

B.1. Software y Hardware utilizado

Durante el proyecto se han utilizado diferentes herramientas de software, las cuales se detallarán a continuación especificando la versión utilizada para ayudar en trabajos futuros. El sistema operativo utilizado ha sido Ubuntu 16.04 con CUDA 10.1 y cuDNN 7.6. Todo el modelo ha sido implementado en Python 3.8.5 utilizando Tensorflow 2.3.0. Además, se han utilizado otras librerías como NumPy 1.18.5, OpenCV 4.5, Tensorflow Addons 0.12.1 y Tensorflow Datasets 4.2.0.

Para realizar el entrenamiento de la red neuronal se ha utilizado una GPU Nvidia GeForce RTX 2080Ti prestada por el Departamento de Informática e Ingeniería de Sistemas de la Universidad de Zaragoza, aunque para algunas pruebas puntuales se ha utilizado una GPU Nvidia GeForce GTX 1660Ti propia.

B.2. Cronograma del proyecto

Todo el proyecto ha sido desarrollado dentro del grupo de investigación RoPeRT del I3A bajo la supervisión de mis tutores, José Jesús Guerrero Campo y Samuel Bruno Berenguel Baeta, con los que se han realizado reuniones periódicas para controlar el progreso del trabajo.

El proyecto comenzó en enero de 2021 y terminó en noviembre de 2021, con una duración de casi 10 meses, estando dividido el trabajo en 6 meses de prácticas en la Universidad de Zaragoza y 4 meses de realización de TFM sobre lo realizado en el periodo de prácticas. En la Figura B.1 se ha representado diagrama de Gantt del proyecto mientras que en la Tabla B.1 se ha representado en detalle las horas empleadas para cada tarea.

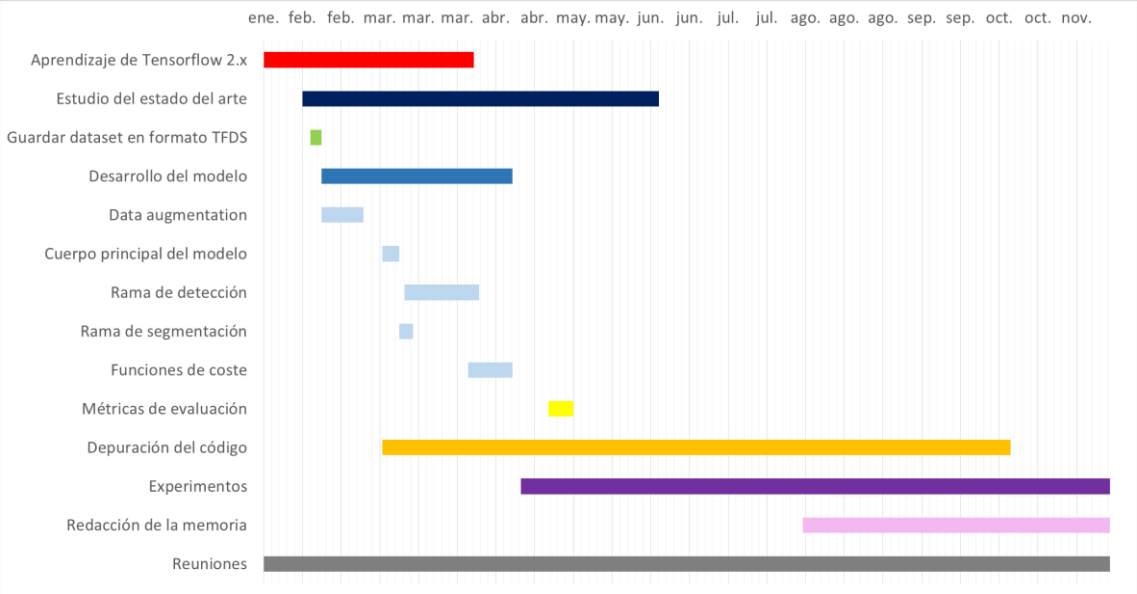


Figura B.1: Diagrama de Gantt del proyecto.

Tarea	Tiempo (horas)
Aprendizaje de Tensorflow 2.x	100
Estudio del estado del arte	70
Guardar dataset en formato TFDS	35
Desarrollo del modelo	225
Data augmentation	*80*
Cuerpo principal del modelo	*20*
Rama de detección	*75*
Rama de segmentación	*20*
Funciones de coste	*30*
Métricas de evaluación	25
Depuración del código	110
Experimentos	90
Redacción de la memoria	110
Reuniones	30
TOTAL	**795**

Tabla B.1: Información detallada de las horas empleadas en cada tarea del proyecto.